Computing and Informatics, Vol. 38, 2019, 1–18, doi: 10.31577/cai_2019_1_1

A NEW DYNAMIC LOAD BALANCING ALGORITHM FOR MULTI-ROIA

Dong Liu

Department of Computer Science
JiNan University
Guangzhou 510632, China
e-mail: wze2k@163.com

Abstract. Real-time Online Interactive Application (ROIA) is an emerging class of large-scale distributed application which can support millions of concurrent users around the world. Due to the dynamic changes in the number of concurrent users as well as the uncertainty of user operations, the dynamic load balancing is a key issue for ROIA. However, most of previous works are dedicated to the load balancing in a single ROIA without considering the variety of different type ROIAs. We take the advantage of differences between ROIAs and propose a new load balancing algorithm for multi-ROIA to improve the scalability of ROIA and increase the resource utilization of system. This paper firstly describes the motivation of the new load balancing algorithm, then presents the dynamic load balancing algorithm for multi-ROIA. Finally, the simulation results are also presented to show the efficiency and feasibility of the new algorithm.

Keywords: ROIA, cloud, MMOG, load balancing, scalability

Mathematics Subject Classification 2010: 68-W99

1 INTRODUCTION

Load balancing is one of the key issues for improving system performance and resource utilization in distributed and parallel computing, and can be divided into two categories: Static Load Balancing (SLB) and Dynamic Load Balancing (DLB). If the load can be determined and divided by a certain method before execution, it

belongs to SLB. But, if it can only keep monitoring the system load and dynamically adjusting the load while executing, it belongs to DLB.

Real-time Online Interactive Application (ROIA) [1] is an emerging type of large-scale distributed application. The popular and market-relevant representatives of ROIA are Massively Multi-player Online Game (MMOG), as well as real-time training and e-learning based on high-performance simulation. Therefore, without losing its universality, this paper takes MMOG as a case study of ROIA.

In ROIA, due to dynamic changes in the number of concurrent users as well as the uncertainty of user operations, the loads of computing and communication are difficult to estimate before running. Therefore, SLB strategy is not suitable for ROIA. And many researchers have done some researches on DLB in ROIA. But most of these works are dedicated to DLB in a single ROIA without considering the variety of different ROIAs in some aspects, such as latency tolerance or interaction complexity. The resource utilization of the system and the ability to deal with load peaks still have some limitations. Therefore, ROIA providers have to overprovision their operating infrastructure to cope with the uncertain peak load, which leads to a low and inefficient resource utilization. On the contrary, if ROIA providers do not overprovision the infrastructure, which may acquire high and efficient resource utilization, but also may reduce the ability of the system to deal with an uncertain peak load. To address this problem, we analyze differences between different types of ROIA, and we take an advantage of these differences for complementing each other; we hope that the new system can achieve higher resource utilization and better ability to deal with the peak load.

In previous works [2], we proposed the Multi-ROIA Cloud Platform (MRCP), a new structure to achieve high scalability in ROIA. In this paper, we focus on the corresponding load balancing algorithm of MRCP. We first describe in Section 2 the differences of different type of ROIA, and our new load balancing algorithm is motivated by these differences. In Section 3, we propose the new load balancing algorithm in detail. The new algorithm uses a mixed strategy of the centralized and distributed strategy, and it uses a different strategy in a different layer. Finally, we present experimental results showing that the new system achieves higher resource utilization and better ability to deal with the uncertain peak load.

2 THE MOTIVATION OF NEW LOAD BALANCING ALGORITHM

ROIA contains many types, and the different type of ROIAs have some difference in latency tolerance, interaction complexity and so on. Motivated by these characteristics, this paper proposes a new load balancing algorithm (named Dynamic Load Balancing Algorithm for Multi-ROIA, DLBAM) to improve the scalability of ROIA, to increase the system resource utilization and enhance the ability to cope with load peaks. The specific characteristics are as follows.

2.1 Difference in Latency Tolerance of ROIA

Mark Claypool and other researchers [3, 4] found that different types of ROIA have different sensitivity to network delay because of a different type of a player action. For example, First-Person Shooting (FPS) games are more sensitive to network delay than Real-Time Strategy (RTS) games. With the increase of network delay, the performance of FPS games decline sharply. On the contrary, the performance of RTS games declines slowly with the increasing network delay. So FPS games need obviously a lower network delay to ensure the good user experience than RTS games need.

Based on this characteristic of ROIA, we can deploy a variety of ROIAs on the same hardware platform. If load balancing is needed, the difference in latency tolerance will be considered. The partial load of ROIA which has a low delay sensitive degree in the heavier loaded server will be migrated to other underloaded server. It can improve the resource utilization, and will not cause a great impact on the user experience.

2.2 Difference in Interaction Complexity of ROIA

ROIAs have differences not only in the latency tolerance but also in the frequency and scale of user interaction. Nae et al. [5] used interaction complexity to divide ROIAs by the frequency and scale of user interaction. Assuming that the number of users is n, the interaction complexity may range from $O(n)$ for ROIAs in which users are mostly solitary or the ROIA does not need to make many state changes or compute complex interactions (e.g. puzzle games), to $O(n^2)$ for ROIA in which many users are interacting individually, and to $O(n^3)$ for ROIA in which groups of many players are interacting.

According to this characteristic of ROIA, if we consider the different interaction complexity in the process of dynamic load balance, we can get more satisfactory results for load balancing. That is, when there is a need for load transfer, it should avoid migrating the load of ROIA which has high interaction complexity. Because such migration may lead to producing a new large communication overhead. Therefore the preference should be given to the ROIA which has a low interaction complexity.

2.3 Difference in Load Change of ROIA

Another feature of ROIA is that the variation of load has a certain regularity, and not causing any mess. Although using a single user on ROIA is rather subjective, but the huge number of users makes the load change of ROIA showing a certain regularity. Moreover, due to time zone differences, the use of ROIA in the various regions of the world is not the same. And it makes the differences in load changes of ROIAs depending which servers are located in the various time zone.

Vlad Nae and other researchers selected the popular MMOG (RuneScape [6]) as the object of study. They collected a lot of data of RuneScape servers and statistically analyzed the variation of the simultaneous online users in RuneScape servers. Through the analysis of literature [5], we can see that, on the one hand, the load of ROIA changes obviously, and the difference between upper and lower load peak is huge, so it needs to have good scalability to improve the system resource utilization; on the other hand, generally, the load of ROIA is showing certain rules of change. This is propably affected by the time of day, the load is roughly in accordance with the day cycle up and down fluctuation. Moreover, the load peak is associated with the local time, so that the load peaks of different regions in different time zones generally do not appear in the same time period.

This characteristics is favorable for improving the degree of system resources utilization. If several resource centers are deployed in the different time zones in the world, due to the load peaks of the different resource centers at different times, then the part load can be migrated between the different resource centers.

3 DYNAMIC LOAD BALANCING ALGORITHM FOR MULTI-ROIA

Based on the characteristics of ROIA analyzed above, this section presents a new dynamic load balancing algorithm for multi-ROIA (DLBAM) to enhance the ability to cope with the peak load and to improve the utilization rate of system resources.

3.1 Classification of ROIA and Basic Idea of Algorithm

In order to facilitate the new algorithm, we firstly need to properly classify the ROIAs according to the above mentioned characteristics.

3.1.1 Classification of ROIA

In order to facilitate the load migration, this paper divides ROIA into two categories (named by Dynamic ROIA and Static ROIA) based on the characteristics of ROIA.

Dynamic ROIA refers to the ROIAs which have high latency tolerance and low interaction complexity. Because of the high latency tolerance, it will not impact the user experience when the part load is migrated to another resource center and the delay increased. Furthermore, because of the low interaction complexity, it will not increase the traffic between the two resource centers when the part load migrated to another resource center. So, the part load of dynamic ROIAs is suitable for migration between the resource centers which are located in different places.

Static ROIA refers to the ROIAs which have low latency tolerance and high interaction complexity. Because of the low latency tolerance, it will greatly impact the user experience when the delay is increased. And due to the high interaction complexity, the traffic between the resource centers will greatly increase when the part load is migrated to another resource center. So, this type of ROIA is not suitable for load migration between the resource centers. It is worth noting that the

static ROIAs refer to ROIAs not suitable for migration between the resource centers but saying that we are not saying that they cannot be migrated. They also may be migrated between the internal servers in a local resource center.

Table 1 summarizes the classification of ROIA in the new algorithm.

Name	Latency Tolerance	Ideal Delay Threshold	Interaction Complexity
Dynamic ROIA	High	1 000 ms	$O(n)$
Static ROIA	Low	< 500 ms	$> O(n * logn)$

Table 1. Classification of ROIA in DLBAM

3.1.2 Basic Idea of DLBAM

According to the analysis of ROIA's load changes in the above section, the load generally fluctuates by the cycle of one day. And because of the difference of time zone, the emergence times of load peak in different resource center are different. For example, in Figure 1, if there is a load peak in place A, the servers in place C may just have a low peak load. So, it shows a complementary characteristics.

If using the complementary characteristics, each resource center will not need to deploy enough hardware resources to deal with the peak load, it is enough just to deploy appropriate hardware resources. When the high peak load is coming, it can migrate a part of load to the resource center which is in the low peak at that moment.

Moreover, this paper divides ROIA into Dynamic ROIAs and Static ROIAs based on the characteristics of ROIA. The mixing deployment of these two kinds of ROIAs ensures the feasibility of load migrating between resource centers.

According to the idea, this paper presents a new hierarchical balancing algorithm. There are three layers of load balancing in the algorithm. The bottom layer is the load balancing inside of each ROIA in each resource center. The middle layer is the load balancing between ROIAs in each resource center. And the top layer is responsible for the load balancing between the resource centers.

The DLBAM algorithm uses a mixed strategy of the centralized and distributed strategy, and it uses a different strategy in a different layer. In the bottom and middle layer, it uses the centralized strategy, and the distributed strategy is taken in the top layer.

Figure 1 shows the structure of the system using DLBAM. In Figure 1, there are four resource centers deployed in globally distributed four locations. Each resource center has deployed some Dynamic ROIAs and some Static ROIAs at the same time. Each ROIA has a ROIA Scheduler which is responsible for load balancing between the internal servers of each ROIA. In addition, ROIA Scheduler will apply for new resources or release occupied resources to MRCP Local Controller (MLC) according to the load condition of servers in the ROIA. MLC is responsible for the load balancing between the ROIAs in the local resource center and also responsible

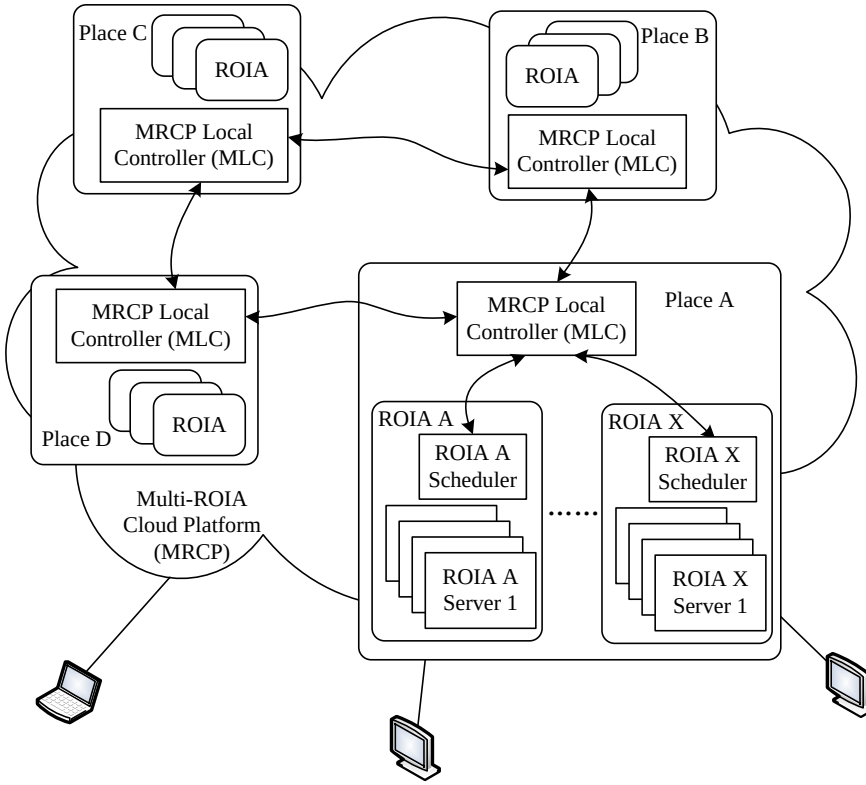


Figure 1. Structure diagram of the system using DLBAM

for putting forward the load migration application to the MLC of other resource center when it is necessary.

3.2 Related Definition and Formula

Before introducing the DLBAM algorithm, some variables are defined as follows:

Assumed that there are n ROIAs in the local resource center, and the current number of the k^{th} ROIA's servers is SN_k. The k^{th} ROIA has AN_k avatars and the number of other entities (such as non-player characters, NPC) is represented by EN_k.

The server e of ROIA k is indicated by S_e^k; the avatar i in ROIA k is indicated by a_i^k. And $a_i^k \in S_e^k$ shows that the avatar a_i^k is in the server S_e^k.

$C(a_i^k)$ represents the calculating cost of avatar a_i^k, such as state updating, game logic computing, environment rendering.

$I(a_i^k, a_j^k)$ represents the amount of information exchanged between the avatar a_i^k and a_j^k $(i \neq j)$. When a_i^k and a_j^k are in the same server, the interaction between a_i^k and a_j^k will only increase the computing cost without traffic cost. Only when a_i^k and a_j^k are in the different server, the interaction between them will increase the traffic load.

$V(x)$ represents the computational load increased by the interaction between the avatars in the same server, and $W(x)$ represents the traffic load caused by the interaction between the avatars in the different servers. x is the amount of interaction information.

Let $M_{ROIA}(S_e^k)$ denote the basic amount of memory used for the ROIA server S_e^k under no user links situation. $M_{as}(a_i^k)$ is the amount of memory used for the information of avatar a_i^k. And m_{es} is the amount of memory used for the information of one NPC entity.

The hardware resources which have great impact on the ROIA performance are mainly CPU, network bandwidth and memory. Therefore, the three main aspects needed to be considered when performing load balance: the computational load, traffic load and memory load.

The computational load of server S_e^k can be represented by Equation (1):

$$LC(S_e^k) = \sum_{a_i^k \in S_e^k} C(a_i^k) + \sum_{a_i^k, a_j^k \in S_e^k} V(I(a_i^k, a_j^k)) \qquad (i \neq j). \tag{1}$$

The first part of Equation (1) is the computing cost sum of server S_e^k, such as for state updating, game logic computing, environment rendering, etc. The last part is the sum of computing cost caused by interaction between avatars in the server S_e^k.

The traffic load of server S_e^k can be represented by Equation (2):

$$LN(S_e^k) = \sum_{d=1 \cap d \neq e}^{SN_k} \Big(\sum_{a_i^k \in S_e^k} \sum_{a_j^k \in S_d^k} W(I(a_i^k, a_j^k)) \Big). \tag{2}$$

The memory load of server S_e^k can be represented by Equation (3):

$$LM(S_e^k) = \sum_{i=1}^{AN_k} M_{as}(a_i^k) + EN_k \cdot m_{es} + M_{ROIA}(S_e^k). \tag{3}$$

So, the total load on server can roughly be estimated by Equation (4):

$$L(S_e^k) = LC(S_e^k) + LN(S_e^k) + LM(S_e^k). \tag{4}$$

Assume that $LC_{server}(S_e^k)$, $LN_{server}(S_e^k)$ and $LM_{server}(S_e^k)$ respectively are the max computational load, traffic load and memory load of server S_e^k without degrading the user experience. So, the resource utilization of server S_e^k can be estimated

by Equations (5),(6) and (7), respectively.

$$UC(S_e^k) = \frac{LC(S_e^k)}{LC_{server}(S_e^k)}, \tag{5}$$

$$UN(S_e^k) = \frac{LN(S_e^k)}{LN_{server}(S_e^k)}, \tag{6}$$

$$UM(S_e^k) = \frac{LM(S_e^k)}{LM_{server}(S_e^k)}. \tag{7}$$

Let $Threshold_{over}$ denote the server overload threshold. When the resource utilization of the server exceeds this threshold, it shows that the load of the server is too much and asks for load migration. In addition, because dynamic ROIA and static ROIA have a great difference in latency tolerance, therefore they may have different overload threshold.

$$Threshold_{over} = \begin{cases} 1, & \text{dynamic ROIA,} \\ 0.95, & \text{static ROIA.} \end{cases} \tag{8}$$

Let $Threshold_{light}$ denote the underloading threshold. When the server resource utilization is under $Threshold_{light}$, it shows that the server load is too light. To avoid the problem of load jitter between servers, it uses the threshold $Threshold_{top}$ for setting the upper limit of the server resource utilization after load balancing.

3.3 The Internal Load Balancing in Each ROIA

The internal load balancing works on each ROIA scheduler. Each one is responsible for load balancing between the servers of each ROIA. It will provide an application to MRCP Local Controller (MLC) for more hardware resources when the resources are not enough for this ROIA. Also, it will release some resources to MLC when there are excess idle resources in this ROIA.

The internal load balancing of ROIA k roughly shows as follows:

Step 1. Update the *ServerList* which contains the information of servers in ROIA k, compute the resource utilization (denoted by U_i^k) of each server in ROIA k and update the *ServerList* in descending order.

Step 2. Select the first server (S_x^k) from $ServerList$, if the resource utilization (U_x^k) of S_x^k is greater than $Threshold_{Over}$ then continue to the next step, otherwise skip to Step 6.

Step 3. Depending on the load capacity required to migrate, select some appropriate servers from the tail of *ServerList*. And according to Equation (2) to estimate the cost of communication, the servers will be selected in accordance with the size of the communication cost in ascending order to form an alternative destination server list – *CanList*.

Step 4. If *CanList* is empty, the ROIA Scheduler will apply to MRCP Local Controller for resources and jump to Step 6. Otherwise, choose the first server (denoted by S_y^k) from *CanList*, use Equation (4) to estimate the resource utilization of S_y^k (denoted by U_y^k), if U_y^k is greater than $Threshold_{Top}$ then remove S_y^k from *CanList* and repeat Step 4, otherwise continue to the next step.

Step 5. Migrate the load to S_y^k, update U_x^k, U_y^k and *ServerList*, and then jump to Step 2.

Step 6. Select the last server (S_x^k) from *ServerList*, if the resource utilization (U_x^k) is less than $Threshold_{Light}$ then continue to next step, else skip to Step 11.

Step 7. Whether the application is a local application, if it is to continue to the next step, otherwise ready to migrate back to the original resource center, and remove it from *ServerList*, then jump to Step 6.

Step 8. Depending on the load capacity which needs to be migrated, select some appropriate servers from the tail of *ServerList*. And use Equation (2) to estimate the cost of communication, the servers will be selected in accordance with the size of the communication cost in ascending order to form an alternative destination server list – *CanList*.

Step 9. If *CanList* is empty, then jump to Step 11. Otherwise, choose the first server (S_y^k) from *CanList*, use Equation (4) to estimate the resource utilization of S_y^k (U_y^k), if U_y^k greater than $Threshold_{Top}$ then remove S_y^k from *CanList* and repeat Step 9, otherwise continue to the next step.

Step 10. Migrate the load to S_y^k, release the resource of S_x^k, update U_y^k and *ServerList*, then jump to Step 6.

Step 11. Sleep a certain time, and then jump to Step 1.

The algorithm consists of two main parts: Step 2 to Step 5 in algorithm are the processing of overloading, and Step 6 to Step 10 in algorithm are the processing of underloading. The applications in local resource center are divided into two classes: local applications and external applications. The external applications are the applications migrated from other resource center which is overloading. For such applications, the processing of underloading is different with the local applications. The external applications prefer to migrate back to the original resource center, rather than to remain in this local resource center.

3.4 The Load Balancing Between ROIAs and Resource Centers

The load balancing between ROIAs and resource centers works on MRCP Local Controller. It mainly includes the processing of the local resources application, the processing of receiving the response to migration application, and the processing of receiving the application for resources release.

In the system, the applications for local resources can be divided into the following three categories:

1. The resource applications proposed by local static ROIA (*SAppList* represents the queue with such applications)
2. The resource applications proposed by local dynamic ROIA (*DAppList* represents the queue with such applications)
3. The foreign resource applications proposed by other resources center (*InMAppList* represents the queue with such applications)

These three types of resource applications have different priorities when they are processed. To minimize the resource migration between the resource centers the local resource application has its priority. Moreover, due to static ROIA has low latency tolerance, so it will give priority to the resource application proposed by the static ROIA.

In addition, the local resource center may send resource application to another resource center during the peak period. This type of application ensures migrating the load to other resource center, hence named the migration application.

In order to distinguish the above mentioned resource applications, it uses two additional queues: *OSAppList* and *ODAppList*. *OSAppList* stores static ROIA's resource applications which made the system to send migration application but temporarily has not received any response. *ODAppList* stores similar applications of dynamic ROIA.

Therefore, the priority order is as follows:

$$OSAppList > SAppList > ODAppList > DAppList > InMAppList.$$

The processing algorithm for resource applications roughly shows as follows. *FreeList* stores the relevant information of available resource in the local resource center.

Step 1. If $SAppList \neq \emptyset$, select the first resource item (SA_1) from *SAppList* and continue to the next step. Otherwise, jump to Step 4.

Step 2. If $FreeList \neq \emptyset$, then search the appropriate resource (S_x) for SA_1 in *FreeList*. And if found then continue to the next step, else move SA_1 from *SAppList* into *OSAppList*, send migration application and jump to Step 1.

If $FreeList = \emptyset$, move SA_1 from *SAppList* into *OSAppList*, send migration application and jump to Step 10.

Step 3. Assign the resource of (S_x) to SA_1 and delete SA_1 from *SAppList*. Then jump to Step 1.

Step 4. If $DAppList \neq \emptyset$, select the first resource item (DA_1) from *DAppList* and continue to next step. Otherwise, jump to Step 7.

Step 5. If $FreeList \neq \emptyset$, then search the appropriate resource (S_y) for DA_1 in *FreeList*. And if found then continue to the next step, else move DA_1 from *DAppList* into *ODAppList*, send migration application and jump to Step 1.

If *FreeList* $= \emptyset$, move DA_1 from *DAppList* into *ODAppList*, send migration application and jump to Step 10.

Step 6. Assign the resource of (S_y) to DA_1 and delete DA_1 from *DAppList*. Then jump to Step 1.

Step 7. If *InMAppList* $\neq \emptyset$, select the first item IMA_1 from *InMAppList* and continue to next step. Otherwise, jump to Step 10.

Step 8. If *FreeList* $\neq \emptyset$, then search the appropriate resource (S_z) for IMA_1 in *FreeList*. And if found then continue to the next step, else send message ("no appropriate resource") to the requester, delete IMA_1 from *InMAppList* and jump to Step 1.

If *FreeList* $= \emptyset$, send message ("no appropriate resource") to all the requester of *InMAppList*, delete all from *InMAppList* and jump to Step 10.

Step 9. Send message ("found appropriate resource") to the requester of IMA_1, delete S_z from *FreeList*, delete IMA_1 from *InMAppList* and jump to Step 1.

Step 10. Sleep a certain time, and then jump to Step 1.

In addition to the above-described algorithm, there is the processing of receiving the response to migration application, the processing of receiving the application for resource release, and so on, in the system.

4 SIMULATION AND RESULTS ANALYSIS

This section describes the simulation of DLBAM and the comparative analysis of relevant results.

4.1 Experimental Environment

Experiments using Python 3.0 ran a simulation system of DLBAM and traditional dynamic load balancing algorithm, and designed a simulation environment.

4.1.1 Simulation of Hardware Resources and Applications

In the experiment environment, it has simulated four resource centers in different time zones. Each center has deployed six different ROIA and the proportion of dynamic ROIA and static ROIA is 2 : 1. Each resource center has 180 servers and assumed each ROIA assigned 30 servers initially. For simplicity, the experiment mainly used the number of concurrent online users to simulate the load size. Under the premise to ensure good quality of service, the assumed maximum load for each server is 2 000 online users. If there are more than 2 000 users on one server, the quality of service begin to decline and may even crash in severe cases (but our experiment did not simulate the case of crash).

In addition, the traditional algorithm for comparison has the same simulation hardware resource. But in the traditional method, the six ROIAs are deployed independently, and there is no load balancing between the resource centers.

4.1.2 Simulation of Each Server Load

The experiment refers to the actual server load data in reference [5] and produces the fluctuant load data for each server. Moreover, the peak load time of each resource center (four resource centers are numbered by Center 0, Center 1, Center 2 and Center 3) is different. It simulates the impact of different time zones to the load changes.

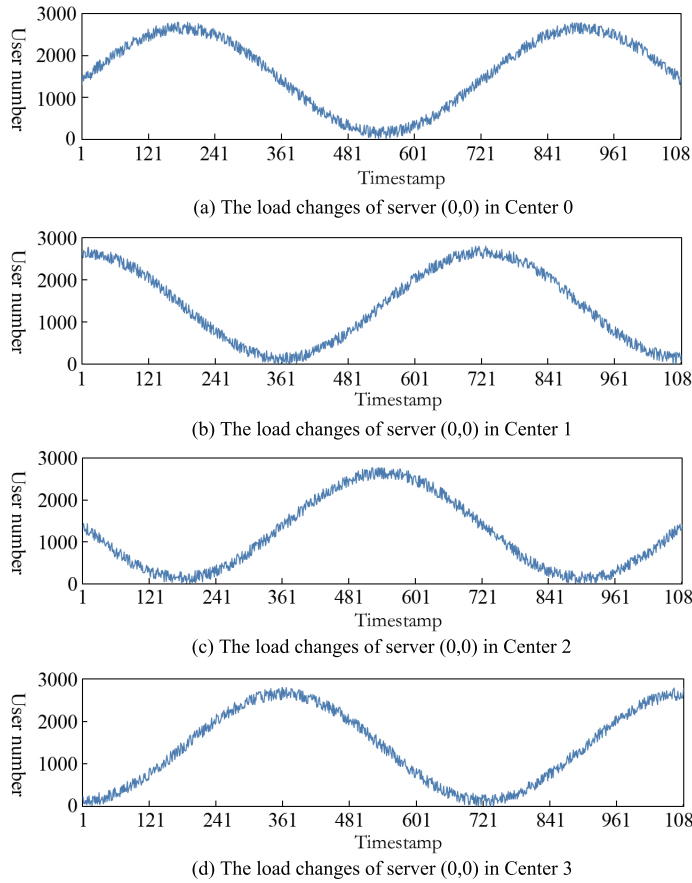


Figure 2. The simulation load changes of server $(0, 0)$ in 4 centers

Figure 2 shows the simulated load changes of the server $(0, 0)$ in each resource center. The ordinate unit of Figure 2 is the number of concurrent online users; the

horizontal axis unit is the timestamp. The interval between two time points in figure represents two minutes. Figure takes a total of 1081 data points of time, i.e., the figure shows the load variations during 36 hours ($1\,080 * 2/60 = 36$ hours).

4.2 Experimental Results and Analysis

The following results are all based on the simulate load change data which is described in Figure 2. Figure 3 shows the difference between the results of traditional algorithm and DLBAM based on the same server and same load input.

Figure 3 shows the load changes of server $(0, 0)$ in Center 2 during the traditional algorithm and DLBAM running. The blue line represents results of the traditional algorithm run. Due to the peak load of each server on the same resource center appears approximately at the same time and no migration between resource centers in the traditional algorithm, so it happens that no resources are available during the peak period, but too much resources are available during the idle period, as blue line shows. Moreover, the number of users exceeds 2 000 approximately during that time from 400 to 700. According to the pre-set experiments standard, it is indicating that the quality of service begins to decline and may even crash.

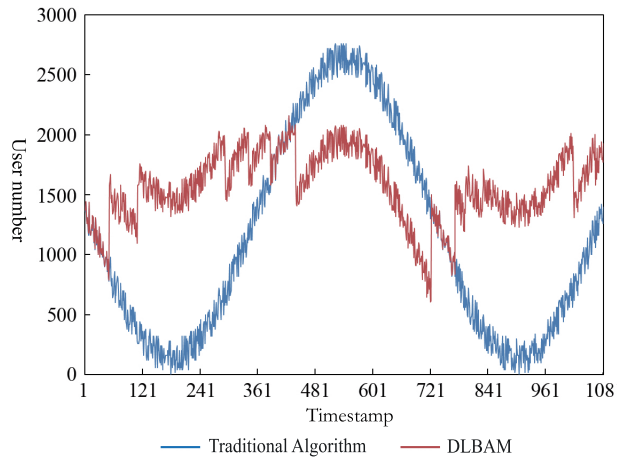


Figure 3. The load changes of server $(0, 0)$ in center 2 during algorithm running

On the contrary, due to having migration between centers in the DBLAM, the user number does not exceed 2 000 during the experiment time, hence the decline of quality of service is avoided. In addition, due to receiving the load of other centers, this server maintains relatively high resource utilization during the low peak period.

Figure 3 shows the difference between the traditional algorithm and DLBAM on one single server and Figure 4 shows the difference from the perspective of entire

resource center. In order to facilitate the representation, it takes the ratio of the actual user number to the maximum load as the ordinate (the ratio is multiplied by 100 in Figure 4). The maximum load is assumed having 2 000 online users for each server to ensure good quality of service. The ratio may reflect the resource utilization.

In Figure 4, it shows the ratio changes of each center during algorithm running. The red lines in the figure generally vary between 40 and 100. This indicates that it is not overloaded during the peak period and keeps some suitable load during the low peak period. But the blue lines generally vary between 5 and 130. That means it is overloaded during the peak period and a lot of free resources appears during the low peak period.

From the above analysis, it can be seen that the performance of DLBAM on a single server or on the whole center is better than the performance of traditional algorithm.

5 RELATED WORK

Because some of related work has been introduced in the previous section, only a few other related works about dynamic load balancing are introduced here.

Ren [7] proposed a dynamic load balancing algorithm for cloud computing on the basis of an existing algorithm called WLC (Weighted Least Connection) [8]. WLC assigned a new task based on the number of links on each node. Firstly, it calculated the number of links on each node in the cloud, and then selected the node with minimum links and assigned the task to the node. So WLC algorithm did not consider the other current situation of each node, such as CPU speed, storage capacity and network bandwidth, etc. Ren proposed an improved algorithm called ESWLC (Exponential Smooth Forecast based on Weighted Least Connection). ESWLC algorithm determined whether the node receives a new task after achieving a relative performance of the node, such as CPU power, memory performance, number of links etc.

Mehta and other researchers took a variety of distributed computing environments (such as cloud computing, grid and cluster) into account and proposed WCAP (Workload and Client Aware Policy) [9] based on content-aware dynamic load balancing algorithm. WCAP is a hybrid approach. However, the performance of WCAP in the real distributed environment (e.g. Hadoop) needs a further research verification.

Wang and other researchers proposed a hierarchy load balancing algorithm, called LBMM (Load Balancing Min-Min) [10], based on the OLB (Opportunistic Load Balancing) algorithm [11]. OLB algorithm is a static load balancing strategy for the purpose of keeping each node of cloud with a certain load. It does not consider the execution time of each node, which may lead to slow down of the task processing and also a bottleneck problem may appear. In order to solve these problems, LBMM algorithm takes three-layer architecture. The first layer is Request Manager

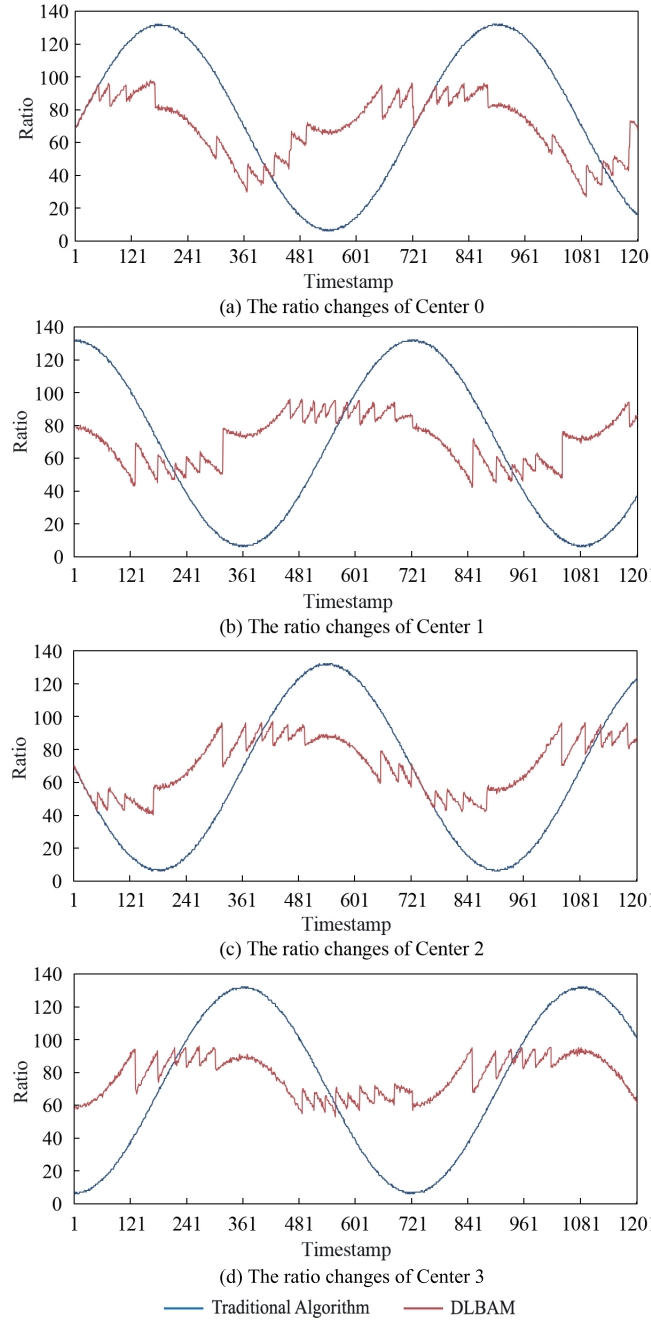


Figure 4. The ratio changes of centers during algorithm running

which is responsible for receiving task requests and assigning the task requests to a Service Manager. When Service Manager receives a task request, it divides it into some subtasks in order to accelerate the processing speed. After the division Service Manager assigns the subtask to service node which is responsible for the execution of subtask.

Bezerra [12] presented a load balancing strategy which uses KD-tree algorithm to dynamically divide the virtual world of ROIA. Kim [13] presented an adaptive load algorithm for solving the problem of traditional graph partitioning method in ROIA. Li [14, 15, 16] and Balogh [17] have done some related works on cloud computing security problems. Nguyen [18] presented a novel development and deployment framework for cloud distributed applications.

In addition, we have also done some preliminary related works on ROIA. Reference [2] proposed a first step of new approach to achieve high scalability in ROIA under cloud environment. And to solve some problem in the traditional Dead Reckoning algorithm, the reference [19] proposed an improved DR algorithm based on target-extrapolation in ROIA.

6 CONCLUSIONS

ROIA is an emerging class of large-scale distributed application which can support millions of concurrent users spread across the world. MMOG is one of the popular and market-relevant representatives of ROIA. Due to the dynamic changes in the number of concurrent users as well as the uncertainty of user operations, the dynamic load balancing is a key issue for ROIA. However, most of the previous works are dedicated to the load balancing in a single ROIA without considering the variety of ROIAs. We take the advantage of differences between ROIAs and propose a new load balancing algorithm for multi-ROIA to improve the scalability of ROIA and increase the resource utilization of the system.

This paper firstly describes the motivation of the new load balancing algorithm. ROIA contains a variety of types, and the different type ROIA has some differences in latency tolerance, interaction complexity, etc. We take an advantage of these characteristics to improve the scalability of ROIA. In Section 3, we present the basic idea of the new algorithm, related definition and formula and the main part of the dynamic load balancing algorithm for multi-ROIA. At the end we describe the simulation of DLBAM and the comparative analyses of relevant results.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61402197).

REFERENCES

[1] Glinka, F.—Raed, A.—Gorlatch, S.—Ploss, A.: A Service-Oriented Interface for Highly Interactive Distributed Application. In: Lin, H. X. et al. (Eds.): Euro-Par 2009 – Parallel Processing Workshops. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6043, 2010, pp. 266–277, doi: 10.1007/978-3-642-14122-5_31.

[2] Liu, D.—Zhao, Y.-L.: A New Approach to Scalable ROIA in Cloud. Proceedings of the 2013 4th Emerging Intelligent Data and Web Technologies (EIDWT), Xi'an, 2013, pp. 51–55, doi: 10.1109/EIDWT.2013.13.

[3] Claypool, M.—Claypool, K.: Latency and Player Actions in Online Games. Communications of the ACM, Vol. 49, 2006, No. 11, pp. 40–45, doi: 10.1145/1167838.1167860.

[4] Claypool, M.: The Effect of Latency on User Performance in Real-Time Strategy Games. Computer Networks, Vol. 49, 2005, No. 1, pp. 52–70, doi: 10.1016/j.comnet.2005.04.008.

[5] Nae, V.—Iosup, A.—Prodan, R.: Dynamic Resource Provisioning in Massively Multiplayer Online Games. IEEE Transaction on Parallel and Distributed Systems, Vol. 22, 2011, No. 3, pp. 380–395, doi: 10.1109/tpds.2010.82.

[6] Jagex Ltd.: RuneScape. `http://www.runescape.com/`, February 2014.

[7] Ren, X.—Lin, R.—Zou, H.: A Dynamic Load Balancing Strategy for Cloud Computing Platform Based on Exponential Smoothing Forecast. Proceedings of International Conference on Cloud Computing and Intelligent Systems (CCIS), Beijing, IEEE, 2011, pp. 220–224, doi: 10.1109/ccis.2011.6045063.

[8] Lee, R.—Jeng, B.: Load-Balancing Tactics in Cloud. Proceedings of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Beijing, 2011, pp. 447–454, doi: 10.1109/cyberc.2011.79.

[9] Mehta, H.—Kanungo, P.—Chandwani, M.: Decentralized Content Aware Load Balancing Algorithm for Distributed Computing Environments. Proceedings of the 2011 International Conference and Workshop on Emerging Trends in Technology (ICWET '11), Mumbai, 2011, pp. 370–375, doi: 10.1145/1980022.1980102.

[10] Wang, S.-C.—Yan, K.-Q.—Liao, W.-P.—Wang, S.-S.: Towards a Load Balancing in a Three-Level Cloud Computing Network. Proceedings of the 3rd International Conference on Computer Science and Information Technology (ICCSIT), New York, 2010, pp. 108–113, doi: 10.1109/iccsit.2010.5563889.

[11] Sang, A.—Wang, X.—Madihian, M. et al.: Coordinated Load Balancing, Handoff/Cell-Site Selection, and Scheduling in Multi-Cell Packet Data Systems. Wireless Networks, Vol. 14, 2008, No. 1, pp. 103–120, doi: 10.1007/s11276-006-8533-7.

[12] Bezerra, C. E. B.—Comba, J. L. D.—Geyer, C. F. R.: Adaptive Load-Balancing for MMOG Servers Using KD-Trees. Computers in Entertainment, Vol. 10, 2012, No. 3, Art. No. 5, doi: 10.1145/2381876.2381881.

[13] Kim, T.-H.: Adaptive Load Partitioning Algorithm for Massively Multiplayer Online Games. In: Kim, K., Chung, K. Y. (Eds.): IT Convergence and Security 2012. Springer, Dordrecht, Lecture Notes in Electrical Engineering, Vol. 215, 2013, pp. 383–391, doi: 10.1007/978-94-007-5860-5_47.

[14] Li, J.—Wang, Q.—Wang, C.—Cao, N.—Ren, K.—Lou, W.: Fuzzy Keyword Search over Encrypted Data in Cloud Computing. Proceedings of The 29th Conference on Computer Communications (INFOCOM), IEEE, 2010, pp. 441–445, doi: 10.1109/INFCOM.2010.5462196.

[15] Li, J.—Chen, X. F.—Li, M.—Li, J.—Lee, P. P. C.—Lou, W.: Secure Deduplication with Efficient and Reliable Convergent Key Management. IEEE Transactions on Parallel and Distributed Systems, Vol. 25, 2014, No. 6, pp. 1615–1625, doi: 10.1109/tpds.2013.284.

[16] Li, J.—Chen, X. F.: Efficient Multi-User Keyword Search over Encrypted Data in Cloud Computing. Computing and Informatics, Vol. 32, 2013, No. 4, pp. 723–738.

[17] Balogh, Z.—Gatial, E.—Hluchý, L.—Toegl, R.—Pirker, M.—Hein, D.: Agent-Based Cloud Resource Management for Secure Cloud Infrastructures. Computing and Informatics, Vol. 33, 2014, No. 6, pp. 1333–1355.

[18] Nguyen, B. M.—Tran, V.—Hluchý, L.: A Generic Development and Deployment Framework for Cloud Computing and Distributed Applications. Computing and Informatics, Vol. 32, 2013, No. 3, pp. 461–485.

[19] Liu, D.: An Improved DR Algorithm Based on Target Extrapolating in ROIA Cloud Platform. International Journal of Distributed Sensor Networks, Vol. 9, 2013, No. 12, Art. No. 637328, 8 pp., doi: 10.1155/2013/637328.

Liu Dong works in the Computer Science Department of JiNan University. His research interests include resource management and monitoring in distributed systems.

Computing and Informatics, Vol. 38, 2019, 19–56, doi: 10.31577/cai_2019_1_19

INCORPORATING STRATIFIED NEGATION INTO QUERY-SUBQUERY NETS FOR EVALUATING QUERIES TO STRATIFIED DEDUCTIVE DATABASES

Son Thanh Cao

School of Engineering and Technology, Vinh University
182 Le Duan street, Vinh, Nghe An, Vietnam
&
Nguyen Tat Thanh University, Ho Chi Minh City, Vietnam
e-mail: sonct@vinhuni.edu.vn

Linh Anh Nguyen*

Division of Knowledge and System Engineering for ICT
Faculty of Information Technology
Ton Duc Thang University, Ho Chi Minh City, Vietnam
&
Institute of Informatics, University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
e-mail: nguyenanhlinh@tdtu.edu.vn, nguyen@mimuw.edu.pl

Abstract. Most of the previously known evaluation methods for deductive databases are either breadth-first or depth-first (and recursive). There are cases when these strategies are not the best ones. It is desirable to have an evaluation framework for stratified Datalog$^\neg$ that is goal-driven, set-at-a-time (as opposed to tuple-at-a-time) and adjustable w.r.t. flow-of-control strategies. These properties are important for efficient query evaluation on large and complex deductive databases. In this paper, by incorporating stratified negation into so-called *query-subquery nets*, we develop an evaluation framework, called QSQN-STR, with such properties for evaluating queries to stratified Datalog$^\neg$ databases. A variety of flow-of-control strategies can be used for QSQN-STR. The generic evaluation method QSQN-STR for stratified Datalog$^\neg$ is sound, complete and has a PTIME data complexity.

Keywords: Deductive databases, datalog with negation, query processing

* corresponding author

Mathematics Subject Classification 2010: 68N17

1 INTRODUCTION

Datalog is a well-known rule-based query language for deductive databases. In [24], Huang et al. wrote *"we are witnessing an exciting revival of interest in recursive Datalog queries in a variety of emerging application domains such as data integration, information extraction, networking, program analysis, security, and cloud computing"* (see also, e.g., [23, 7]). Datalog expresses the Horn fragment with the safety condition[1] and without function symbols of first-order logic and uses the traditional monotonic semantics. The extension Datalog$^\neg$ of Datalog allows expressing non-monotonic queries by using negation in the bodies of program clauses. It uses a non-monotonic semantics like the standard semantics for stratified Datalog$^\neg$ programs and the well-founded semantics for the general case. A Datalog$^\neg$ program is stratifiable if it can be divided into strata such that, if a negative literal of a predicate p occurs in the body of a program clause in a stratum, then the clauses defining p must belong to an earlier stratum. A deductive database consists of a Datalog/Datalog$^\neg$ program (for defining intensional predicates) and an instance of extensional predicates.

This work studies query processing for stratified Datalog$^\neg$ databases. The topic is worthy of consideration due to practical applications of deductive databases.

1.1 Related Work

Researchers have developed a number of evaluation methods for Datalog databases, such as QSQ [43, 1], QSQR [43, 31], QoSaQ [44] and Magic-Sets [5, 6] (by Magic-Sets we mean the evaluation method that combines the magic-set transformation with the improved semi-naive evaluation method).

QSQ (Query-Subquery) [43, 1] is a framework for evaluating queries to Datalog databases. Its approach is top-down (i.e., query processing is closely related to the main goal) and set-at-a-time (i.e., operations are set-oriented but not tuple-oriented). It implements a *tabulation* (tabling/memoing) technique by using so-called *input*, *answer* and *supplement* relations to guarantee termination. *Adornments* for intensional predicates (and their corresponding *input* and *answer* relations) are used to enable exploiting relational operations like join and projection. In general, QSQ uses adornments to simulate SLD-resolution in pushing constant symbols from goals to subgoals. An enhanced version of QSQ, called *annotated QSQ*, also uses *annotations* to simulate SLD-resolution in pushing repeats of variables from goals to subgoals. A variety of flow-of-control strategies (which are similar to search strategies and called *control strategies* for short) can be used for QSQ.

[1] For a definition of the safety condition, see the paragraph after Definition 1.

QSQR (QSQ Recursive), introduced by Vieille in [43], is a query evaluation method for Datalog databases that follows the QSQ approach and uses a recursive strategy. Roughly speaking, the strategy is *depth-first*, but due to tabulation, as observed by Vieille [44], the QSQR approach is like iterative deepening search. The versions of QSQR presented in [43, 1] are incomplete [31, 44, 29]. This is corrected in [29] by using an outer loop that clears global *input* relations for each iteration.

In [44], Vieille introduced another method, called QoSaQ, for evaluating queries to Datalog databases. It is an adaptation of SLD-AL resolution. The method can be implemented as a set-oriented procedure, but as stated by Vieille himself, the practical interest of the method lies in its one-inference-at-a-time basis, as opposed to the set-at-a-time approach. The intention is to permit an advanced analysis of the duplicate elimination issue.

The magic-sets technique [5, 6] simulates the top-down QSQ approach by rewriting the Datalog program together with the given query to another equivalent one that when evaluated using a bottom-up technique (e.g., the improved semi-naive evaluation) produces only facts produced by the QSQ evaluation. Adornments are used as in the QSQ approach. To simulate annotations, the magic-sets transformation is augmented with subgoal rectification (see, e.g., [1]).

In [11, 9], we provided a framework called QSQN (Query-Subquery Nets) for evaluating queries to Horn knowledge bases. It uses a parameter for the limit on the nesting depths of terms occurring in the computation. When this limit is set to 0, the framework can be used for evaluating queries to Datalog databases. QSQN is an adaptation and a generalization of the QSQ approach for Horn knowledge bases. One of the key differences is that it does not use adornments and annotations, but uses substitutions instead. This is natural for the case with function symbols and without the safety condition. Like QSQ, every control strategy can be used for QSQN. The notion of query-subquery net makes a linkage to flow networks and is intuitive for developing efficient evaluation algorithms.

A top-down approach with tabulation for dealing with stratified Datalog$^\neg$ was proposed in [26, 41, 37]. The evaluation procedures given in [26, 41, 37] are similar to each other, with some differences as discussed in [37]. They are called "QSQR/SLS-procedure" in [26, 37] and we will refer to them as the QSQR/SLS method. This method relies on using a derivation forest (of global SLS-resolution) with tabulation and is implemented using the recursive approach like QSQR.

In [37], apart from QSQR/SLS, Ross also proposed a bottom-up evaluation method for stratified Datalog$^\neg$ by presenting a magic-sets transformation, which simulates the top-down QSQR/SLS method, but the program obtained from the transformation can be evaluated using a bottom-up technique. Programs obtained from the transformation are not stratified Datalog$^\neg$ programs, as they use special "literals" for checking whether the computation of the corresponding negative goals has been completed.

In [4], Balbin et al. proposed another bottom-up evaluation method for stratified Datalog$^\neg$. Their method applies a magic-sets transformation and a bottom-up computation with recursive calls for evaluating negative goals.

The well-founded semantics is a commonly accepted choice for (general) Datalog$^\neg$, as it coincides with the standard semantics for stratified Datalog$^\neg$, and using it Datalog$^\neg$ has a PTIME data complexity. This semantics was first introduced by Van Gelder et al. for normal logic programs [19] and can be characterized by the alternating fixpoint [18]. Several calculi for normal logic programs that are sound and complete w.r.t. the well-founded semantics have been developed. One of them is SLG-resolution. In [15], Chen et al. presented efficient techniques for implementing SLG-resolution. Their method maintains positive and negative dependencies among subgoals in a top-down evaluation, detects positive and negative loops, delays subgoals when possible loops occur, checks completion of subgoals and resumes their activeness when possible. It is tuple-oriented and its implementation XSB [40] can be used as an engine for in-memory Datalog$^\neg$ databases.

Kemp et al. [25] and Morishita [30] proposed bottom-up evaluation methods for Datalog$^\neg$ under the well-founded semantics. Their methods are based on Van Gelder's alternating fixpoint characterization and use a magic-sets transformation with adornments but without annotations.

In [14], together with a colleague we extended QSQN to obtain a method called QSQN-WF for evaluating queries to Datalog$^\neg$ databases under the well-founded semantics. It follows Przymusinski's SLS-resolution [34], with Van Gelder's alternating fixpoint semantics [18] on the background, but uses a query-subquery net to implement tabulation and the set-at-a-time technique.

1.2 Motivations

We first discuss some important aspects of query evaluation (in Sections 1.2.1–1.2.3), and then state motivations of our work (in Section 1.2.4).

1.2.1 Adjustability w.r.t. Control Strategies

The techniques used for query evaluation are usually separated into two classes depending on whether they focus on top-down or bottom-up evaluation [1]. Here, "top-down" is understood as "*goal-driven*" (i.e., query processing is relevant to the subgoals and therefore closely related to the main goal). As the bottom-up evaluation methods based on the magic-sets technique simulate the top-down evaluation, they are also goal-driven. Since the terms "top-down" and "bottom-up" are antonyms, it is better to classify top-down evaluation as goal-driven and characterize bottom-up evaluation methods by an additional property. Being goal-driven can be treated as a requirement for efficient evaluation methods.

The aforementioned bottom-up evaluation methods for Datalog [5, 6], stratified Datalog$^\neg$ [37, 4] and Datalog$^\neg$ [25, 30] use a magic-sets transformation and a bottom-up computation like the improved semi-naive evaluation. So, they can be characterized as goal-driven and *breadth-first* (i.e., based on using a breadth-first control strategy).[2] On the other hand, the top-down evaluation methods QSQR [43, 31]

[2] The naive evaluation can be described as follows: repeat applying all of the rules

and QoSaQ [44] (for Datalog), QSQR/SLS [26, 41, 37] (for stratified Datalog$^\neg$) and SLG-resolution [15, 40] (for normal logic programs and Datalog$^\neg$ databases) can be characterized as goal-driven and *depth-first*.[3] The frameworks QSQ [43, 1] (for Datalog), QSQN [11, 9] (for Horn knowledge bases and Datalog databases) and QSQN-WF [14] (for Datalog$^\neg$) follow the goal-driven approach but allow *every* control strategy.

The breadth-first and depth-first approaches are just two among possible approaches. There are cases when they are not the best ones [11]. When developing a framework for query evaluation one should make it general to a certain extent so that a variety of control strategies can be used. In particular, it is desirable to be able to control the computation flow dynamically.

1.2.2 Set-at-a-Time vs. Tuple-at-a-Time

The evaluation methods QoSaQ [44] (for Datalog) and SLG-resolution [15, 40] (for normal logic programs and Datalog$^\neg$ databases) are tuple-at-a-time (tuple-oriented). They use complex data structures for handling individual subgoals (tuples), and when the extensional relations and the search space are too large, in-memory computation may be impossible. XSB [40] is an efficient engine for in-memory deductive databases due to the suspension-resumption mechanism, advantages of WAM (Warren Abstract Machine) and other optimizations. Such techniques are highly tuple-oriented. When the extensional relations are too large and the program defining intensional predicates is sophisticated, accesses to the secondary storage may be unavoidable, and the set-at-a-time approach is preferable.

Regarding the evaluation methods QSQR [43, 31] (for Datalog) and QSQR/SLS [26, 41, 37] (for stratified Datalog$^\neg$), they can be implemented using either the tuple-at-a-time approach or the set-at-a-time approach. But, using the latter one the recursive strategy is unavoidable. As observed in [29, Remark 3.2], using the recursive approach, *input* relations should be cleared occasionally (e.g., at the beginning of each iteration of the main loop) in order to allow recomputations using updated *answer* relations. This causes redundant computations.

1.2.3 Why Are Evaluation Methods for Stratified Datalog$^\neg$ Needed?

The question is rather "are the known evaluation methods for (general) Datalog$^\neg$ efficient for evaluating queries to stratified Datalog$^\neg$ databases?". The general answer is "they are not as efficient as expected for that kind of tasks". The reason is that they were developed to cope with unstratified negation and are thus superfluous. For example, the methods proposed in [25, 30, 14] are based on Van Gelder's

sequentially, one after the other, until no new facts were derived during the last iteration. Its approach is like breadth-first search. The improved semi-naive evaluation (see, e.g., [1]) shares this property.

[3] The mentioned methods use a recursive control strategy, which is like the depth-first search strategy implemented using recursive calls.

alternating fixpoint characterization and use an (additional) outer loop to guarantee that an alternating fixpoint can be reached. When applied to stratified Datalog$^\neg$, that causes certain redundant (re)computations.

Apart from the well-founded semantics, the stable model semantics [20] is also a well-known semantics for normal logic programs (see, e.g., the survey [3]). These two semantics coincide for certain classes of logic programs [35, 16, 21], including stratified logic programs and stratified Datalog$^\neg$ programs. The stable model semantics is used for answer set programming (ASP), and systems like DLV [27], NP Datalog [22] and *clasp* [17], which deal among others with ASP, can be used for answering queries to stratified Datalog$^\neg$ databases. However, as the main aim of ASP engines is to find an answer set (i.e., a stable model) for a given logic program, they are not goal-driven and, in general, not as efficient as expected for answering queries to stratified Datalog$^\neg$ databases.

1.2.4 The Need for a New Evaluation Framework for Stratified Datalog$^\neg$

As discussed in Sections 1.1–1.2.3, the previously known methods that can be used for evaluating queries to stratified Datalog$^\neg$ databases are:

- breadth-first [4, 37, 25, 30] or depth-first [26, 41, 37, 15, 40]; or/and
- tuple-at-a-time [15, 40]; or/and
- designed for (general) Datalog$^\neg$ [25, 30, 14] or normal logic programs [15, 40], and not as efficient as expected for stratified Datalog$^\neg$.

That is, none of the previously known evaluation methods is goal-driven, set-at-a-time, adjustable w.r.t. control strategies, and designed specially for stratified Datalog$^\neg$ but not (general) Datalog$^\neg$. As these properties are important for efficient query evaluation on large and complex stratified Datalog$^\neg$ databases, it is desirable to develop an evaluation framework for stratified Datalog$^\neg$ with such properties.

1.3 Our Contributions

In this paper, we provide a novel framework, called QSQN-STR, for evaluating queries to stratified Datalog$^\neg$ databases. It extends the QSQN framework [11, 9] with the ability to handle stratified negation (but is formulated for stratified Datalog$^\neg$ databases instead of stratified knowledge bases in first-order logic). QSQN-STR is goal-driven, set-at-a-time and allows a variety of control strategies. In particular, every control strategy "*admissible w.r.t. strata's stability*" can be used for QSQN-STR. Roughly speaking, the admissibility w.r.t. strata's stability only requires that the computation can check whether a (ground) negative goal $\sim B$ of an intensional predicate p succeeds by searching the *answer* relation of p only after the (goal-driven) processing for the lower strata up to the stratum containing clauses defining p has been completed. QSQN-STR uses a net of nodes that correspond to *input*, *answer* and *supplement* relations like the ones used for QSQ [43, 1] but without adornments. The net is constructed from the given stratified Datalog$^\neg$ program.

It contains simple data structures that are needed for performing query evaluation. At the abstract level, the skeleton of QSQN-STR is as follows:

> while there are edges (u, v) such that u contains data to be processed for the edge (u, v), do:
>
> - select such an edge so that the selection is admissible w.r.t. strata's stability;
> - process the data at u to produce and transfer data through the edge (u, v).

As QSQN-STR allows every control strategy that is admissible w.r.t. strata's stability, it is really a framework. We also refer to it as a generic evaluation method for stratified Datalog$^\neg$. This method is sound, complete and has a PTIME data complexity.

What control strategies should be used for QSQN-STR is left for the implementation and experimentation phases. Besides, operations specified for QSQN-STR can be optimized at the implementation phase. We have implemented a prototype of QSQN-STR in Java, using a control strategy named IDFS2, which is specified in [9]. The prototype has not yet been optimized. So, in general, it cannot compete with highly optimized engines like XSB [40]. Nevertheless, we have performed experiments and made a comparison between our prototype of QSQN-STR, DES-DBMS[4] (version 5.0.1) and SWI-Prolog[5] (version 6.4) w.r.t. the execution time by using a number of tests. The experimental results show that our prototype of QSQN-STR outperforms DES-DBMS by a few orders of magnitude for all of the tests. It is competitive with SWI-Prolog for the tests for which SWI-Prolog can terminate properly.

This paper is a revised and extended/modified version of the conference paper [8] and a chapter of the first author's PhD dissertation [9]. The QSQN-STR framework presented in this paper is formulated for stratified Datalog$^\neg$ databases but not stratified knowledge bases in first-order logic. It has been improved by allowing a larger class of control strategies and adopting an essential optimization[6]. Consequently, the proof of soundness and completeness has been updated. Furthermore, the presentation has been significantly improved.

1.4 The Structure of This Paper

The rest of this paper is structured as follows. Section 2 recalls the most important concepts and definitions. In Section 3, we give a new presentation of the QSQN framework, which is thorough and more understandable than the one in [11, 9]. In Section 4, we incorporate stratified negation into query-subquery nets and extend QSQN to QSQN-STR. (To get the gist of QSQN-STR, the reader may watch the demonstration [12] in the PowerPoint-like mode first.) Conclusions are given

[4] The Datalog Education System (DES) with a DBMS via ODBC, available at `http://des.sourceforge.net` (see also, e.g., [39]).

[5] Available at `http://www.swi-prolog.org/`

[6] See the step 2a of $\texttt{fire}'(u, v)$ in Definition 23.

in Section 5. Due to the lack of space, our proofs and experimental results are presented only in the online appendix [13].

2 PRELIMINARIES

We assume that the reader is familiar with basic notions of first-order logic. In this section, we recall only the most important definitions and notions that are needed for our work, which are based on [1, 2, 14, 23, 28, 33]. We refer the reader to [1, 28] for further reading.

A *signature* for Datalog$^\neg$ consists of constants, variables and predicates. Each predicate is classified either as *intensional* or as *extensional*. Due to the absence of function symbols, a *term* is defined to be either a constant or a variable. An *atom* is an expression of the form $p(t_1, \ldots, t_n)$, where $n \geq 0$, p is an n-ary predicate and each t_i is a term. A *literal* is either an atom (called a *positive literal*) or the negation of an atom (called a *negative literal*). *Formulas* are defined in the usual way. An *expression* is a term, a tuple of terms, a formula without quantifiers or a list of formulas without quantifiers. A *simple expression* is either a term or an atom. An expression is *ground* if it does not contain variables.

2.1 Substitution and Unification

A *substitution* is a finite set $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$, where $x_1, \ldots, x_k$ are pairwise distinct variables, $t_1, \ldots, t_k$ are terms, and $t_i \neq x_i$ for all $1 \leq i \leq k$. The set $dom(\theta) = \{x_1, \ldots, x_k\}$ is called the *domain* of θ, and $range(\theta) = \{t_1, \ldots, t_k\}$ the *range* of θ. The *restriction* of a substitution θ to a set X of variables is the substitution $\theta_{|X} = \{(x/t) \in \theta \mid x \in X\}$. By ε we denote the *empty substitution*.

Given an expression E and a substitution $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$, the *instance* of E by θ, denoted by $E\theta$, is defined to be the expression obtained from E by simultaneously replacing every occurrence of x_i in E by t_i, for all $1 \leq i \leq k$.

Given substitutions $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$ and $\delta = \{y_1/s_1, \ldots, y_h/s_h\}$, the *composition* $\theta\delta$ of θ and δ is defined to be the substitution obtained from the sequence $\{x_1/(t_1\delta), \ldots, x_k/(t_k\delta), y_1/s_1, \ldots, y_h/s_h\}$ by deleting any binding $x_i/(t_i\delta)$ with $x_i = (t_i\delta)$ and deleting any binding y_j/s_j with $y_j \in \{x_1, \ldots, x_k\}$.

A substitution θ is *idempotent* if $\theta\theta = \theta$. It is known that $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$ is idempotent if and only if none of $x_1, \ldots, x_k$ occurs in any $t_1, \ldots, t_k$. If θ and δ are substitutions such that $\theta\delta = \delta\theta = \varepsilon$, then we call them *renaming substitutions*. A substitution θ is *more general* than a substitution δ if there exists a substitution γ such that $\delta = \theta\gamma$. According to this definition, θ is more general than itself.

Let Γ be a set of simple expressions. A substitution θ is called a *unifier* for Γ if $\Gamma\theta$ is a singleton. If $\Gamma\theta = \{\varphi\}$, then we say that θ unifies Γ (into φ). A unifier θ for Γ is called a *most general unifier* (mgu) for Γ if θ is more general than every unifier of Γ. There is an effective algorithm, called the *unification algorithm*, for checking

whether a set Γ of simple expressions is unifiable (i.e., has a unifier) and computing an idempotent mgu for Γ if Γ is unifiable (see, e.g., [28]).

2.2 Stratified Datalog$^\neg$

We recall here the definition of databases in Datalog and (stratified) Datalog$^\neg$.

Definition 1. A *safe Datalog$^\neg$ program clause* (w.r.t. the leftmost selection function) has the form $A \leftarrow B_1, \ldots, B_k$, with $k \geq 0$, and satisfies the following conditions:

1. A is an atom and each B_i is a literal, where negation is denoted by $\sim$,
2. every variable occurring in A also occurs in $(B_1, \ldots, B_k)$,
3. every variable occurring in a negative literal B_j also occurs in some positive literal B_i with $1 \leq i < j$.

The atom A is called the *head* and $(B_1, \ldots, B_k)$ the *body* of the program clause. When $k = 0$, the body is empty and the clause can be written without $\leftarrow$. If p is the predicate of A, then the program clause is called a *program clause defining* p. Such a program clause is treated as an expression (so we can talk about its instances).

A *safe Datalog$^\neg$ program* (w.r.t. the leftmost selection function) is a finite set of safe Datalog$^\neg$ program clauses. A safe Datalog$^\neg$ program without negative literals in the clauses' bodies is called a *safe Datalog program*. From now on, by a *Datalog$^\neg$* (resp. *Datalog*) *program* we mean a safe Datalog$^\neg$ (resp. Datalog) program. The second and third conditions in Definition 1 are called the *safety condition of Datalog$^\neg$*. The second condition itself is also called the *safety condition of Datalog*.

Given a Datalog$^\neg$ program P, a *stratification* of P is a partition $P = P_1 \cup \ldots \cup P_n$ such that, for each $1 \leq i \leq n$, we have that:[7]

- if an intensional predicate p occurs in a positive literal in the body of a clause from P_i, then the clauses defining p must belong to $P_1 \cup \ldots \cup P_i$,
- if an intensional predicate p occurs in a negative literal in the body of a clause from P_i, then $i > 1$ and the clauses defining p must belong to $P_1 \cup \ldots \cup P_{i-1}$.

Each P_i is called a *stratum* of the stratification. A Datalog$^\neg$ program is called a *stratified Datalog$^\neg$ program* if it has a stratification.

An *instance* of extensional predicates is a mapping I that associates each extensional n-ary predicate p to a finite set $I(p)$ of n-ary tuples of constants. Sometimes, I is treated as the set $\{p(\bar{t}) \mid \bar{t} \in I(p)\}$ and each $p(\bar{t}) \in I$ is treated as the program clause $p(\bar{t}) \leftarrow$. The *size* of I is defined to be the cardinality of the mentioned set.

A *Datalog$^\neg$* (resp. *Datalog*) *database* is a pair (P, I), where P is a Datalog$^\neg$ (resp. Datalog) program consisting of clauses defining intensional predicates and I is an instance of extensional predicates. A *stratified Datalog$^\neg$ database* is a Datalog$^\neg$ database (P, I) with P being a stratified Datalog$^\neg$ program.

[7] All of the sets $P_1, \ldots, P_n$ are assumed to be non-empty.

2.3 The Standard Semantics of Stratified Datalog$^\neg$

In this subsection, let (P, I) be a stratified Datalog$^\neg$ database. The *Herbrand universe* of (P, I), denoted by $U_{P,I}$, is the set of all constants occurring in (P, I). The *Herbrand base* of (P, I), denoted by $B_{P,I}$, is the set of all ground atoms of the form $p(t_1, \ldots, t_n)$, where p is a predicate used in (P, I) and each t_i belongs to $U_{P,I}$. A *Herbrand interpretation* for (P, I) is a subset of $B_{P,I}$.

If $\mathcal{I}$ is a Herbrand interpretation and $p(\bar{t})$ a ground atom, then by $\mathcal{I}(p(\bar{t}))$ we denote that $p(\bar{t}) \in \mathcal{I}$, and by $\mathcal{I}(\sim p(\bar{t}))$ we denote that $p(\bar{t}) \notin \mathcal{I}$.

Let $ground(P \cup I)$ be the set of all ground instances of clauses in $P \cup I$, and $\mathcal{I}$ a Herbrand interpretation for (P, I). The *immediate consequence operator* of (P, I), denoted by $T_{P,I}$, is defined on $\mathcal{I}$ as follows:

$$T_{P,I}(\mathcal{I}) = \{A \mid A \leftarrow B_1, \ldots, B_k \in ground(P \cup I) \text{ and } \mathcal{I}(B_i) \text{ holds for all } 1 \leq i \leq k\}.$$

Let $T_{P,I} \uparrow \omega$ be defined as follows:

$$\begin{aligned} T_{P,I} \uparrow 0 &= I, \\ T_{P,I} \uparrow (n+1) &= T_{P,I}(T_{P,I} \uparrow n) \cup T_{P,I} \uparrow n, \quad \text{for } n \in \mathbb{N}, \\ T_{P,I} \uparrow \omega &= \bigcup_{n=0}^{\omega} T_{P,I} \uparrow n. \end{aligned}$$

Let $P_1 \cup \ldots \cup P_n$ be a stratification of P. We define

$$\begin{aligned} M_{\emptyset,I} &= I, \\ M_{P_1,I} &= T_{P_1,I} \uparrow \omega, \\ M_{P_1 \cup P_2,I} &= T_{P_2,M_{P_1,I}} \uparrow \omega, \\ &\vdots \\ M_{P_1 \cup \ldots \cup P_n,I} &= T_{P_n,M_{P_1 \cup \ldots \cup P_{n-1},I}} \uparrow \omega. \end{aligned}$$

We call $M_{P,I} = M_{P_1 \cup \ldots \cup P_n,I}$ the *standard Herbrand model* of (P, I).

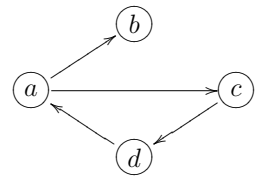
It can be shown that the standard Herbrand model of (P, I) does not depend on the chosen stratification of P (see, e.g., [2, Theorem 11]).

Example 1. Consider the stratified Datalog$^\neg$ database (P, I) given below, where P is a modified version of a Datalog$^\neg$ program given in [36], *path* and *acyclic* are intensional predicates, *edge* is an extensional predicate, x, y and z are variables and a–f are constants. An atom $edge(x, y)$ means that there is an edge from the node x to the node y. An atom $path(x, y)$ means that there exists a path (consisting of edges) that connects the node x to the node y. An atom $acyclic(x, y)$ means that the node x is connected by a path to the node y, but not vice versa.

- P consists of the following clauses:

$$\begin{aligned} path(x,y) &\leftarrow edge(x,y), \\ path(x,y) &\leftarrow path(x,z), edge(z,y), \\ acyclic(x,y) &\leftarrow path(x,y), \sim path(y,x). \end{aligned}$$

- I is specified and illustrated as follows: $I(edge) = \{(a,b),(a,c),(c,d),(d,a)\}$.



The only stratification of P is $P_1 \cup P_2$, where:

$$\begin{array}{ll} P_1: & path(x,y) \leftarrow edge(x,y), \\ & path(x,y) \leftarrow path(x,z), edge(z,y), \\ \hline P_2: & acyclic(x,y) \leftarrow path(x,y), \sim path(y,x). \end{array}$$

The standard Herbrand model $M_{P,I}$ of (P,I) is constructed as follows:

$$\begin{aligned} M_{\emptyset,I} &= I, \\ M_{P_1,I} &= M_{\emptyset,I} \cup \{path(x,y) \mid (x,y) \in \{a,c,d\} \times \{a,b,c,d\}\}, \\ M_{P_1 \cup P_2,I} &= M_{P_1,I} \cup \{acyclic(a,b), acyclic(c,b), acyclic(d,b)\}. \end{aligned}$$

Thus, $M_{P,I} = M_{P_1 \cup P_2,I}$ is the standard Herbrand model of (P,I).

We define a *query* to a stratified Datalog$^\neg$ database (P,I) to be a formula of the form $q(\overline{x})$, where q is an intensional predicate and $\overline{x}$ is a tuple of pairwise distinct variables (of the same arity as q). A *correct answer* for a query $q(\overline{x})$ to a stratified Datalog$^\neg$ database (P,I) is a tuple $\overline{t}$ of constants of the same arity as $\overline{x}$ such that $q(\overline{t}) \in M_{P,I}$. The *data complexity* of an algorithm for computing all (correct) answers for a query $q(\overline{x})$ to a stratified Datalog$^\neg$ database (P,I) is measured in the size of I.

Remark 1. Note that, if φ can be the body of a Datalog$^\neg$ program clause, then it can be treated as a query to a stratified Datalog$^\neg$ database (P,I) by adding to P a new program clause $q(\overline{x}) \leftarrow \varphi$ to obtain P', where q is a new intensional predicate and $\overline{x}$ consists of all the variables occurring in φ, and then using the query $q(\overline{x})$ to the stratified Datalog$^\neg$ database (P',I).

2.4 SLD-Resolution

SLD-resolution [2, 28] is a calculus for the Horn fragment of first-order logic, which is more general than Datalog in that function symbols are allowed and program clauses do not need to satisfy the safety condition. In this subsection, we recall a formulation of SLD-resolution for Datalog, which is useful for our introduction of QSQN in the next section.

If E is an expression or a substitution, then by $Vars(E)$ we denote the set of all variables occurring in E. We say that an expression E is a *variant* of an expression E' if there exist renaming substitutions θ and γ such that $E = E'\theta$ and $E' = E\gamma$. In a computational process, a *fresh variant* of φ, where φ can be a term, a tuple of terms, an atom or a program clause $A \leftarrow B_1, \ldots, B_k$, is $\varphi\theta$, where θ is a renaming substitution such that $dom(\theta) = Vars(\varphi)$ and $range(\theta)$ consists of variables that were not used earlier in the computation.

A *goal (without negation)* has the form $\leftarrow B_1, \ldots, B_k$, where $B_1, \ldots, B_k$ are atoms. If $k = 0$, then the goal is called the *empty goal* and denoted by $\Box$.

A goal G' is *derived* from a goal $G = \leftarrow A_1, \ldots, A_i, \ldots, A_k$ and a Datalog program clause $\varphi = (A \leftarrow B_1, \ldots, B_h)$ using an mgu θ and the *selected atom* A_i if θ is an mgu for A_i and A, and $G' = \leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_h, A_{i+1}, \ldots, A_k)\theta$. In that case, G' is called a *resolvent* of G and φ. If $i = 1$, then we say that G' is derived from G and φ using *the leftmost selection function*.

In the rest of this subsection, let P be a Datalog program and G a goal.

An *SLD-derivation* from $P \cup \{G\}$ consists of a (finite or infinite) sequence $G_0 = G$, G_1, G_2, ... of goals, a sequence $\varphi_1, \varphi_2, \ldots$ of variants of program clauses of P and a sequence $\theta_1, \theta_2, \ldots$ of mgu's such that each G_{i+1} is derived from G_i and φ_{i+1} using θ_{i+1}. Each φ_i is called an *input program clause*.

When constructing an SLD-derivation, for generality and clarity, it is assumed that each φ_i does not have any variable that already appears in the derivation up to G_{i-1}. The simplest way to guarantee this is to choose each φ_i as a fresh variant of a program clause from P.

An *SLD-refutation* of $P \cup \{G\}$ is a finite SLD-derivation from $P \cup \{G\}$ with the empty goal as the last goal in the derivation.

A *computed answer* θ for $P \cup \{G\}$ is the substitution obtained by restricting the composition $\theta_1 \ldots \theta_n$ to the variables of G, where $\theta_1, \ldots, \theta_n$ is the sequence of mgu's occurring in an SLD-refutation of $P \cup \{G\}$.

3 QUERY-SUBQUERY NETS REVISITED

The notion of query-subquery net and the related evaluation framework QSQN for evaluating queries to Horn knowledge bases were introduced by us in [11, 9]. They can be used for evaluating queries to Datalog databases by setting the term-depth limit to 0. In this section, we present a thorough and more understandable description of QSQN by using a running example and relating QSQN to SLD-resolution with tabulation.

3.1 An Example Illustrating SLD-Resolution with Tabulation

Consider the stratum P_1 from Example 1 and let I be the extensional instance specified by $I(edge) = \{(a, b), (b, c)\}$. Let $P = P_1 \cup I$, with I treated as a set of atoms of *edge*. Thus, P is the Datalog program consisting of the following clauses:

$$\begin{array}{ll} (1)\ \ path(x, y) \leftarrow edge(x, y), & (3)\ \ edge(a, b), \\ (2)\ \ path(x, y) \leftarrow path(x, z), edge(z, y), & (4)\ \ edge(b, c). \end{array}$$

Consider the goal $\leftarrow path(x, y)$. It is easy to see that there are three computed answers for $P \cup \{\leftarrow path(x, y)\}$: $\{x/a, y/b\}$, $\{x/b, y/c\}$ and $\{x/a, y/c\}$. We first demonstrate how to use SLD-resolution (with the leftmost selection function) together with a technique called *tabulation* to obtain these answers and justify that there are no more computed answers. The process is summarized in Figure 1 and explained in detail below. We use one sequence of natural numbers to name clauses, tuples in relations, and steps in the process (enumerated in the following list). When a tuple is added to a relation at a step k, it is also numbered k.

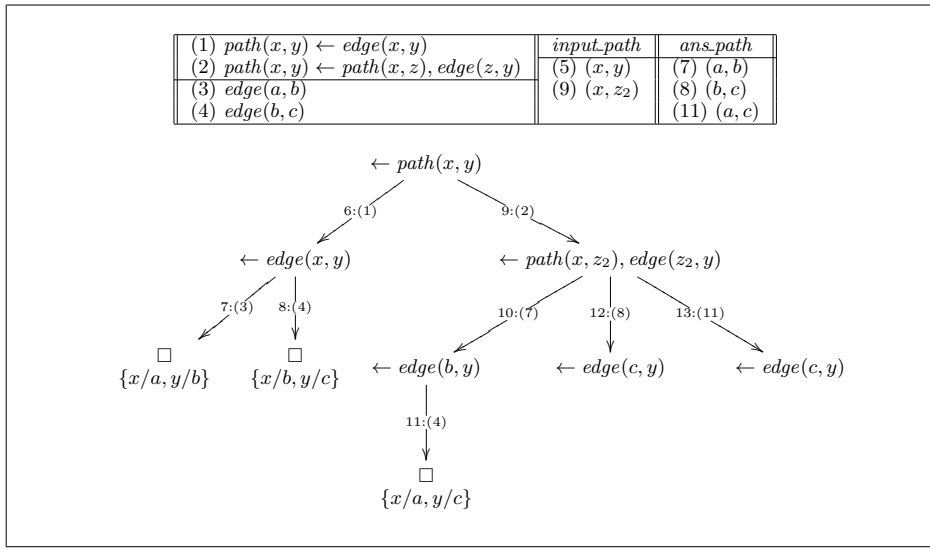


Figure 1. An illustration of SLD-resolution with tabulation. The considered Datalog program consists of the clauses (1)–(4), and the goal is $\leftarrow path(x, y)$. A label of the form $m : (n)$ of an edge from a node v to a node w in the displayed tree means that: at the step m the goal at the node v is resolved by using the clause or the *answer atom* numbered n, resulting in the goal (the resolvent) at the node w. Steps of the process are numbered from 6, as the smaller numbers are reserved for the clauses and the first *input atom*. Details are given in Section 3.1.

(5) As the clauses of P are numbered 1–4, we enumerate this list from 5. To keep the fact that we have started to deal with the goal $\leftarrow path(x, y)$, we store the tuple (x, y) in the relation named *input_path*. Tuples of this relation represent so called *input atoms* of the predicate *path*.

(6) Resolving the goal $\leftarrow path(x, y)$ by using a fresh variant $path(x_1, y_1) \leftarrow edge(x_1, y_1)$ of the clause (1) and the mgu $\{x_1/x, y_1/y\}$ results in the goal $\leftarrow edge(x, y)$.

(7) Resolving the goal $\leftarrow edge(x, y)$ by using the atom $edge(a, b)$ from I and the mgu $\{x/a, y/b\}$ results in the empty goal. We obtain the first computed answer $\{x/a, y/b\}$. As the main goal is $\leftarrow path(x, y)$, this computed answer can be represented by $path(a, b)$. To keep this answer, we store the tuple (a, b) in the relation named *ans_path*. Tuples of this relation represent so called *answer atoms* of the predicate *path*.

(8) Returning to the moment after Step 6 and resolving the goal $\leftarrow edge(x, y)$ by using the atom $edge(b, c)$ from I and the mgu $\{x/b, y/c\}$ results in the empty goal. We obtain the second computed answer $\{x/b, y/c\}$. Analogously as explained above, we add the tuple (b, c) to the relation *ans_path*.

(9) Returning to the moment after Step 5 and resolving the goal $\leftarrow path(x, y)$ by using a fresh variant $path(x_2, y_2) \leftarrow path(x_2, z_2), edge(z_2, y_2)$ of the clause (2) and the mgu $\{x_2/x, y_2/y\}$ results in the goal $\leftarrow path(x, z_2), edge(z_2, y)$. To resolve this goal using $path(x, z_2)$ as the selected atom (i.e., using the leftmost selection function), we can use:

- either a program clause defining *path* from P,
- or an answer atom of *path* that has been tabled earlier in *ans_path*.

Resolving the subgoal $\leftarrow path(x, z_2)$ by using a program clause defining *path* from P can be done in a similar way as we have been doing for the main goal $\leftarrow path(x, y)$ of the process. That is, one can continue by adding the tuple (x, z_2) to the relation *input_path* and so on. However, we first check whether this task can be ignored. Since (x, z_2) is an instance of (a fresh variant of) the existing tuple (x, y) in *input_path*, which stands for the goal $\leftarrow path(x, y)$ that has already been dealt with and is more general than the goal $\leftarrow path(x, z_2)$, storing (x, z_2) in *input_path* and processing the goal $\leftarrow path(x, z_2)$ in the usual way are unnecessary and therefore skipped.

Resolving the subgoal $\leftarrow path(x, z_2)$ by using an answer atom of *path* represented by a tuple in the relation *ans_path* is reported below. We have here a backtracking point, as there are a few of such tuples.

(10) Resolving the goal $\leftarrow path(x, z_2), edge(z_2, y)$ by selecting the atom $path(x, z_2)$ and using the tuple (a, b) in the relation *ans_path* and the mgu $\{x/a, z_2/b\}$ results in the goal $\leftarrow edge(b, y)$.

(11) Resolving the goal $\leftarrow edge(b, y)$ using the atom $edge(b, c)$ from I and the mgu $\{y/c\}$ results in the empty goal. We obtain the third computed answer $\{x/a,$

$y/c\}$ for the main goal $\leftarrow path(x, y)$, which is the restriction of the composition $\{x_2/x, y_2/y\}\{x/a, z_2/b\}\{y/c\}$ to the set $\{x, y\}$. We store this answer by adding the tuple (a, c) to the relation *ans_path*.

(12) Returning to the backtracking point mentioned at Step 9 and resolving the goal $\leftarrow path(x, z_2), edge(z_2, y)$ by selecting the atom $path(x, z_2)$ and using the tuple (b, c) in the relation *ans_path* and the mgu $\{x/b, z_2/c\}$ results in the goal $\leftarrow edge(c, y)$. This goal cannot be resolved by using any atom of *edge* from I. So, we finish this search branch (derivation).

(13) Returning to the backtracking point mentioned at Step 9 and resolving the goal $\leftarrow path(x, z_2), edge(z_2, y)$ by selecting the atom $path(x, z_2)$ and using the tuple (a, c) in the relation *ans_path* and the mgu $\{x/a, z_2/c\}$ results in the goal $\leftarrow edge(c, y)$. Once again, this goal cannot be resolved by using any atom of *edge* from I. As there are no active backtracking points, we finish the process. The computed answers are represented by the tuples in the relation *ans_path*.[8]

Consider the return to the backtracking point mentioned at Step 9 and the continuation at Step 13. It uses the tuple (a, c), which was added to the relation *ans_path* at Step 11, i.e., after the creation of the backtracking point. The considered example is simple and after Step 13 we can finish the process. But, what would happen if the used Datalog program was more complicated and the search tree (as displayed in Figure 1) had the third branch departing from the root with more computed answers? How could they be supplied for resolving the subgoal $\leftarrow path(x, z_2)$ in the second branch departing from the root? SLD-resolution systems with tabulation like OLDT use the "suspension-resumption mechanism" and the "stack-wise representation" to deal with this problem [42]. Both of these techniques are tuple-oriented (tuple-at-a-time) and not suitable for processing queries to deductive databases, as the task should be done set-at-a-time in order to deal with big data and reduce the number of accesses to the secondary storage.

3.2 Query-Subquery Nets as Data Structures for Processing Queries

Let us discuss a design of data structures for processing queries by simulating SLD-resolution with tabulation in the way so that the processing can be done set-at-a-time and every strategy for searching for answers (called a *control strategy*) can be applied. That leads to the notion of query-subquery net.

3.2.1 An Illustrative Example

Before defining query-subquery nets formally, we discuss the data structures needed for processing queries to the Datalog database (P_1, I) considered in Section 3.1.

[8] In the general case, only answer atoms that are instances of the main input atom are taken. In this concrete case, all the answer atoms $path(a, b)$, $path(b, c)$ and $path(a, c)$ are instances of the main input atom $path(x, y)$.

In general, we want to use a net for the processing, called a *query-subquery net* (or *QSQ-net* for short). It is a directed graph with nodes being appropriate data structures.

As discussed in Section 3.1, the approach uses the relations *input_path* and *ans_path*, so let the net have two nodes named ***input_path*** and ***ans_path***. Each of these nodes has an attribute named ***tuples***, which represents the corresponding relation. Thus, by writing *tuples*(*input_path*) (resp. *tuples*(*ans_path*)) we have in mind the relation *input_path* (resp. *ans_path*) mentioned in Section 3.1.

Tuples in the node *input_path* stand for goal atoms of the predicate *path*. That is, a tuple $\bar{t} \in tuples(input\_path)$ stands for the goal $\leftarrow path(\bar{t})$. Assume that, when a tuple is added to *tuples*(*input_path*), its variables have already been renamed so that they do not occur in the Datalog program P_1. In this way, when resolve a goal using a program clause of P_1, we do not have to rename variables of the program clause.

How can a tuple $\bar{t} \in tuples(input\_path)$ be processed? We can resolve the goal $\leftarrow path(\bar{t})$ by using the program clause (1) or (2).

- Consider the case when the goal $\leftarrow path(\bar{t})$ is resolved by using the program clause (1) (i.e., $path(x, y) \leftarrow edge(x, y)$). To do the task, let the node *input_path* have a connection to a node named $\boldsymbol{pre\_filter}_1$, where the subscript denotes the program clause's number. As attributes of pre_filter_1, we have $\boldsymbol{atom}(pre\_filter_1) = path(x, y)$, which is the head of the program clause, and $\boldsymbol{post\_vars}(pre\_filter_1) = \{x, y\}$, which is the set of variables occurring in the body of the program clause. Then, the task can be done by transferring the tuple $\bar{t}$ from the node *input_path* to the node pre_filter_1. At pre_filter_1, $path(\bar{t})$ is unified with $atom(pre\_filter_1)$ (i.e., $path(x, y)$) by using an mgu γ, and the pair $(\bar{t}\gamma, \gamma_{|post\_vars(pre\_filter_1)})$ is transfered to the unique successor of pre_filter_1, which is named $filter_{1,1}$. The tuple $\bar{t}\gamma$ in that pair is needed for further computation. In general, such tuples will be updated on-the-fly by taking into account subsequent mgu's in the derivation and at the end will represent answers for the goal related to the tuple taken from *input_path*. Similarly, the substitution $\gamma_{|post\_vars(pre\_filter_1)}$ is also needed for further computation. We restrict γ to $post\_vars(pre\_filter_1)$ because the other bindings in γ are redundant for further computation. We call the pair $(\bar{t}\gamma, \gamma_{|post\_vars(pre\_filter_1)})$ a *subquery*.

  After resolving the goal $\leftarrow path(\bar{t})$ using the program clause (1) (i.e., $path(x, y) \leftarrow edge(x, y)$) and the mgu γ, the resulting resolvent is $\leftarrow edge(x, y)\gamma$. So, the node $filter_{1,1}$ is related to processing this goal. In general, the subscript (i, j) of a node $\boldsymbol{filter}_{i,j}$ states that the node is related with the atom numbered j in the body of the program clause numbered i. As attributes of $filter_{1,1}$, at least we need $\boldsymbol{atom}(filter_{1,1}) = edge(x, y)$. For convenience, we also use the attributes $\boldsymbol{pred}(filter_{1,1}) = edge$ (the predicate of $atom(filter_{1,1})$) and $\boldsymbol{kind}(filter_{1,1}) = extensional$ (the kind of the predicate *edge*). For the general case of $filter_{i,j}$, we also need the attribute $\boldsymbol{pre\_vars}(filter_{i,j})$ (resp. $\boldsymbol{post\_vars}(filter_{i,j})$) for keeping variables occurring in the atoms numbered from j (resp. $j + 1$) in the body of

the program clause numbered i. For the currently considered example, we have $pre\_vars(filter_{1,1}) = \{x, y\}$ and $post\_vars(filter_{1,1}) = \emptyset$.

What should be done with the subquery $(\bar{t}\gamma, \gamma_{|post\_vars(pre\_filter_1)})$ transferred to $filter_{1,1}$ from pre_filter_1? We can either process it immediately or store it in $filter_{1,1}$ in order to accumulate more and more subqueries at the node $filter_{1,1}$ before processing them set-at-a-time. The Boolean option for this is called the *memorizing type* of $filter_{1,1}$ and denoted by $\boldsymbol{T}(filter_{1,1})$. In the case when $T(filter_{1,1}) = true$, if the subquery $(\bar{t}\gamma, \gamma_{|post\_vars(pre\_filter_1)})$ has not yet been considered for the node $filter_{1,1}$, we add it to the relations $\boldsymbol{subqueries}(filter_{1,1})$ and $\boldsymbol{unprocessed\_subqueries}(filter_{1,1})$, which are additional attributes of $filter_{1,1}$. A subquery is said to have already been considered for $filter_{1,1}$ if it is less general than a subquery from $subqueries(filter_{1,1})$ (see Definition 5). When a subquery from $unprocessed\_subqueries(filter_{1,1})$ is processed, it is deleted from this relation.

For abbreviation, let $(\bar{t}', \delta)$ denote $(\bar{t}\gamma, \gamma_{|post\_vars(pre\_filter_1)})$. How can the subquery $(\bar{t}', \delta)$ be processed at the node $filter_{1,1}$? That is, how can the earlier mentioned goal $\leftarrow edge(x, y)\gamma$ be processed at $filter_{1,1}$? We unify $atom(filter_{1,1})\delta$ (i.e., $edge(x, y)\gamma$) with each atom $edge(\bar{s}) \in I$ using an mgu γ' and transfer the tuple $\bar{t}'\gamma'$ to the unique successor of $filter_{1,1}$, which is named $post\_filter_1$. In general, each node $filter_{i,j}$ with $kind(filter_{i,j}) = extensional$ has exactly one successor, which is either $filter_{i,j+1}$ (if it exists) or $\boldsymbol{post\_filter}_i$. For the currently considered example, we do not have a node named $filter_{1,2}$ because the program clause (1) has only one atom in its body.

What should be done with the tuple $\bar{t}'\gamma'$ transferred to $post\_filter_1$ from $filter_{1,1}$? We have that $\gamma\gamma'_{|Vars(\bar{t})}$ is a computed answer for the goal $\leftarrow path(\bar{t})$. Thus, the tuple $\bar{t}'\gamma' = \bar{t}\gamma\gamma'$ specifies this computed answer. So, let the node $post\_filter_1$ have a connection to the node ans_path, then all we need to do is to transfer the tuple $\bar{t}'\gamma'$ through this connection to add it to the relation $tuples(ans\_path)$.

Summing up, to process a tuple from $input\_path$ by using the program clause (1) (i.e., $path(x, y) \leftarrow edge(x, y)$), the designed QSQ-net uses the following path of nodes with appropriate attributes:

$$input\_path \longrightarrow pre\_filter_1 \longrightarrow filter_{1,1} \longrightarrow post\_filter_1 \longrightarrow ans\_path.$$

- Consider the case when the goal $\leftarrow path(\bar{t})$ is resolved by using the program clause (2) (i.e., $path(x, y) \leftarrow path(x, z), edge(z, y)$). Analogously as for the previous case, the designed QSQ-net uses the following path of nodes:

  $$input\_path \longrightarrow pre\_filter_2 \longrightarrow filter_{2,1} \longrightarrow filter_{2,2} \longrightarrow post\_filter_2 \longrightarrow ans\_path$$

  where:

  - $atom(pre\_filter_2) = path(x, y)$,

- $atom(filter_{2,1}) = path(x, z)$ and $kind(filter_{2,1}) = intensional$,
- $atom(filter_{2,2}) = edge(z, y)$ and $kind(filter_{2,2}) = extensional$.

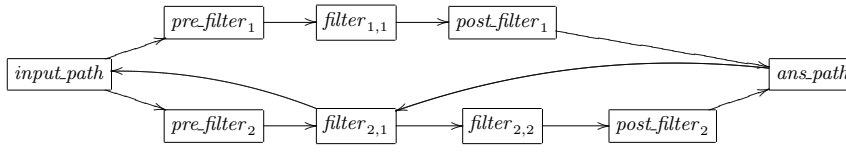
Consider a subquery $(\vec{t}', \delta)$ in the node $filter_{2,1}$. It stands for the goal $\leftarrow path(x, z)\delta, edge(z, y)\delta$. Let us pay attention to the subgoal $\leftarrow path(x, z)\delta$. As demonstrated for SLD-resolution with tabulation, to process this subgoal we should consider adding an appropriate tuple to *input_path*. In addition, to resolve the subgoal $\leftarrow path(x, z)\delta$ we can use not only the program clauses of P_1 but also the answer atoms of *path* represented by the tuples stored in *ans_path*.

To deal with the first matter, let the node $filter_{2,1}$ have a connection to the node *input_path*, then we can transfer the tuple $(x, z)\delta$ through that connection. If this tuple has not yet been considered for *input_path*, we add a fresh variant of it to $tuples(input\_path)$. A tuple is said to have already been considered for *input_path* if its fresh variant is an instance of a tuple from $tuples(input\_path)$.

Like the case of $filter_{1,1}$, the attribute $unprocessed\_subqueries(filter_{2,1})$ of $filter_{2,1}$ keeps subqueries that have not been processed at $filter_{2,1}$ to produce data to transfer to $filter_{2,2}$. The node $filter_{2,1}$ has, however, also a connection to the node *input_path*. So, we also need an attribute $\boldsymbol{unprocessed\_subqueries_2}(filter_{2,1})$ to keep subqueries that have not been processed at $filter_{2,1}$ to produce data to transfer to *input_path*.

At the node $filter_{2,1}$, to resolve the subgoal $\leftarrow path(x, z)\delta$ by using the answer atoms of *path* represented by the tuples stored in *ans_path*, we need a connection from the node *ans_path* to the node $filter_{2,1}$. New tuples added to *ans_path* are transferred through that connection and accumulated in the relation $\boldsymbol{unprocessed\_tuples}(filter_{2,1})$ before being processed (at some later steps). This relation is an additional attribute of $filter_{2,1}$.

Summing up, the QSQ-net designed for the considered Datalog program has the following topological structure:



It is illustrated in Figure 2 together with attributes of the nodes. The node *input_path* has an attribute $\boldsymbol{unprocessed}(E)$ for each outgoing edge E. A tuple $\overline{t} \in unprocessed(E)$ at the node *input_path* means that the tuple $\overline{t}$ has not been transferred from this node through the edge E (i.e., it has not been processed at the node *input_path* for the edge E). The case of the node *ans_path* is similar. Also note that we do not use the attribute $T(filter_{2,1})$ because $kind(filter_{2,1}) = intensional$. In other words, subqueries transferred to a node $filter_{i,j}$ with $kind(filter_{i,j}) = intensional$ are always memorized.

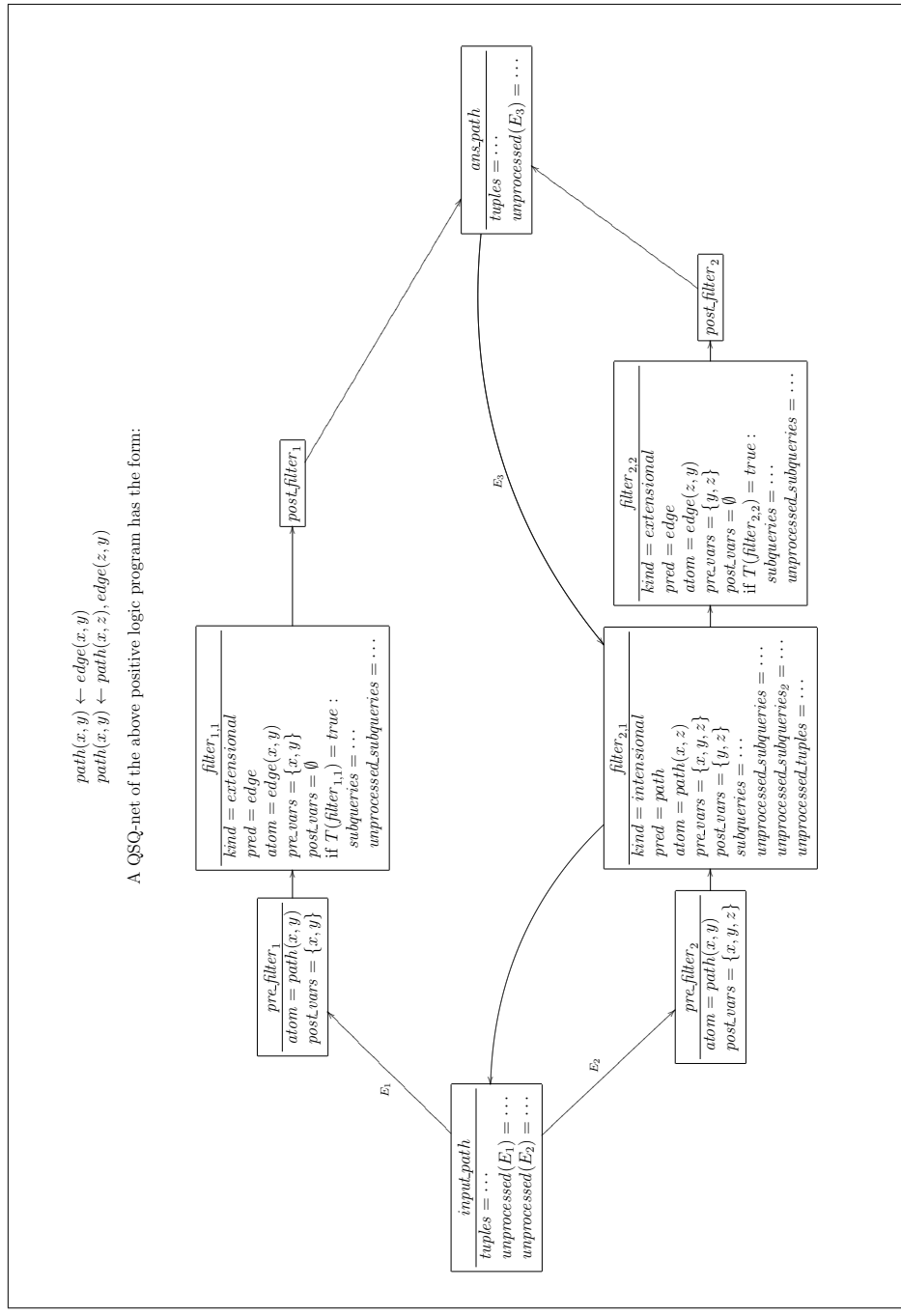


Figure 2. An example of a QSQ-net, where *path* is an intensional predicate, *edge* is an extensional predicate, and x, y, z are variables

3.2.2 A Formal Definition of Query-Subquery Nets

We now recall a formal definition of QSQ-nets [11]. From now to the end of Section 3, let P be a Datalog program and $\varphi_1, \ldots, \varphi_m$ all the program clauses of P, with $\varphi_i = (A_i \leftarrow B_{i,1}, \ldots, B_{i,n_i})$ for $1 \leq i \leq m$, where $n_i \geq 0$.

Definition 2. A *query-subquery net structure* (*QSQN structure* for short) of P is a tuple (V, E, T) such that:

- V is a set of nodes that consists of:
  - $input\_p$ and ans_p, for each intensional predicate p of P,
  - pre_filter_i, $filter_{i,1}$, ..., $filter_{i,n_i}$, $post\_filter_i$, for each $1 \leq i \leq m$.
- E is a set of edges that consists of:
  - $(filter_{i,1}, filter_{i,2})$, ..., $(filter_{i,n_i-1}, filter_{i,n_i})$, for each $1 \leq i \leq m$,
  - $(pre\_filter_i, filter_{i,1})$ and $(filter_{i,n_i}, post\_filter_i)$, for each $1 \leq i \leq m$ with $n_i \geq 1$,
  - $(pre\_filter_i, post\_filter_i)$, for each $1 \leq i \leq m$ with $n_i = 0$,
  - $(input\_p, pre\_filter_i)$ and $(post\_filter_i, ans\_p)$, for each $1 \leq i \leq m$, where p is the predicate of A_i,
  - $(filter_{i,j}, input\_p)$ and $(ans\_p, filter_{i,j})$, for each intensional predicate p and each $1 \leq i \leq m$ and $1 \leq j \leq n_i$ such that $B_{i,j}$ is a literal of p.[9]
- T is a function, called the *memorizing type* of the QSQN structure, mapping each node $filter_{i,j} \in V$ such that the predicate of $B_{i,j}$ is extensional to *true* or *false*.[10]

In a QSQN structure (V, E, T) of P, if $(v, w) \in E$ then we call w a *successor* of v, and v a *predecessor* of w. Note that V and E are uniquely specified by P. We call the pair (V, E) the *QSQN topological structure* of P.

By a *subquery* we mean a pair of the form $(\bar{t}, \delta)$, where $\bar{t}$ is a tuple of terms and δ an idempotent substitution such that $dom(\delta) \cap Vars(\bar{t}) = \emptyset$.

Definition 3. A *query-subquery net* (*QSQ-net* for short) of P is a tuple (V, E, T, C) such that (V, E, T) is a QSQN structure of P, C is a mapping that associates each node $v \in V$ with a structure called the *contents* of v, and the following conditions are satisfied:

- $C(v)$, where $v = input\_p$ or $v = ans\_p$, consists of:
  - $tuples(v)$: a set of tuples of terms with the same arity as p,

[9] We use the term "literal" here instead of "atom" or "positive literal" so that the definition can be extended for stratified Datalog$^\neg$ easily.

[10] Recall that the aim of T is that if $T(filter_{i,j}) = false$ then subqueries for $filter_{i,j}$ are always processed immediately without being accumulated at $filter_{i,j}$.

  - $unprocessed(v, w)$ for each $(v, w) \in E$: a subset of $tuples(v)$.

- $C(v)$, where $v = pre\_filter_i$, consists of:
  - $atom(v) = A_i$ and $post\_vars(v) = Vars((B_{i,1}, \ldots, B_{i,n_i}))$.
- $C(v)$, where $v = post\_filter_i$, is empty.
- $C(v)$, where $v = filter_{i,j}$ and p is the predicate of $B_{i,j}$, consists of:
  - $kind(v) = extensional$ if p is extensional, and $kind(v) = intensional$ otherwise,
  - $pred(v) = p$ and $atom(v) = B_{i,j}$,
  - $pre\_vars(v) = Vars((B_{i,j}, \ldots, B_{i,n_i}))$ and $post\_vars(v) = Vars((B_{i,j+1}, \ldots, B_{i,n_i}))$,
  - $subqueries(v)$: a set of subqueries $(\bar{t}, \delta)$ such that $\bar{t}$ has the same arity as the predicate of A_i,
  - $unprocessed\_subqueries(v) \subseteq subqueries(v)$,
  - in the case when p is intensional:
    - $unprocessed\_subqueries_2(v) \subseteq subqueries(v)$,
    - $unprocessed\_tuples(v)$: a set of tuples of terms with the same arity as p.
- if $v = filter_{i,j}$, $kind(v) = extensional$ and $T(v) = false$, then $subqueries(v) = \emptyset$ (and both $subqueries(v)$ and $unprocessed\_subqueries(v)$ can be ignored).

We use the term QSQ-net as a noun and QSQN mostly as an adjective. Both of them are abbreviations of "query-subquery net". When standing alone, QSQN refers to the related evaluation framework, which is specified and discussed in the next subsection.

An *empty QSQ-net* of P is a QSQ-net of P such that all the sets of the form $tuples(v)$, $unprocessed(v, w)$, $subqueries(v)$, $unprocessed\_subqueries(v)$, $unprocessed\_subqueries_2(v)$ or $unprocessed\_tuples(v)$ of the net are empty.

In a QSQ-net, if $v = pre\_filter_i$ or $v = post\_filter_i$ or ($v = filter_{i,j}$ and $kind(v) = extensional$), then v has exactly one successor, which we denote by $succ(v)$.

If v is $filter_{i,j}$ with $kind(v) = intensional$ and $pred(v) = p$, then v has exactly two successors. In that case, let $succ(v)$ be $filter_{i,j+1}$ if $n_i > j$, and $post\_filter_i$ otherwise, and let $succ_2(v) = input\_p$. The set $unprocessed\_subqueries(v)$ is used for the edge $(v, succ(v))$, while $unprocessed\_subqueries_2(v)$ is used for the edge $(v, succ_2(v))$.

For convenience, we denote $pre\_vars(post\_filter_i) = \emptyset$. Thus, if $v = succ(u)$, where u is pre_filter_i of $filter_{i,j}$, then $post\_vars(u) = pre\_vars(v)$.

3.3 The QSQN Framework for Evaluating Queries

Based on QSQ-nets, in [11, 9] we proposed the QSQN framework for evaluating queries to Horn knowledge bases. We recall here its version restricted to the case without function symbols for evaluating queries to Datalog databases.

The QSQN method for evaluating a query $q(\overline{x})$ to a Datalog database (P, I) can be described informally as follows:

1. create an empty QSQ-net (V, E, T, C) of P;
2. let $\overline{x}'$ be a fresh variant of $\overline{x}$;
3. add $\overline{x}'$ to $tuples(input\_q)$;
4. for each $(input\_q, v) \in E$, add $\overline{x}'$ to $unprocessed(input\_q, v)$;
5. while there are edges $(u, v) \in E$ such that there are data at u to be processed for the edge (u, v), do:
   (a) select such an edge (any strategy can be used);
   (b) process the data at u to produce data to transfer through the edge (u, v);
6. return $tuples(ans\_q)$.

A strategy for selecting an edge at the step 5a is called a *control strategy*. As every strategy can be used, the QSQN method is really a framework for evaluating queries.

Definition 4 (`active-edge`)**.** The data at a node u to be processed for an edge $(u, v) \in E$ are:

- $unprocessed(u, v)$ if u is $input\_p$ or ans_p (for some p),
- $unprocessed\_subqueries(u)$ if u is $filter_{i,j}$, $kind(u) = extensional$ and $T(u) = true$,
- $unprocessed\_subqueries(u) \cup unprocessed\_tuples(u)$ if u is $filter_{i,j}$, $kind(u) = intensional$ and $v = succ(u)$,
- $unprocessed\_subqueries_2(u)$ if u is $filter_{i,j}$, $kind(u) = intensional$ and $v = succ_2(u)$,
- the empty set otherwise.

Let `active-edge`(u, v) be the Boolean function stating that the set representing data at the node u to be processed for the edge (u, v) is not empty. If `active-edge`(u, v) holds, then we say that the edge (u, v) is *active*.

We will specify what is "processing the data at a node u to produce data to transfer through an edge (u, v)". We call that processing "firing the edge (u, v)" and let `fire`(u, v) be a procedure for doing it. Thus, the main loop of the QSQN method (i.e., the step 5) can be rewritten to: "while there exists $(u, v) \in E$ such that `active-edge`(u, v) holds, select such a pair (u, v) and `fire`(u, v)".

The types of data transferred through edges of a QSQ-net are as follows:

- data transferred through an edge of the form $(input\_p, v)$, $(v, input\_p)$, (v, ans_p) or (ans_p, v) are a set of tuples (of terms) of the same arity as p,

- data transferred through an edge (u, v) with $v = \mathit{filter}_{i,j}$ and u not being of the form $\mathit{ans\_p}$ are a set of subqueries that can be added to $\mathit{subqueries}(v)$,
- data transferred through an edge $(v, \mathit{post\_filter}_i)$ are a set of subqueries $(\bar{t}, \varepsilon)$ such that $\bar{t}$ is a tuple (of constants) of the same arity as the predicate of A_i.

Before specifying the transfer of data through edges, we first give some auxiliary definitions.

Definition 5. If $(\bar{t}, \delta)$ and $(\bar{t}', \delta')$ are subqueries that can be transferred through an edge to v, then we say that $(\bar{t}, \delta)$ is *more general* than $(\bar{t}', \delta')$ w.r.t. v, and $(\bar{t}', \delta')$ is *less general* than $(\bar{t}, \delta)$ w.r.t. v, if there exists a substitution γ such that $\bar{t}\gamma = \bar{t}'$ and $(\delta\gamma)_{|\mathit{pre\_vars}(v)} = \delta'$.

Definition 6 (`add-subquery`)**.** We define `add-subquery`$(\bar{t}, \delta, \Gamma, v)$ as a procedure that adds the subquery $(\bar{t}, \delta)$ to the set Γ but keeps in Γ only the most general subqueries w.r.t. the node v. It can be implemented as follows:

if no subquery in Γ is more general than $(\bar{t}, \delta)$ w.r.t. v then:

- delete from Γ all subqueries less general than $(\bar{t}, \delta)$ w.r.t. v;
- add $(\bar{t}, \delta)$ to Γ.

Definition 7 (`add-tuple`)**.** We define `add-tuple`$(\bar{t}, \Gamma)$ as a procedure that adds a fresh variant of the tuple $\bar{t}$ to the set Γ but keeps in Γ only the most general tuples. It can be implemented as follows:

- let $\bar{t}'$ be a fresh variant of $\bar{t}$;
- if $\bar{t}'$ is not an instance of any tuple from Γ then:
  - delete from Γ all tuples that are instances of $\bar{t}'$;
  - add $\bar{t}'$ to Γ.

Definition 8 (`transfer`)**.** We define `transfer`(D, u, v) as the following procedure for transferring the data D through the edge (u, v), using some subroutines specified later:

1. if $D = \emptyset$ then return;
2. if v is $\mathit{post\_filter}_i$ then `transfer`$(\{\bar{t} \mid (\bar{t}, \varepsilon) \in D\}, v, \mathit{succ}(v))$;
3. else if u is $\mathit{ans\_p}$ then $\mathit{unprocessed\_tuples}(v) := \mathit{unprocessed\_tuples}(v) \cup D$;
4. else if v is $\mathit{ans\_p}$ then `transfer`$_1(D, u, v)$;
5. else if v is $\mathit{input\_p}$ then `transfer`$_2(D, u, v)$;
6. else if u is $\mathit{input\_p}$ then `transfer`$_3(D, u, v)$;
7. else if v is $\mathit{filter}_{i,j}$, $\mathit{kind}(v) = \mathit{extensional}$ and $T(v) = \mathit{false}$ then `transfer`$_4(D, u, v)$;
8. else `transfer`$_5(D, u, v)$.

Observe that the case specified by the last "else" in the above definition (i.e., the step 8) is characterized by the conjunction that v is $filter_{i,j}$, ($kind(v) = extensional$ and $T(v) = true$ or $kind(v) = intensional$) and u is not of the form ans_p. The procedure $\texttt{transfer}_5(D, u, v)$ is defined only for this case, with D being a set of subqueries. Roughly speaking, it just accumulates the subqueries from D at v but ignores the ones that have already been considered for v and keeps only the most general ones.

Definition 9 ($\texttt{transfer}_5$)**.** The procedure $\texttt{transfer}_5(D, u, v)$ for the aforementioned case is (can be) implemented as follows:

for each $(\bar{t}, \delta) \in D$ do:

if no subquery in $subqueries(v)$ is more general than $(\bar{t}, \delta)$ w.r.t. v then

- delete from $subqueries(v)$ and $unprocessed\_subqueries(v)$ all subqueries less general than $(\bar{t}, \delta)$ w.r.t. v;
- add $(\bar{t}, \delta)$ to both $subqueries(v)$ and $unprocessed\_subqueries(v)$;
- if $kind(v) = intensional$ then:
  - delete from $unprocessed\_subqueries_2(v)$ all subqueries less general than $(\bar{t}, \delta)$ w.r.t. v;
  - add $(\bar{t}, \delta)$ to $unprocessed\_subqueries_2(v)$.

Definition 10 ($\texttt{transfer}_1$)**.** The procedure $\texttt{transfer}_1(D, u, v)$ for the case when $v = ans\_p$ (and $u = post\_filter_i$ for some i and D is a set of tuples of constants) is (can be) implemented as follows:

for each $\bar{t} \in D - tuples(v)$ do:

- add $\bar{t}$ to $tuples(v)$;
- for each $(v, w) \in E$, add $\bar{t}$ to $unprocessed(v, w)$.

The procedure $\texttt{transfer}_2(D, u, v)$ is defined only for the case when $v = input\_p$, with $u = filter_{i,j}$ (for some i and j) and D being a set of tuples of terms. Roughly speaking, it just accumulates fresh variants of the tuples from D at v but ignores the ones that have already been considered for v and keeps only the most general ones. Recall that we apply variable renaming for input atoms but not program clauses.

Definition 11 ($\texttt{transfer}_2$)**.** The procedure $\texttt{transfer}_2(D, u, v)$ for the case $v = input\_p$ is (can be) implemented as follows:

for each $\bar{t} \in D$ do:

- let $\bar{t}'$ be a fresh variant of $\bar{t}$;
- if $\bar{t}'$ is not an instance of any tuple from $tuples(v)$ then
  - delete from $tuples(v)$ all tuples that are instances of $\bar{t}'$;
  - add $\bar{t}'$ to $tuples(v)$;

- for each $(v, w) \in E$ do
  - delete from $unprocessed(v, w)$ all tuples that are instances of $\bar{t}'$;
  - add $\bar{t}'$ to $unprocessed(v, w)$.

The procedure $\texttt{transfer}_3(D, u, v)$ is defined only for the case when u is $input\_p$, with $v = pre\_filter_i$ for some i. It does not accumulate D at v but processes it immediately to create data, which are then transferred to $succ(v)$. The reader can recall the example on page 34 of using pre_filter_1 to process a tuple $\bar{t} \in tuples(input\_path)$. In general, the node pre_filter_i is used for resolving an input atom (represented by a tuple from D) by unifying it with the head of the program clause numbered i.

Definition 12 ($\texttt{transfer}_3$)**.** The procedure $\texttt{transfer}_3(D, u, v)$ for the case u is $input\_p$ is (can be) implemented as follows:

- $\Gamma := \emptyset$;
- for each $\bar{t} \in D$ do

  if $p(\bar{t})$ and $atom(v)$ are unifiable by an mgu γ then
  $\texttt{add-subquery}(\bar{t}\gamma, \gamma_{|post\_vars(v)}, \Gamma, succ(v))$;

- $\texttt{transfer}(\Gamma, v, succ(v))$.

The procedure $\texttt{transfer}_4(D, u, v)$ is defined only for the case when u is not of the form ans_p, v is $filter_{i,j}$, $kind(v) = extensional$ and $T(v) = false$. According to the intention of the memorizing type T, it does not accumulate the subqueries from D at v but processes them immediately to create data, which are then transferred to $succ(v)$. The reader can recall the example on page 35 of processing a subquery $(\bar{t}', \delta)$ at the node $filter_{1,1}$.

Definition 13 ($\texttt{transfer}_4$)**.** The procedure $\texttt{transfer}_4(D, u, v)$ for the aforementioned case is (can be) implemented as follows:

- let $p = pred(v)$ and set $\Gamma := \emptyset$;
- for each $(\bar{t}, \delta) \in D$ and each $\bar{t}' \in I(p)$ do

  if $atom(v)\delta$ and $p(\bar{t}')$ are unifiable by an mgu γ then
  $\texttt{add-subquery}(\bar{t}\gamma, (\delta\gamma)_{|post\_vars(v)}, \Gamma, succ(v))$;

- $\texttt{transfer}(\Gamma, v, succ(v))$.

We have fully specified the procedure $\texttt{transfer}(D, u, v)$ for transferring the data D through the edge (u, v). We now specify the procedure $\texttt{fire}(u, v)$ for "firing the edge (u, v)", i.e., for processing the data at u to produce data to transfer through the edge (u, v). Recall that this procedure is called only for active edges (u, v).

Definition 14 ($\texttt{fire}$)**.** The procedure $\texttt{fire}(u, v)$ is implemented as follows, using some subroutines specified later:

- if u is *input_p* or *ans_p* then
  - `transfer`$(\mathit{unprocessed}(u,v), u, v)$;
  - $\mathit{unprocessed}(u,v) := \emptyset$;
- else if v is *input_p* then `fire`$_1(u,v)$;
- else if u is $\mathit{filter}_{i,j}$ and $\mathit{kind}(u) = \mathit{extensional}$ then `fire`$_2(u,v)$;
- else if u is $\mathit{filter}_{i,j}$ and $\mathit{kind}(u) = \mathit{intensional}$ then `fire`$_3(u,v)$.

The procedure `fire`$_1(u,v)$ is defined only for the case when v is *input_p*, with u being $\mathit{filter}_{i,j}$ (for some i and j) and $\mathit{kind}(u) = \mathit{intensional}$. For each subquery $(\overline{t}, \delta)$ at u that has not yet been processed for the edge (u,v), the procedure transfers a fresh variant of $\overline{t}'$, where $p(\overline{t}') = \mathit{atom}(u)\delta$, through the edge (u,v) in order to add it to $\mathit{tuples}(\mathit{input\_p})$ if it has not been considered for v. The reader can recall the example on page 36 of dealing with the edge $(\mathit{filter}_{2,1}, \mathit{input\_path})$.

Definition 15 (`fire`$_1$)**.** The procedure `fire`$_1(u,v)$ for the case v is *input_p* is (can be) implemented as follows:

- let $p = \mathit{pred}(u)$ and set $\Gamma := \emptyset$;
- for each $(\overline{t}, \delta) \in \mathit{unprocessed\_subqueries}_2(u)$ do
  - let $p(\overline{t}') = \mathit{atom}(u)\delta$;
  - `add-tuple`$(\overline{t}', \Gamma)$;
- $\mathit{unprocessed\_subqueries}_2(u) := \emptyset$;
- `transfer`(Γ, u, v).

The procedure `fire`$_2(u,v)$ is defined only for the case when u is $\mathit{filter}_{i,j}$ and $\mathit{kind}(u) = \mathit{extensional}$. In this case, as the edge (u,v) is active, we must have that $T(u) = \mathit{true}$. Before specifying this procedure, recall the procedure `transfer`$_4(D, x, u')$ defined in Definition 13 for the case when u' is $\mathit{filter}_{i,j} = \mathit{succ}(x)$, $\mathit{kind}(u') = \mathit{extensional}$ and $T(u') = \mathit{false}$. This latter procedure processes the data D immediately at u' to create data, which are then transferred to $\mathit{succ}(u')$. The procedure `fire`$_2(u,v)$ processes $\mathit{unprocessed\_subqueries}(u,v)$ at u in a similar way and then empties $\mathit{unprocessed\_subqueries}(u,v)$. The reader can also recall the example on page 35 of processing a subquery at the node $\mathit{filter}_{1,1}$.

Definition 16 (`fire`$_2$)**.** The procedure `fire`$_2(u,v)$ for the aforementioned case is (can be) implemented as follows:

- let $p = \mathit{pred}(u)$ and set $\Gamma := \emptyset$;

- for each $(\bar{t}, \delta) \in \mathit{unprocessed\_subqueries}(u)$ and each $\bar{t}' \in I(p)$ do

  if $\mathit{atom}(u)\delta$ and $p(\bar{t}')$ are unifiable by an mgu γ then
  $\texttt{add-subquery}(\bar{t}\gamma, (\delta\gamma)_{|\mathit{post\_vars}(u)}, \Gamma, v)$;

- $\mathit{unprocessed\_subqueries}(u) := \emptyset$;
- $\texttt{transfer}(\Gamma, u, v)$.

The procedure $\texttt{fire}_3(u, v)$ is defined only for the case when u is $\mathit{filter}_{i,j}$, $\mathit{kind} = \mathit{intensional}$ and $v = \mathit{succ}(u)$. Let $p = \mathit{pred}(u)$. For each subquery $(\bar{t}, \delta)$ at u and each tuple $\bar{t}' \in \mathit{tuples}(\mathit{ans\_}p)$, if they (as a pair) have not been processed at u, then the procedure processes them in a similar way as the procedure $\texttt{fire}_2(u, v)$ does for a subquery $(\bar{t}, \delta)$ at u and a tuple $\bar{t}'$ from the extensional relation of the predicate of $\mathit{atom}(u)$. Note, however, that for $\texttt{fire}_3(u, v)$ we have two subcases: either the subquery $(\bar{t}, \delta)$ has not been processed at u for the tuple $\bar{t}'$, or the tuple $\bar{t}'$ has not been processed at u for the subquery $(\bar{t}, \delta)$.

Definition 17 ($\texttt{fire}_3$)**.** The procedure $\texttt{fire}_3(u, v)$ for the aforementioned case is (can be) implemented as follows:

- let $p = \mathit{pred}(u)$ and set $\Gamma := \emptyset$;
- for each $(\bar{t}, \delta) \in \mathit{unprocessed\_subqueries}(u)$ and each $\bar{t}' \in \mathit{tuples}(\mathit{ans\_}p)$ do

  if $\mathit{atom}(u)\delta$ and $p(\bar{t}')$ are unifiable by an mgu γ then
  $\texttt{add-subquery}(\bar{t}\gamma, (\delta\gamma)_{|\mathit{post\_vars}(u)}, \Gamma, v)$;

- $\mathit{unprocessed\_subqueries}(u) := \emptyset$;
- for each $(\bar{t}, \delta) \in \mathit{subqueries}(u)$ and each $\bar{t}' \in \mathit{unprocessed\_tuples}(u)$ do

  if $\mathit{atom}(u)\delta$ and $p(\bar{t}')$ are unifiable by an mgu γ then
  $\texttt{add-subquery}(\bar{t}\gamma, (\delta\gamma)_{|\mathit{post\_vars}(u)}, \Gamma, v)$;

- $\mathit{unprocessed\_tuples}(u) := \emptyset$;
- $\texttt{transfer}(\Gamma, u, v)$.

We have fully specified the QSQN method for evaluating queries to Datalog databases. An example illustrating the QSQN method can be found in [9]. Note that, in this method, processing subqueries has been designed so that:

- every subquery that is subsumed by another one is ignored,
- for *input_* relations, every tuple that is subsumed by another one is ignored,
- the processing is divided into smaller steps which can be delayed at each node to maximize adjustability and allow various control strategies,
- the processing is done set-at-a-time (e.g., for all the unprocessed subqueries or tuples accumulated in a given node).

In [9, 32] we have proved that the QSQN method for evaluating queries to Datalog databases is sound, complete and has a PTIME data complexity.

4 QUERY-SUBQUERY NETS WITH STRATIFIED NEGATION

In this section, we present our framework called QSQN-STR for evaluating queries to stratified Datalog$^\neg$ databases. It is based on query-subquery nets that are extended to deal with stratified negation.

In what follows, let P be a stratified Datalog$^\neg$ program and $\varphi_1, \ldots, \varphi_m$ all the clauses of P, with $\varphi_i = (A_i \leftarrow B_{i,1}, \ldots, B_{i,n_i})$ for $1 \leq i \leq m$, where $n_i \geq 0$.

Definition 18. A *query-subquery net structure with stratified negation* (*QSQN-STR structure* for short) of P, is a tuple (V, E, T) defined as in the case of QSQN structure (see Definition 2) with the following modification:

> for each intensional predicate p and each $1 \leq i \leq m$ and $1 \leq j \leq n_i$ such that $B_{i,j}$ is a literal of p, the pair $(ans\_p, filter_{i,j})$ is an edge (i.e., belongs to E) iff $B_{i,j}$ is a positive literal.

The pair (V, E) is called the *QSQN-STR topological structure* of P.

Figure 3 illustrates the QSQN-STR topological structure of the stratified Datalog$^\neg$ program given in Example 1.

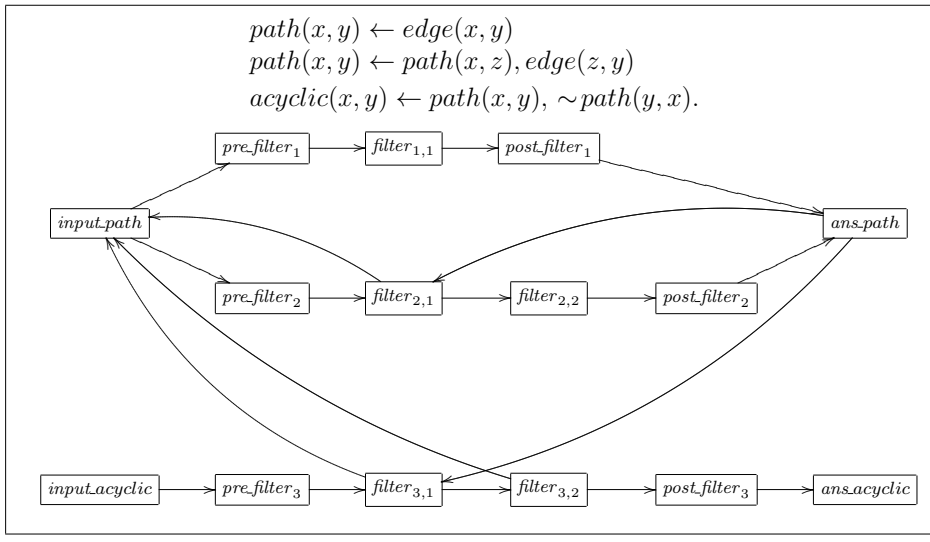


Figure 3. The QSQN-STR topological structure of the program given in Example 1

Definition 19. A *query-subquery net with stratified negation* (*QSQ-STR-net* for short)[11] of P, is a tuple (V, E, T, C) such that (V, E, T) is a QSQN-STR structure of P, and C is a mapping that associates each node $v \in V$ with a structure called

[11] We use the term QSQ-STR-net as a noun, and QSQN-STR mostly as an adjective.

the *contents* of v, which differs from the one for QSQ-net (see Definition 3) in the following:

If $v = \mathit{filter}_{i,j}$ and p is the predicate of $B_{i,j}$, then:

- $C(v)$ also contains $neg(v)$, where $neg(v) = \mathit{true}$ if $B_{i,j}$ is a negative literal, and $neg(v) = \mathit{false}$ otherwise,
- $atom(v)$ is redefined as follows: $atom(v) = B_{i,j}$ if $B_{i,j}$ is a positive literal, and $atom(v) = B'$ if $B_{i,j} = \sim B'$,
- in the case when p is intensional and $neg(v) = \mathit{true}$: $\mathit{unprocessed\_tuples}(v)$ is empty and can thus be ignored.

The notion of being *empty* is defined for QSQ-STR-net similarly as for QSQ-net.

The addition of the attribute $neg(v)$ and the modification of the attribute $atom(v)$ in the above definition are natural and do not require explanation. The below remark justifies the third difference stated in the above definition and the one in Definition 18.

From now on, by a *goal* we mean an expression of the form $\leftarrow B_1, \ldots, B_k$, where each B_i is a (positive or negative) literal.

Remark 2. Consider the QSQ-STR-net of the Datalog program P given in Example 1, whose structure is illustrated in Figure 3. We have $atom(\mathit{filter}_{3,2}) = path(y, x)$ and $neg(\mathit{filter}_{3,2}) = \mathit{true}$. Since P is safe, any $(\bar{t}, \delta) \in \mathit{subqueries}(\mathit{filter}_{3,2})$ has the properties that $\bar{t}$ is ground and δ has the form $\{x/c_1, y/c_2\}$ for some constants c_1 and c_2. The subquery $(\bar{t}, \delta)$ corresponds to the goal $\leftarrow \sim atom(\mathit{filter}_{3,2})\delta$, which is $\leftarrow \sim path(c_2, c_1)$. To resolve it, using the "negation as failure" approach, we deal with the goal $\leftarrow path(c_2, c_1)$. As this goal is ground, the answer can be either ε or failure. As usual, we transfer the tuple (c_2, c_1) through the edge $(\mathit{filter}_{3,2}, \mathit{input\_path})$ and add it to $\mathit{tuples}(\mathit{input\_path})$ if it is not an instance of another one in $\mathit{tuples}(\mathit{input\_path})$, and then proceed to check whether the tuple will be added to $\mathit{tuples}(\mathit{ans\_path})$. Despite that check, we do not need to transfer any tuple from $\mathit{ans\_path}$ through the edge $(\mathit{ans\_path}, \mathit{filter}_{3,2})$ to add to $\mathit{unprocessed\_tuples}(\mathit{filter}_{3,2})$. That is why we do not need the edge $(\mathit{ans\_path}, \mathit{filter}_{3,2})$ and the set $\mathit{unprocessed\_tuples}(\mathit{filter}_{3,2})$ will always be empty.

Based on QSQ-STR-nets we now specify our QSQN-STR method for evaluating queries to stratified Datalog$^\neg$ databases. We will use some subroutines defined earlier for the QSQN method, including `active-edge`(u, v), `add-subquery`$(\bar{t}, \delta, \Gamma, v)$ and `add-tuple`$(\bar{t}, \Gamma)$.

For transferring data D through an edge (u, v) the QSQN-STR method uses a procedure named `transfer`$'(D, u, v)$, which differs from `transfer`(D, u, v) only in processing the case when v is $\mathit{filter}_{i,j}$, $\mathit{kind}(v) = \mathit{extensional}$, $T(v) = \mathit{false}$ and $neg(v) = \mathit{true}$. In this case, a subquery $(\bar{t}, \delta) \in D$ corresponds to the goal $\leftarrow \sim atom(v)\delta, B_{i,j+1}\delta, \ldots, B_{i,n_i}\delta$. If $atom(v)\delta$ does not belong to the instance of the extensional predicate $p = \mathit{pred}(v)$, then the subquery is transferred further

to $succ(v)$ after restricting δ to $post\_vars(v)$. The task is done by the procedure $\texttt{transfer}'_{4b}(D, u, v)$ specified below.

Definition 20 ($\texttt{transfer}'_{4b}$)**.** The procedure $\texttt{transfer}'_{4b}(D, u, v)$ for the aforementioned case is (can be) implemented as follows:

- let $p = pred(v)$ and set $\Gamma := \emptyset$;
- for each $(\bar{t}, \delta) \in D$ do

  if $atom(v)\delta \notin \{p(\bar{t}') \mid \bar{t}' \in I(p)\}$ then
  $\texttt{add-subquery}(\bar{t}, \delta_{|post\_vars(u)}, \Gamma, succ(v))$;

- $\texttt{transfer}'(\Gamma, v, succ(v))$.

Recall that the procedures $\texttt{transfer}_1(D, u, v)$, $\texttt{transfer}_2(D, u, v)$ and $\texttt{transfer}_5(D, u, v)$ do not call the procedure $\texttt{transfer}$, but $\texttt{transfer}_3(D, u, v)$ and $\texttt{transfer}_4(D, u, v)$ do.

Definition 21 ($\texttt{transfer}'$, cf. Definition 8)**.** Let $\texttt{transfer}'_3(D, u, v)$ and $\texttt{transfer}'_4(D, u, v)$ be the procedures obtained from $\texttt{transfer}_3(D, u, v)$ and $\texttt{transfer}_4(D, u, v)$, respectively, by replacing the call $\texttt{transfer}(\Gamma, v, succ(v))$ with $\texttt{transfer}'(\Gamma, v, succ(v))$. Then, the procedure $\texttt{transfer}'(D, u, v)$ is (can be) implemented as follows:

1. if $D = \emptyset$ then return;
2. if v is $post\_filter_i$ then $\texttt{transfer}'(\{\bar{t} \mid (\bar{t}, \varepsilon) \in D\}, v, succ(v))$;
3. else if u is ans_p then $unprocessed\_tuples(v) := unprocessed\_tuples(v) \cup D$;
4. else if v is ans_p then $\texttt{transfer}_1(D, u, v)$;
5. else if v is $input\_p$ then $\texttt{transfer}_2(D, u, v)$;
6. else if u is $input\_p$ then $\texttt{transfer}'_3(D, u, v)$;
7. else if v is $filter_{i,j}$, $kind(v) = extensional$ and $T(v) = false$ then
   (a) if $neg(v) = false$ then $\texttt{transfer}'_4(D, u, v)$;
   (b) else $\texttt{transfer}'_{4b}(D, u, v)$;
8. else $\texttt{transfer}_5(D, u, v)$.

Assume that an edge (u, v) is active (i.e., $\texttt{active-edge}(u, v)$ holds) and is selected by an *admissible* control strategy, which will be specified later. Then, the QSQN-STR method uses a procedure named $\texttt{fire}'(u, v)$ instead of $\texttt{fire}(u, v)$ for processing the data at u to produce data to transfer through the edge (u, v). Without optimizations it would differ from $\texttt{fire}(u, v)$ only in processing the case when u is $filter_{i,j}$, $neg(u) = true$ and $v = succ(u)$. In this case, a subquery $(\bar{t}, \delta) \in unprocessed\_subqueries(u)$ corresponds to the goal $\leftarrow \sim atom(u)\delta, B_{i,j+1}\delta, \ldots, B_{i,n_i}\delta$.

Let $p = pred(u)$ and let R be $I(p)$ if $kind(u) = extensional$, and $tuples(ans\_p)$ otherwise. If $atom(u)\delta$ is different from $p(\overline{t}')$ for every $\overline{t}' \in R$, then the subquery is transferred to v after restricting δ to $post\_vars(u)$. The task is done by the procedure $\texttt{fire}'_4(u, v)$ specified below.

Definition 22 ($\texttt{fire}'_4$)**.** The procedure $\texttt{fire}'_4(u, v)$ for the aforementioned case is implemented as follows:

- let $p = pred(u)$ and set $\Gamma := \emptyset$;
- let R be $I(p)$ if $kind(u) = extensional$, and $tuples(ans\_p)$ otherwise;
- for each $(\overline{t}, \delta) \in unprocessed\_subqueries(u)$ do

  if $atom(u)\delta \notin \{p(\overline{t}') \mid \overline{t}' \in R\}$ then
  $\texttt{add-subquery}(\overline{t}, \delta_{|post\_vars(u)}, \Gamma, v)$;
- $unprocessed\_subqueries(u) := \emptyset$;
- $\texttt{transfer}'(\Gamma, u, v)$.

Observe that, when $u = filter_{i,j}$, $neg(u) = true$, $v = succ(u)$ and $\texttt{active-edge}(u, v)$ holds, if $kind(u) = extensional$, then $T(u) = true$. The procedure $\texttt{fire}'_4(u, v)$ for the subcase when $kind(u) = extensional$ (and $T(u) = true$) processes the data at u to transfer through the edge (u, v) in a similar way as the procedure $\texttt{transfer}'_{4b}(D, x, u')$ (see Definition 20) processes the data D immediately at u' to create data, which are then transferred to $succ(u')$. Here, u' plays a similar role as u, but $T(u') = false$, while $T(u) = true$.

Consider the procedure $\texttt{fire}'_4(u, v)$ for the subcase when $kind(u) = intensional$. For a subquery $(\overline{t}, \delta) \in unprocessed\_subqueries(u)$, the check whether $atom(u)\delta$ does not belong to $\{p(\overline{t}') \mid \overline{t} \in tuples(ans\_p)\}$ for $p = pred(u)$ should be done only at a suitable moment, i.e., when necessary work has been done to guarantee that an answer for the goal $\leftarrow atom(u)\delta$, if it exists, has been stored in $tuples(ans\_p)$ as the tuple $\overline{s}$ with $p(\overline{s}) = atom(u)\delta$. This means that, if $u = filter_{i,j}$, $neg(u) = true$, $kind(u) = intensional$ and $v = succ(u)$, then the edge (u, v) can be selected for "firing" (by $\texttt{fire}'(u, v)$, which calls $\texttt{fire}'_4(u, v)$) only when it is active and, additionally, satisfies appropriate conditions. In other words, the used control strategy (for selecting an edge to fire) should satisfy appropriate conditions. We will introduce a class of such control strategies shortly, which consists of so called *control strategies admissible w.r.t. strata's stability*.

The following definition formally specifies the procedure $\texttt{fire}'(u, v)$. Recall that this procedure is called only for active edges (u, v).

Definition 23 ($\texttt{fire}'$, cf. Definition 14)**.** Let $\texttt{fire}'_1(u, v)$, $\texttt{fire}'_2(u, v)$ and $\texttt{fire}'_3(u, v)$ be the procedures obtained from $\texttt{fire}_1(u, v)$, $\texttt{fire}_2(u, v)$ and $\texttt{fire}_3(u, v)$, respectively, by replacing the call $\texttt{transfer}(\Gamma, u, v)$ with $\texttt{transfer}'(\Gamma, u, v)$. Then, the procedure $\texttt{fire}'(u, v)$ is (can be) implemented as follows:

1. if u is *ans_p* then
   (a) $\texttt{transfer}'(\mathit{unprocessed}(u,v), u, v)$;
   (b) $\mathit{unprocessed}(u,v) := \emptyset$;
2. else if u is *input_p* then
   (a) $\texttt{transfer}'(\mathit{unprocessed}(u,v) - \mathit{tuples}(\mathit{ans\_p}), u, v)$;
   (b) $\mathit{unprocessed}(u,v) := \emptyset$;
3. else if v is *input_p* then $\texttt{fire}'_1(u,v)$;
4. else if u is $\mathit{filter}_{i,j}$, $\mathit{neg}(u) = \mathit{false}$ and $\mathit{kind}(u) = \mathit{extensional}$ then $\texttt{fire}'_2(u,v)$;
5. else if u is $\mathit{filter}_{i,j}$, $\mathit{neg}(u) = \mathit{false}$ and $\mathit{kind}(u) = \mathit{intensional}$ then $\texttt{fire}'_3(u,v)$;
6. else if u is $\mathit{filter}_{i,j}$ and $\mathit{neg}(u) = \mathit{true}$ then $\texttt{fire}'_4(u,v)$.

The exclusion of tuples belonging to $\mathit{tuples}(\mathit{ans\_p})$ from the transfer at the step 2a of $\texttt{fire}'(u,v)$ is an optimization. Note that every processed goal of the form $\leftarrow \sim p(\bar{t})$ is ground, and before processing the goal $\leftarrow p(\bar{t})$ (for "negation as failure") by using a program clause of P, one can check whether $\bar{t} \in \mathit{tuples}(\mathit{ans\_p})$. If so, then we already have the answer ε and can ignore the goal $\leftarrow p(\bar{t})$. One can also optimize the procedure $\texttt{transfer}_2(D, u, \mathit{input\_p})$ by excluding tuples belonging to $\mathit{tuples}(\mathit{ans\_p})$.

We now define control strategies admissible w.r.t. strata's stability.

From now on, let P be a stratified Datalog$^\neg$ program and $P_1 \cup \ldots \cup P_K$ a stratification of P. The notions defined below are dependent on this fixed stratification.

Given a QSQ-STR-net (V, E, T, C) of P, we say that a node $v \in V$ *belongs to the layer* k, where $1 \leq k \leq K$, if v is constructed by some program clauses in P_k.[12] In that case, we say that the layer number of v is k, denoted by $\mathit{layer}(v) = k$.

A QSQ-STR-net of P is said to be *stable* up to a layer k if every edge (u, v) such that the layer numbers of u and v are less than or equal to k is not active.

Definition 24 (Admissibility w.r.t. Strata's Stability)**.** A control strategy for a given QSQ-STR-net of P (i.e., a strategy for selecting an edge to apply the procedure $\texttt{fire}'$ to it) is said to be *admissible w.r.t. strata's stability* if at the moment when an edge $(v, \mathit{succ}(v))$ with $v = \mathit{filter}_{i,j}$ is selected, if $\mathit{neg}(v) = \mathit{true}$, $\mathit{layer}(v) = k$, $\mathit{pred}(v) = p$ and p is an intensional predicate with $\mathit{layer}(\mathit{input\_p}) = h$, then the QSQ-STR-net is stable up to the layer h and the edge $(v, \mathit{input\_p})$ is not active.

By restricting to the case $\mathit{neg}(v) = \mathit{true}$, the condition of admissibility w.r.t. strata's stability in the above definition is weaker than the ones in [8, 9]. That is, we have extended the class of control strategies admissible w.r.t. strata's stability in comparison with the ones in [8, 9].

Finally, our QSQN-STR method for evaluating queries to stratified Datalog$^\neg$ databases is formulated by Algorithm 1. To ease the checking we have gathered

[12] That is, P_k contains a clause φ_i such that v is of the form *input_p*, $\mathit{pre\_filter}_i$, $\mathit{filter}_{i,j}$, $\mathit{post\_filter}_i$, or *ans_p*, where p is the predicate of A_i.

Algorithm 1: evaluating a query $q(\overline{x})$ to a stratified Datalog$^\neg$ database (P, I).

Input: a stratified Datalog$^\neg$ database (P, I), a stratification $P = P_1 \cup \ldots \cup P_K$ of P and a query $q(\overline{x})$.

Output: the set of all correct answers for the query $q(\overline{x})$ on (P, I).

let (V, E, T) be a QSQN-STR structure of P;
// T can be chosen arbitrarily or appropriately
set C so that (V, E, T, C) is an empty QSQ-STR-net of P;
let $\overline{x}'$ be a fresh variant of $\overline{x}$;
tuples(*input_q*) := $\{\overline{x}'\}$;
foreach $(input\_q, v) \in E$ **do** *unprocessed*$(input\_q, v) := \{\overline{x}'\}$;
while *there exists* $(u, v) \in E$ *such that* active-edge(u, v) *holds* **do**
    select any edge $(u, v) \in E$ such that active-edge(u, v) holds and the selection satisfies the admissibility w.r.t. strata's stability;
    fire$'(u, v)$;
return *tuples*(*ans_q*);

its full pseudocode into the online appendix [13]. An example illustrating how Algorithm 1 works step by step is also provided in [13]. A more friendly presentation of that example in the PowerPoint-like mode is also available online [12].

Observe that, if P is a Datalog program, then a run of Algorithm 1 coincides with an application of the QSQN evaluation method. That is, QSQN-STR coincides with QSQN when restricted to Datalog.

Theorem 1. The QSQN-STR method formulated by Algorithm 1 for evaluating queries to stratified Datalog$^\neg$ databases is sound, complete and has a PTIME data complexity.

The proof of this theorem is provided in the online appendix [13].

5 CONCLUSIONS

The previously known methods that can be used for evaluating queries to stratified Datalog$^\neg$ databases, except QSQN-WF [14], are either breadth-first [4, 37, 25, 30] or depth-first (and recursive) [26, 41, 37, 15, 40]. There are cases when these control strategies are not the best ones. QSQN-WF [14] is an evaluation framework for (general) Datalog$^\neg$ under the well-founded semantics and is not efficient when applied to stratified Datalog$^\neg$ because it would execute a considerable amount of redundant computation in order to guarantee that the alternative fixpoint has been reached.

In this paper, we have developed QSQN-STR as a novel evaluation framework for stratified Datalog$^\neg$. It is goal-driven, set-at-a-time and adjustable w.r.t. control

strategies. These properties are important for efficient query evaluation on large and complex deductive databases. Every control strategy admissible w.r.t. strata's stability can be used for QSQN-STR. The generic method QSQN-STR is sound, complete and has a PTIME data complexity for evaluating queries to stratified Datalog$^\neg$ databases.

QSQN-STR extends QSQN [11, 9] with the ability to handle stratified negation. Restricting to Datalog, QSQN is similar to QSQ [43, 1] but has some essential differences. First, it is formulated so that all operations can be precisely specified. Second, it does not use adornments for intensional predicates and their corresponding *input* and *answer* relations. This has some advantages:

- Operations of the same kind on an intensional predicate can be grouped and done together, independently from the adornments. This allows reducing the number of accesses to the secondary storage. The matter of how to efficiently execute the evaluation by using relational operations like join and projection is left for the implementation phase.
- *Input* relations contain tuples of terms possibly with variables, and information about repeats of variables in a goal is exploited. In the case of QSQ, *input* relations contain tuples of constant symbols, and only the annotated version of QSQ keeps and exploits information about repeats of variables in a goal.
- Only the most general tuples are kept in *input* relations. (Similarly, only the most general tuples and subqueries are kept in the other relations.) This allows reducing redundant computation. In the case of QSQ, tuples from different adorned *input* relations of the same intensional predicate are not compared to each other, and thus QSQ executes certain amount of redundant recomputation.

QSQN-STR inherits the aforementioned good properties of QSQN. As future work, further (conditional) optimizations can be incorporated into QSQN-STR. For example, we can extend QSQN-STR with tail-recursion elimination [38] in the way as QSQN-TRE [10, 9] extends QSQN.

Acknowledgements

We would like to thank the anonymous reviewers for many comments and suggestions, which have helped us to improve the paper significantly.

REFERENCES

[1] Abiteboul, S.—Hull, R.—Vianu, V.: Foundations of Databases. Addison Wesley, 1995.

[2] Apt, K. R.—Blair, H. A.—Walker, A.: Towards a Theory of Declarative Knowledge. Chapter 2. In: Minker, J. (Ed.): Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, 1988, pp. 89–148, doi: 10.1016/C2013-0-07699-2. ISBN 978-0-934613-40-8.

[3] Apt, K. R.—Bol, R. N.: Logic Programming and Negation: A Survey. The Journal of Logic Programming, Vol. 19-20, Suppl. 1, 1994, pp. 9–71, doi: 10.1016/0743-1066(94)90024-8.

[4] Balbin, I.—Port, G. S.—Ramamohanarao, K.—Meenakshi, K.: Efficient Bottom-Up Computation of Queries on Stratified Databases. The Journal of Logic Programming, Vol. 11, 1991, No. 3-4, pp. 295–344, doi: 10.1016/0743-1066(91)90030-s.

[5] Bancilhon, F.—Maier, D.—Sagiv, Y.—Ullman, J. D.: Magic Sets and Other Strange Ways to Implement Logic Programs. Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS 1986), ACM, 1986, pp. 1–15, doi: 10.1145/6012.15399.

[6] Beeri, C.—Ramakrishnan, R.: On the Power of Magic. The Journal of Logic Programming, Vol. 10, 1991, No. 3-4, pp. 255–299, doi: 10.1016/0743-1066(91)90038-q.

[7] Calì, A.—Gottlob, G.—Lukasiewicz, T.: A General Datalog-Based Framework for Tractable Query Answering over Ontologies. Journal of Web Semantics, Vol. 14, 2012, pp. 57–83, doi: 10.1016/j.websem.2012.03.001.

[8] Cao, S. T.: Query-Subquery Nets with Stratified Negation. In: Le, T. H., Nguyen, N., Do, T. (Eds.): Advanced Computational Methods for Knowledge Engineering. Springer, Cham, Advances in Intelligent Systems and Computing, Vol. 358, 2015, pp. 355–366, doi: 10.1007/978-3-319-17996-4_32.

[9] Cao, S. T.: Methods for Evaluating Queries to Horn Knowledge Bases in First-Order Logic. Ph.D. thesis, University of Warsaw, 2016.

[10] Cao, S. T.—Nguyen, L. A.: An Empirical Approach to Query-Subquery Nets with Tail-Recursion Elimination. In: Bassiliades, N. et al. (Eds.): New Trends in Database and Information Systems II (ADBIS 2014). Springer, Cham, Advances in Intelligent Systems and Computing, Vol. 312, 2015, pp. 109–120, doi: 10.1007/978-3-319-10518-5_9.

[11] Cao, S. T.—Nguyen, L. A.: Query-Subquery Nets for Horn Knowledge Bases in First-Order Logic. Journal of Information and Telecommunication, Vol. 1, 2017, No. 1, pp. 79–99, doi: 10.1080/24751839.2017.1295664.

[12] Cao, S. T.—Nguyen, L. A.: Online Appendix: A Demonstration of the QSQN-STR Method for Evaluating Queries to Stratified Datalog$^{\neg}$. Available at: `http://mimuw.edu.pl/~nguyen/QSQN-STR-demonstration.pdf`, 2018.

[13] Cao, S. T.—Nguyen, L. A.: An Online Appendix for the Current Paper. Available at: `http://mimuw.edu.pl/~nguyen/QSQN-STR-appendix.pdf`, 2018.

[14] Cao, S. T.—Nguyen, L. A.—Nguyen, N. T.: Extending Query-Subquery Nets for Deductive Databases under the Well-Founded Semantics. Cybernetics and Systems, Vol. 48, 2017, No. 3, pp. 249–266, doi: 10.1080/01969722.2016.1276777.

[15] Chen, W.—Swift, T.—Warren, D. S.: Efficient Top-Down Computation of Queries under the Well-Founded Semantics. The Journal of Logic Programming, Vol. 24, 1995, No. 3, pp. 161–199, doi: 10.1016/0743-1066(94)00028-5.

[16] Dung, P. M.: On the Relations Between Stable and Well-Founded Semantics of Logic Programs. Theoretical Computer Science, Vol. 105, 1992, No. 1, pp. 7–25, doi: 10.1016/0304-3975(92)90285-n.

[17] Gebser, M.—Kaufmann, B.—Neumann, A.—Schaub, T.: *clasp*: A Conflict-Driven Answer Set Solver. In: Baral, C., Brewka, G., Schlipf, J. (Eds.): Logic Programming and Nonmonotonic Reasoning (LPNMR 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4483, 2007, pp. 260–265, doi: 10.1007/978-3-540-72200-7_23.

[18] Van Gelder, A.: The Alternating Fixpoint of Logic Programs with Negation. Journal of Computer and System Sciences, Vol. 47, 1993, No. 1, pp. 185–221, doi: 10.1016/0022-0000(93)90024-q.

[19] Van Gelder, A.—Ross, K. A.—Schlipf, J. S.: The Well-Founded Semantics for General Logic Programs. Journal of the ACM, Vol. 38, 1991, No. 3, pp. 619–649, doi: 10.1145/116825.116838.

[20] Gelfond, M.—Lifschitz, V.: The Stable Model Semantics for Logic Programming. Proceedings of International Logic Programming Conference and Symposium (ICLP/SLP 1988), MIT Press, 1988, pp. 1070–1080.

[21] Gire, F.: Equivalence of Well-Founded and Stable Semantics. The Journal of Logic Programming, Vol. 21, 1994, No. 2, pp. 95–111, doi: 10.1016/0743-1066(94)90002-7.

[22] Greco, S.—Molinaro, C.—Trubitsyna, I.—Zumpano, E.: NP Datalog: A Logic Language for Expressing Search and Optimization Problems. Theory and Practice of Logic Programming, Vol. 10, 2010, No. 2, pp. 125–166, doi: 10.1017/S1471068409990251.

[23] Green, T. J.—Huang, S. S.—Loo, B. T.—Zhou, W.: Datalog and Recursive Query Processing. Foundations and Trends in Databases, Vol. 5, 2013, No. 2, pp. 105–195, doi: 10.1561/1900000017.

[24] Huang, S. S.—Green, T. J.—Loo, B. T.: Datalog and Emerging Applications: An Interactive Tutorial. Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD 2011), ACM, 2011, pp. 1213–1216, doi: 10.1145/1989323.1989456.

[25] Kemp, D. B.—Srivastava, D.—Stuckey, P. J.: Bottom-Up Evaluation and Query Optimization of Well-Founded Models. Theoretical Computer Science, Vol. 146, 1995, Nos. 1–2, pp. 145–184, doi: 10.1016/0304-3975(94)00153-a.

[26] Kemp, D. B.—Topor, R. W.: Completeness of a Top-Down Query Evaluation Procedure for Stratified Databases. Proceedings of International Logic Programming Conference and Symposium (ICLP/SLP 1988), MIT Press, 1988, pp. 178–194.

[27] Leone, N.—Pfeifer, G.—Faber, W.—Eiter, T.—Gottlob, G.—Perri, S.—Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computational Logic, Vol. 7, 2006, No. 3, pp. 499–562, doi: 10.1145/1149114.1149117.

[28] Lloyd, J. W.: Foundations of Logic Programming. 2nd edition. Springer, 1987, doi: 10.1007/978-3-642-83189-8.

[29] Madalińska-Bugaj, E.—Nguyen, L. A.: A Generalized QSQR Evaluation Method for Horn Knowledge Bases. ACM Transactions on Computational Logic, Vol. 13, 2012, No. 4, Art. No. 32, doi: 10.1145/2362355.2362360.

[30] Morishita, S.: An Extension of Van Gelder's Alternating Fixpoint to Magic Programs. Journal of Computer and System Sciences, Vol. 52, 1996, No. 3, pp. 506–521, doi: 10.1006/jcss.1996.0038.

[31] Nejdl, W.: Recursive Strategies for Answering Recursive Queries – the RQA/FQI Strategy. Proceedings of the 13th International Conference on Very Large Data Bases (VLDB '87), Morgan Kaufmann, 1987, pp. 43–50, doi: 10.1007/978-3-642-46620-5_3.

[32] Nguyen, L. A.—Cao, S. T.: Query-Subquery Nets. CoRR, abs/1201.2564, 2012.

[33] Nilsson, U.—Maluszynski, J.: Logic, Programming and Prolog. 2nd edition. John Wiley and Sons, Inc., 1995. ISBN: 978-0471959960.

[34] Przymusinski, T. C.: Every Logic Program Has a Natural Stratification and an Iterated Least Fixed Point Model. Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1989), ACM Press, 1989, pp. 11–21, doi: 10.1145/73721.73723.

[35] Przymusinski, T. C.: The Well-Founded Semantics Coincides with the Three-Valued Stable Semantics. Fundamenta Informaticae, Vol. 13, 1990, No. 4, pp. 445–463.

[36] Ramamohanarao, K.—Harland, J.: An Introduction to Deductive Database Languages and Systems. The VLDB Journal, Vol. 3, 1994, No. 2, pp. 107–122, doi: 10.1007/bf01228878.

[37] Ross, K. A.: Modular Stratification and Magic Sets for Datalog Programs with Negation. Journal of the ACM, Vol. 41, 1994, No. 6, pp. 1216–1266, doi: 10.1145/195613.195646.

[38] Ross, K. A.: Tail Recursion Elimination in Deductive Databases. ACM Transactions on Database Systems, Vol. 21, 1996, No. 2, pp. 208–237, doi: 10.1145/232616.232628.

[39] Sáenz-Pérez, F.: DES: A Deductive Database System. Electronic Notes in Theoretical Computer Science, Vol. 271, 2011, pp. 63–78, doi: 10.1016/j.entcs.2011.02.011.

[40] Sagonas, K. F.—Swift, T.—Warren, D. S.: XSB as an Efficient Deductive Database Engine. Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data (SIGMOD '94), ACM Press, 1994, pp. 442–453.

[41] Seki, H.—Itoh, H.: A Query Evaluation Method for Stratified Programs under the Extended CWA. Proceedings of International Logic Programming Conference and Symposium (ICLP/SLP 1988), MIT Press, 1988, pp. 195–211.

[42] Tamaki, H.—Sato, T.: OLD Resolution with Tabulation. In: Shapiro, E. (Ed.): Third International Conference on Logic Programming (ICLP 1986). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 225, 1986, pp. 84–98, doi: 10.1007/3-540-16492-8_66.

[43] Vieille, L.: Recursive Axioms in Deductive Databases: The Query/Subquery Approach. Proceedings of the First International Conference on Expert Database Systems, 1986, pp. 179–194.

[44] Vieille, L.: Recursive Query Processing: The Power of Logic. Theoretical Computer Science, Vol. 69, 1989, No. 1, pp. 1–53, doi: 10.1016/0304-3975(89)90088-1.

Son Thanh Cao received his Ph.D. degree in computer science from the University of Warsaw in 2016. He is currently Lecturer at Vinh University in Vietnam. His research interests include logic programming, deductive database, description logic, semantic Web and artificial intelligence.

Linh Anh Nguyen received the degrees of Ph.D. and Dr.Sc. (Habilitation) in computer science from the University of Warsaw in 2000 and 2009, respectively. He is currently Associate Professor with the Institute of Informatics, University of Warsaw in Poland. Since 2014 he has been cooperating with the Faculty of Information Technology, Ton Duc Thang University in Vietnam. His research interests include modal and description logics, automated reasoning, rule-based languages, deductive databases, and concept learning.

Online Appendix

for the Paper "Incorporating Stratified Negation into Query-Subquery Nets for Evaluating Queries to Stratified Deductive Databases" [3]

This appendix contains:
- an example illustrating the QSQN-STR method (Section A),
- proofs of data complexity, soundness and completeness of the QSQN-STR method (Section B),
- experimental results (Section C), and
- the pseudocode of the QSQN-STR method (Section D).

A An Illustrative Example

The aim of this example is to illustrate how Algorithm 1 works step by step. It uses the stratified Datalog$^\neg$ database (P, I) and the stratification $P = P_1 \cup P_2$ given in [3, Example 2.2]. The QSQN-STR topological structure of the program P is illustrated in Figure 1.

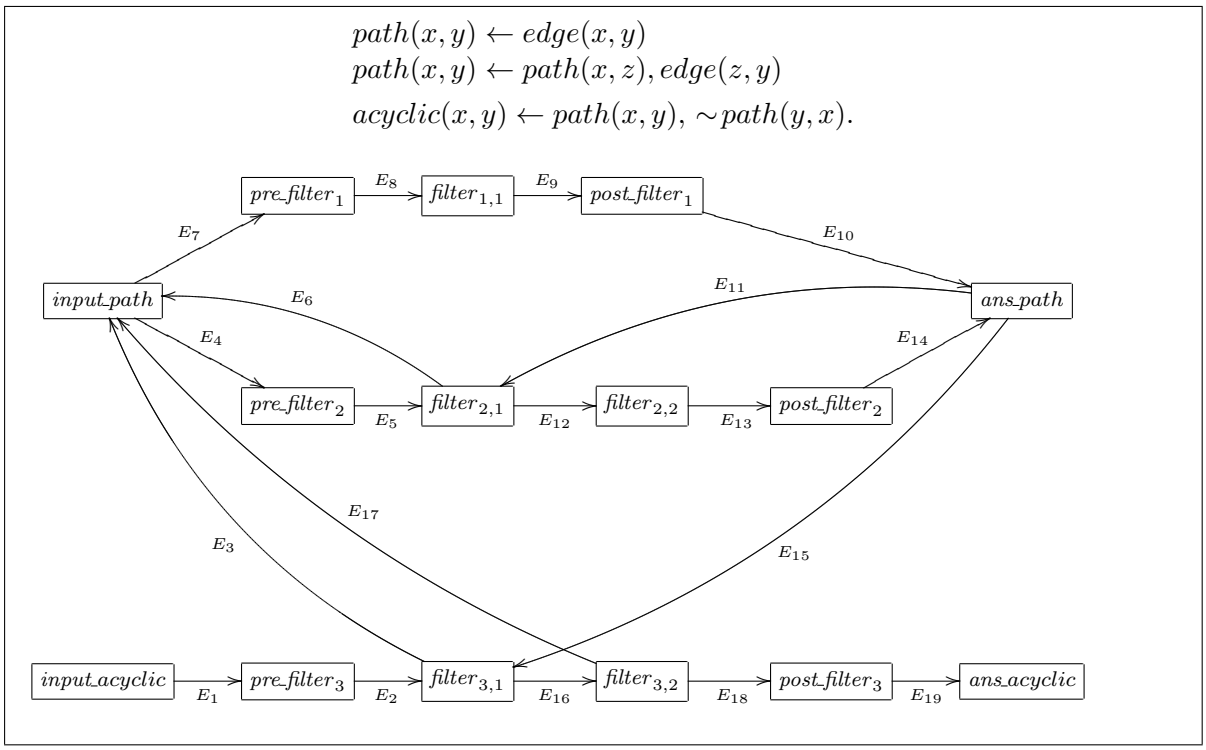
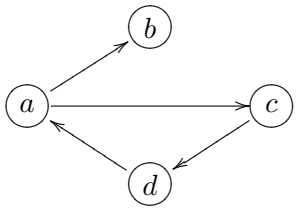


Fig. 1. The QSQN-STR topological structure of the program given in [3, Example 2.2]

Recall that the extensional instance I is specified by $I(edge) = \{(a, b),(a, c),(c, d),(d, a)\}$ and illustrated below:



We give below a trace of running Algorithm 1 for the query $acyclic(x, y)$ on the stratified Datalog$^\neg$ database (P, I). For convenience, we name the edges of the net by E_i with $1 \leq i \leq 19$,

as shown in Figure 1. Let $T(v) = \mathit{false}$ for each $v = \mathit{filter}_{i,j}$ with $\mathit{kind}(v) = \mathit{extensional}$. Assume that Algorithm 1 evaluates the query $\mathit{acyclic}(x, y)$ to the knowledge base (P, I) using a control strategy that selects active edges for "firing" in the order (E_1, E_3, E_4, E_6, E_7, E_{11}, E_{12}, E_{11}, E_{12}, E_{11}, E_{12}, E_{15}, E_{16}, E_{17}, E_{18}), which is admissible w.r.t. strata's stability and corresponds to the IDFS2 control strategy specified in [1]. Recall that, by [3, Example 2.2], the set of answers should be $\{(a, b), (c, b), (d, b)\}$. To ease the checking, the reader can use the full pseudocode of Algorithm 1 gathered in Section D. In addition, a much more friendly presentation of this example in the PowerPoint-like mode is available online [4].

Algorithm 1 starts with an empty QSQN-STR and then adds a fresh variant (x_1, y_1) of (x, y) to the empty sets $\mathit{tuples}(\mathit{input\_acyclic})$ and $\mathit{unprocessed}(E_1)$. This makes the edge E_1 active.

1. **E_1–E_2**

   Firing the active edge E_1 (by $\texttt{fire}'$), the algorithm transfers (x_1, y_1) through this edge and empties the set $\mathit{unprocessed}(E_1)$. This produces $\{((x_1, y_1), \{x/x_1, y/y_1\})\}$ (by $\texttt{transfer}'_3$ via $\texttt{transfer}'$), which is then transferred through the edge E_2 and added to the empty sets $\mathit{subqueries}(\mathit{filter}_{3,1})$, $\mathit{unprocessed\_subqueries}(\mathit{filter}_{3,1})$ and $\mathit{unprocessed\_subqueries}_2(\mathit{filter}_{3,1})$ (by $\texttt{transfer}_5$ via $\texttt{transfer}'$).

2. **E_3**

   Firing the active edge E_3 (by $\texttt{fire}'_1$ via $\texttt{fire}'$), the algorithm empties the set $\mathit{unprocessed\_subqueries}_2(\mathit{filter}_{3,1})$ and transfers (x_2, y_2), a fresh variant of (x_1, y_1), through the edge E_3 and adds its fresh variant (x_3, y_3) to the empty sets $\mathit{tuples}(\mathit{input\_path})$, $\mathit{unprocessed}(E_4)$ and $\mathit{unprocessed}(E_7)$ (by $\texttt{transfer}_2$ via $\texttt{transfer}'$).

3. **E_4–E_5**

   Firing the active edge E_4 (by $\texttt{fire}'$), the algorithm transfers (x_3, y_3) through this edge and empties the set $\mathit{unprocessed}(E_4)$. This produces $\{((x_3, y_3), \{x/x_3, y/y_3\})\}$ (by $\texttt{transfer}'_3$ via $\texttt{transfer}'$), which is then transferred through the edge E_5 and added to the empty sets $\mathit{subqueries}(\mathit{filter}_{2,1})$, $\mathit{unprocessed\_subqueries}(\mathit{filter}_{2,1})$ and $\mathit{unprocessed\_subqueries}_2(\mathit{filter}_{2,1})$ (by $\texttt{transfer}_5$ via $\texttt{transfer}'$).

4. **E_6**

   Firing the active edge E_6 (by $\texttt{fire}'_1$ via $\texttt{fire}'$), the algorithm empties the set $\mathit{unprocessed\_subqueries}_2(\mathit{filter}_{2,1})$ and transfers (x_4, z_4), a fresh variant of (x_3, z), through the edge E_6 and adds nothing to $\mathit{tuples}(\mathit{input\_path})$ (by $\texttt{transfer}_2$ via $\texttt{transfer}'$), because there exists $(x_3, y_3) \in \mathit{tuples}(\mathit{input\_path})$, which is more general than any other tuples.

5. **E_7–E_8–E_9–E_{10}**

   Firing the active edge E_7 (by $\texttt{fire}'$), the algorithm transfers (x_3, y_3) through this edge and empties the set $\mathit{unprocessed}(E_7)$. This produces $\{((x_3, y_3), \{x/x_3, y/y_3\})\}$ (by $\texttt{transfer}'_3$ via $\texttt{transfer}'$), which is then transferred through the edge E_8, producing $\{((a, b), \varepsilon), ((a, c), \varepsilon), ((c, d), \varepsilon), ((d, a), \varepsilon)\}$ (by $\texttt{transfer}'_4$ via $\texttt{transfer}'$), which is then transferred through the edge E_9, producing $\{(a, b), (a, c), (c, d), (d, a)\}$ (by $\texttt{transfer}'$), which in turn is then transferred through the edge E_{10} and added to the empty sets $\mathit{tuples}(\mathit{ans\_path})$, $\mathit{unprocessed}(E_{11})$ and $\mathit{unprocessed}(E_{15})$ (by $\texttt{transfer}_1$ via $\texttt{transfer}'$).

6. **E_{11}**

   Firing the active edge E_{11} (by $\texttt{fire}'$), the algorithm transfers $\{(a, b), (a, c), (c, d), (d, a)\}$ through this edge and empties the set $\mathit{unprocessed}(E_{11})$. This adds those tuples to the empty set $\mathit{unprocessed\_tuples}(\mathit{filter}_{2,1})$ (by $\texttt{transfer}'$).

7. **E_{12}–E_{13}–E_{14}**

   Firing the active edge E_{12} (by $\texttt{fire}'_3$ via $\texttt{fire}'$) and processing the sets $\mathit{unprocessed\_subqueries}(\mathit{filter}_{2,1})$ and $\mathit{unprocessed\_tuples}(\mathit{filter}_{2,1})$, the algorithm empties these sets and produces the set of subqueries $\{((a, y_3), \{y/y_3, z/b\}), ((a, y_3), \{y/y_3, z/c\}), ((c, y_3), \{y/y_3, z/d\}), ((d, y_3), \{y/y_3, z/a\})\}$, which is transferred through the edge E_{12}, producing $\{((a, d), \varepsilon), ((c, a), \varepsilon), ((d, b), \varepsilon), ((d, c), \varepsilon)\}$ (by $\texttt{transfer}'_4$ via $\texttt{transfer}'$), which is then transferred through the edge E_{13}, producing $\{(a, d), (c, a), (d, b), (d, c)\}$ (by $\texttt{transfer}'$), which in turn is then transferred through the edge E_{14} and added to the sets $\mathit{tuples}(\mathit{ans\_path})$, $\mathit{unprocessed}(E_{11})$ and $\mathit{unprocessed}(E_{15})$ (by $\texttt{transfer}'_1$ via $\texttt{transfer}'$). After these steps, we have:

   - $\mathit{tuples}(\mathit{ans\_path}) = \{(a, b), (a, c), (c, d), (d, a), (a, d), (c, a), (d, b), (d, c)\}$,
   - $\mathit{unprocessed}(E_{11}) = \{(a, d), (c, a), (d, b), (d, c)\}$,
   - $\mathit{unprocessed}(E_{15}) = \mathit{tuples}(\mathit{ans\_path})$.

8. **E_{11} and E_{12}–E_{13}–E_{14}**

   The algorithm repeatedly fires the edges E_{11} and E_{12} until no new tuple is added to $\mathit{tuples}(\mathit{ans\_path})$, $\mathit{unprocessed}(E_{11})$ and $\mathit{unprocessed}(E_{15})$. This takes two rounds and after such steps, the edges E_{11} and E_{12} become inactive and we have:

   - $\mathit{tuples}(\mathit{ans\_path}) = \mathit{unprocessed}(E_{15}) = \{(a, b),\ (a, c),\ (c, d),\ (d, a),\ (a, d),\ (c, a),\ (d, b),\ (d, c),\ (a, a),\ (c, b),\ (c, c),\ (d, d)\}$.

9. **E_{15}**

   Firing the active edge E_{15} (by $\texttt{fire}'$), the algorithm transfers the aforementioned tuples of $\mathit{unprocessed}(E_{15})$ through this edge and empties the set $\mathit{unprocessed}(E_{15})$. This adds those tuples to the empty set $\mathit{unprocessed\_tuples}(\mathit{filter}_{3,1})$ (by $\texttt{transfer}'$). After these steps, we have:

   - $\mathit{subqueries}(\mathit{filter}_{3,1}) = \mathit{unprocessed\_subqueries}(\mathit{filter}_{3,1}) = \{((x_1, y_1), \{x/x_1, y/y_1\})\}$,
   - $\mathit{unprocessed\_tuples}(\mathit{filter}_{3,1}) = \{(a, b),\ (a, c),\ (c, d),\ (d, a),\ (a, d),\ (c, a),\ (d, b),\ (d, c),\ (a, a),\ (c, b),\ (c, c),\ (d, d)\}$.

10. **E_{16}**

    Firing the active edge E_{16} (by $\texttt{fire}'_3$ via $\texttt{fire}'$) and processing the sets $\mathit{unprocessed\_subqueries}(\mathit{filter}_{3,1})$ and $\mathit{unprocessed\_tuples}(\mathit{filter}_{3,1})$, the algorithm empties these sets and produces the set of subqueries $\{((a, b), \{x/a, y/b\}), ((a, c), \{x/a, y/c\}), ((c, d), \{x/c, y/d\}), ((d, a), \{x/d, y/a\}), ((a, d), \{x/a, y/d\}), ((c, a), \{x/c, y/a\}), ((d, b), \{x/d, y/b\}), ((d, c), \{x/d, y/c\}), ((a, a), \{x/a, y/a\}), ((c, b), \{x/c, y/b\}), ((c, c), \{x/c, y/c\}), ((d, d), \{x/d, y/d\})\}$, which is transferred through the edge E_{16} and added to the empty sets $\mathit{subqueries}(\mathit{filter}_{3,2})$, $\mathit{unprocessed\_subqueries}(\mathit{filter}_{3,2})$ and $\mathit{unprocessed\_subqueries}_2(\mathit{filter}_{3,2})$ (by $\texttt{transfer}_5$ via $\texttt{transfer}'$).

11. **E_{17}**

    Firing the active edge E_{17} (by $\texttt{fire}'_1$ via $\texttt{fire}'$), the algorithm empties the set $\mathit{unprocessed\_subqueries}_2(\mathit{filter}_{3,2})$ and transfers the set of tuples $\{(b, a), (c, a), (d, c), (a, d), (d, a), (a, c), (b, d), (c, d), (a, a), (b, c), (c, c), (d, d)\}$ through the edge E_{17} and adds nothing to $\mathit{tuples}(\mathit{input\_path})$ (by $\texttt{transfer}_2$ via $\texttt{transfer}'$), because there exists $(x_3, y_3) \in \mathit{tuples}(\mathit{input\_path})$, which is more general than any other tuples.

12. **E_{18}–E_{19}**

    Observe that the edge E_{18} is active, the current net is stable up to the layer 1 (no edge among E_4–E_{14} is active), and the edge E_{17} is inactive. Thus, selecting the edge E_{18} (for firing) satisfies the admissibility w.r.t. strata's stability. Firing this edge (by $\texttt{fire}'_4$ via $\texttt{fire}'$) and processing $\mathit{unprocessed\_subqueries}(\mathit{filter}_{3,2})$, the algorithm empties this set and produces the set of subqueries $\{((a,b),\{\varepsilon\}), ((d,b),\{\varepsilon\}), ((c,b),\{\varepsilon\})\}$, which is transferred through the edge E_{18}, producing $\{(a,b), (d,b), (c,b)\}$ (by $\texttt{transfer}'$), which, in turn, is then transferred through the edge E_{19} and added to the empty set $\mathit{tuples}(\mathit{ans\_acyclic})$ (by $\texttt{transfer}_1$ via $\texttt{transfer}'$).

At this point, no edge in the net is active. The algorithm terminates and returns the set $\mathit{tuples}(\mathit{ans\_acyclic}) = \{(a,b), (d,b), (c,b)\}$.

B Data Complexity, Soundness and Completeness

In this section, we prove that the QSQN-STR evaluation method for stratified Datalog$^\neg$ is sound, complete and has a PTIME data complexity.

The following lemma states a property of Algorithm 1.

Lemma 1. *For every intensional predicate r used in P, if $\bar{t} \in \mathit{tuples}(\mathit{ans\_}r)$, then $\bar{t}$ is a ground tuple (i.e., a tuple without variables).*

This property follows from the safety conditions of the Datalog$^\neg$ program P. Technically, one can prove it by induction on the moment of adding $\bar{t}$ to $\mathit{tuples}(\mathit{ans\_}r)$ and an inner induction on j that, if a subquery $(\bar{t}', \delta)$ is transferred to a node $\mathit{filter}_{i,j}$, where the predicate of A_i is r, then $\mathit{Vars}(\bar{t}') \subseteq \mathit{Vars}((B_{i,j}, \ldots, B_{i,n_i})\delta)$ and, for every $x \in \mathit{Vars}((B_{i,1}, \ldots, B_{i,j-1})) \cap \mathit{Vars}((B_{i,j}, \ldots, B_{i,n_i}))$, $\delta(x)$ is a constant (i.e., δ contains a pair x/c for some constant c). Additionally, as the next step, if a subquery $(\bar{t}, \delta)$ is transferred to the node $\mathit{post\_filter}_i$, then $\delta = \varepsilon$ and $\bar{t}$ is a ground tuple. The proof is straightforward and omitted.

Lemma 2. *Algorithm 1 runs in polynomial time in the size of I.*

Proof. Let n be the size of I. Without loss of generality, we assume that all intensional predicates of the signature are used in P. As the Datalog$^\neg$ program P is fixed, the arities of all intensional predicates are bounded by a constant, and the number of constant symbols occurring in P is also bounded by a constant.

For each intensional predicate p, let $\mathit{all\_tuples}(\mathit{input\_}p)$ denote the set of all tuples that are added to $\mathit{tuples}(\mathit{input\_}p)$ during the run of the algorithm (including the ones that are deleted from $\mathit{tuples}(\mathit{input\_}p)$ at some later steps). The cardinality of $\mathit{all\_tuples}(\mathit{input\_}p)$ is bounded by a polynomial in n. The reasons are as follows:

- Let k be the arity of p. Before a tuple $\bar{t}$ is added to $\mathit{tuples}(\mathit{input\_}p)$, $\bar{t}$ is not an instance of a fresh variant of any $\bar{t}' \in \mathit{tuples}(\mathit{input\_}p)$, hence there exists a renaming substitution $\theta_{\bar{t}}$ such that $\mathit{dom}(\theta_{\bar{t}}) = \mathit{Vars}(\bar{t})$, $\mathit{range}(\theta_{\bar{t}}) \subseteq \{x_1, \ldots, x_k\}$ and $\bar{t}\theta_{\bar{t}}$ is not an instance of a fresh variant of any $\bar{t}' \in \mathit{tuples}(\mathit{input\_}p)$. When a tuple is deleted from $\mathit{tuples}(\mathit{input\_}p)$, its fresh variant must be an instance of a tuple that will be added to $\mathit{tuples}(\mathit{input\_}p)$ at the next step. Hence, if $\{\bar{t}, \bar{t}'\} \subseteq \mathit{all\_tuples}(\mathit{input\_}p)$ and $\bar{t} \neq \bar{t}'$, then $\bar{t}\theta_{\bar{t}} \neq \bar{t}'\theta_{\bar{t}'}$.
- The sets $\mathit{all\_tuples}(\mathit{input\_}p)$ and $\{\bar{t}\theta_{\bar{t}} \mid \bar{t} \in \mathit{all\_tuples}(\mathit{input\_}p)\}$ have the same cardinality, which is bounded by a polynomial in n because each element of the latter set is a k-ary tuple constructed from the variables $x_1, \ldots, x_k$ and the constants occurring in $P \cup I$.

For each intensional predicate p, the cardinality of $\mathit{tuples}(\mathit{ans\_}p)$ is also bounded by a polynomial in n. This follows from Lemma 1.

Each "elementary operation" executed by the algorithm is related to a $\bar{t} \in \mathit{all\_tuples}(\mathit{input\_p})$ for some p and can be labeled by the pair $(\bar{t}, p)$, which is chosen so that $\bar{t} \in \mathit{all\_tuples}(\mathit{input\_p})$ is the most direct cause of the operation. Observe that, for each $(\bar{t}, p)$, the number of "elementary operations" executed by the algorithm and labeled by $(\bar{t}, p)$ is bounded by a polynomial in n. As the cardinality of $\mathit{all\_tuples}(\mathit{input\_p})$ for each intensional predicate p is bounded by a polynomial in n, we conclude that the algorithm runs in polynomial time in n. ∎

We will need the well-known Lifting Lemma, whose restriction to Datalog is presented below. Its proof can be found in [9].

Lemma 3 (Lifting Lemma). *Let P be a Datalog program, G a goal and θ a substitution. Suppose there exists an SLD-refutation of $P \cup \{G\theta\}$ using mgu's $\theta_1, \ldots, \theta_n$ such that the variables of the input program clauses are distinct from the variables in G and θ. Then, there exist a substitution γ and an SLD-refutation of $P \cup \{G\}$ using the same sequence of input program clauses, the same selected atoms, and mgu's $\theta'_1, \ldots, \theta'_n$ such that $\theta\theta_1 \ldots \theta_n = \theta'_1 \ldots \theta'_n\gamma$.*

For each predicate p of the signature (now called the *primary signature*), let p' be a new extensional predicate (for playing the role of $\sim p$). For each $1 \leq k \leq K$, let P'_k be the Datalog program obtained from P_k by replacing every $\sim p$ with p'. For each $0 \leq k \leq K$, let $M_k = M_{P_1 \cup \ldots \cup P_k, I}$ and let I_k be the instance of extensional predicates specified as follows:

- if p is an extensional predicate from the primary signature, then $I_k(p) = I(p)$;
- if p is an h-ary predicate from the primary signature, then $I_k(p') = \{\bar{t} \mid \bar{t}$ is an h-ary tuple of constants from $U_{P,I}$ such that $p(\bar{t}) \notin M_k\}$.

Lemma 4. *For every $1 \leq k \leq K$, every intensional predicate p of the primary signature and every tuple $\bar{t}$ of constants, $p(\bar{t}) \in M_k$ iff $p(\bar{t}) \in T_{P'_1 \cup \ldots \cup P'_k, I_{k-1}} \uparrow \omega$.*

This lemma immediately follows from the fact that the standard semantics of stratified Datalog$^\neg$ agrees with the stable model semantics [5].

Lemma 5. *During a run of Algorithm 1, for every intensional predicate r of P with $\mathit{layer}(\mathit{input\_r}) = k$ and for every tuples $\bar{t}$ and $\bar{t}'$ of terms,*

a) if $\bar{t} \in \mathit{tuples}(\mathit{ans\_r})$, then $r(\bar{t}) \in M_{P,I}$,

b) if the QSQ-STR-net is stable up to the layer k, $\bar{t} \in \mathit{tuples}(\mathit{input\_r})$, $r(\bar{t}') \in M_{P,I}$ and $\bar{t}'$ is an instance of $\bar{t}$, then $\bar{t}' \in \mathit{tuples}(\mathit{ans\_r})$.

Proof. We prove this lemma by induction on k. The base case $k = 0$ is trivial.

For the induction step, we first show that a run of the QSQN-STR method for the Datalog$^\neg$ program $P_1 \cup \ldots \cup P_k$ can be treated as a run of the QSQN method for the Datalog program $P'_1 \cup \ldots \cup P'_k$ by considering each $\sim p$ as the extensional predicate p' specified by I_{k-1}. For this, we only need to show that if $\mathit{pred}(A_i) = r$, $\mathit{kind}(\mathit{filter}_{i,j}) = \mathit{intensional}$, $\mathit{neg}(\mathit{filter}_{i,j}) = \mathit{true}$, $\mathit{pred}(\mathit{filter}_{i,j}) = p$, $\mathit{layer}(\mathit{input\_p}) = h$ and $h < k$, then:

(i) if $\bar{t} \in \mathit{tuples}(\mathit{ans\_p})$, then $p'(\bar{t}) \notin I_{k-1}$,

(ii) if $(\bar{t}_{j-1}, \delta_{j-1}) \in \mathit{subqueries}(\mathit{filter}_{i,j})$, then for every tuple $\bar{t}$ of constants from $U_{P,I}$ such that $p(\bar{t})$ is an instance of $\mathit{atom}(\mathit{filter}_{i,j})\delta_{j-1}$ and $p'(\bar{t}) \notin I_{k-1}$, $\bar{t}$ was added by Algorithm 1 to $\mathit{tuples}(\mathit{ans\_p})$ at some step before the subquery $(\bar{t}_{j-1}, \delta_{j-1})$ is processed for the edge $(\mathit{filter}_{i,j}, \mathit{succ}(\mathit{filter}_{i,j}))$.

Assume that the premises of the main implication hold. Consider the assertion (i) and assume that $\bar{t} \in \mathit{tuples}(\mathit{ans\_p})$. By the inductive assumption (a), $p(\bar{t}) \in M_{P,I}$. Thus, $p(\bar{t}) \in M_{k-1}$ and hence $p(\bar{t}) \notin I_{k-1}$. For the assertion (ii), assume that $(\bar{t}_{j-1}, \delta_{j-1}) \in \mathit{subqueries}(\mathit{filter}_{i,j})$, $\bar{t}$ is a tuple of constants from $U_{P,I}$ such that $p(\bar{t})$ is an instance of $\mathit{atom}(\mathit{filter}_{i,j})\delta_{j-1}$ and $p'(\bar{t}) \notin I_{k-1}$. We have that $p(\bar{t}) \in M_{k-1}$, and hence $p(\bar{t}) \in M_{P,I}$. Since the used control strategy is admissible

w.r.t. strata's stability, before calling $\texttt{fire}'(\mathit{filter}_{i,j}, \mathit{succ}(\mathit{filter}_{i,j}))$, the subquery $(\bar{t}_{j-1}, \delta_{j-1})$ has already been processed for the edge $(\mathit{filter}_{i,j}, \mathit{input\_p})$ and, as a consequence, $\mathit{tuples}(\mathit{input\_p})$ contains a tuple $\bar{t}''$ such that a fresh variant of $\mathit{atom}(\mathit{filter}_{i,j})\delta_{j-1}$ is an instance of $p(\bar{t}'')$. Thus, $\bar{t}$ is an instance of $\bar{t}''$. Furthermore, at that moment the QSQ-STR-net is stable up to the layer h. By the inductive assumption (b) for h instead of k and p, $\bar{t}''$, $\bar{t}$ instead of r, $\bar{t}$, $\bar{t}'$, respectively, we have that $\bar{t} \in \mathit{tuples}(\mathit{ans\_p})$, which completes the proof of the assertion (ii).

By the assertions (i) and (ii), we can now treat a run of Algorithm 1 on the part consisting of the layers up to k of the QSQ-STR-net as a run of the QSQN method on a QSQ-net by considering each $\sim p$ as the extensional predicate p' specified by I_{k-1}.[1]

Consider the assertion (a) and assume that the premise of the implication holds. Since $\bar{t} \in \mathit{tuples}(\mathit{ans\_r})$, by the soundness of the QSQN method for Datalog (see [1, Lemma 4.2] for the case when $l = 0$ and $T(p) = \mathit{false}$ for every intensional predicate p), $\bar{t}$ is a correct answer for $(P'_1 \cup \ldots \cup P'_k) \cup I_{k-1} \cup \{\leftarrow r(\bar{t})\}$. By the correctness of the fixpoint semantics of positive logic program (see, e.g., [6, Theorems 6.5 and 6.6]), it follows that $r(\bar{t}) \in T_{P'_1 \cup \ldots \cup P'_k, I_{k-1}}\uparrow\omega$. Hence, by Lemma 4, $r(\bar{t}) \in M_k$, which implies $r(\bar{t}) \in M_{P,I}$. This completes the proof of the assertion (a).

Consider the assertion (b) and assume that the premises of the implication hold. Since $r(\bar{t}') \in M_{P,I}$, we have that $r(\bar{t}') \in M_k$, and by Lemma 4, $r(\bar{t}') \in T_{P'_1 \cup \ldots \cup P'_k, I_{k-1}}\uparrow\omega$. By the correctness of the fixpoint semantics of positive logic program (see, e.g., [6, Theorems 6.5 and 6.6]) and the completeness of SLD-resolution (see, e.g., [6, Theorems 8.6]), there exists an SLD-refutation of $(P'_1 \cup \ldots \cup P'_k) \cup I_{k-1} \cup \{\leftarrow r(\bar{t}')\}$ with mgu's $\theta_1, \ldots, \theta_n$. Since $\bar{t}'$ is an instance of $\bar{t}$, there exists a substitution θ such that $\bar{t}' = \bar{t}\theta$. By the Lifting Lemma 3, there exists an SLD-refutation of $(P'_1 \cup \ldots \cup P'_k) \cup I_{k-1} \cup \{\leftarrow r(\bar{t})\}$ with mgu's $\theta'_1, \ldots, \theta'_n$ such that $\theta\theta_1 \ldots \theta_n = \theta'_1 \ldots \theta'_n\delta$ for some substitution δ. By the completeness of the QSQN method for Datalog (see [1, Lemma 4.3] for the case when $l = 0$ and $T(p) = \mathit{false}$ for every intensional predicate p), $\bar{t}\theta'_1 \ldots \theta'_n$ is an instance of a fresh variant of some tuple $\bar{t}'' \in \mathit{tuples}(\mathit{ans\_r})$. Since $\bar{t}' = \bar{t}'\theta_1 \ldots \theta_n = \bar{t}\theta\theta_1 \ldots \theta_n = \bar{t}\theta'_1 \ldots \theta'_n\delta$ is an instance of $\bar{t}\theta'_1 \ldots \theta'_n$, $\bar{t}'$ is also an instance of $\bar{t}''$. Since $\bar{t}''$ is a ground tuple (by Lemma 1), it follows that $\bar{t}' = \bar{t}''$. This completes the proof of the assertion (b). ∎

Corollary 1. *After a run of Algorithm 1 for a query $q(\bar{x})$ to a stratified Datalog$^\neg$ database (P, I), for every tuple of terms $\bar{t}$, $\bar{t} \in \mathit{tuples}(\mathit{ans\_q})$ iff $q(\bar{t}) \in M_{P,I}$.*

This corollary immediately follows from Lemma 5. Together with Lemma 2, it implies the following theorem.

Theorem 1. *The QSQN-STR method formulated by Algorithm 1 for evaluating queries to stratified Datalog$^\neg$ databases is sound, complete and has a PTIME data complexity.*

C Preliminary Experiments

We have implemented a prototype of QSQN-STR in Java, using a control strategy named IDFS2, which is specified in [1]. We have made a comparison between our prototype of QSQN-STR and Datalog Educational System (DES – a deductive database system) [7] w.r.t. the number of generated tuples in the answer relations that correspond to intensional predicates. The experimental results given in [1, Section 6.5] show that the number of generated tuples in the *answer* relations that correspond to negated intensional predicates in the case of QSQN-STR is often smaller than the one in the case of DES.

Our prototype of QSQN-STR [2] has not yet been optimized. So, in general, it cannot compete with highly optimized engines like XSB [8]. Nevertheless, we have performed experiments and

[1] The edge $(\mathit{filter}_{i,j}, \mathit{input\_p})$ may cause adding more tuples to $\mathit{tuples}(\mathit{input\_p})$, but they do not affect the soundness and completeness of the QSQN method ([1, Lemmas 4.2 and 4.3] for the case when $l = 0$ and $T(p) = \mathit{false}$ for every intensional predicate p).

made a comparison between our prototype of QSQN-STR, DES-DBMS[2] (version 5.0.1) and SWI-Prolog[3] (version 6.4) w.r.t. the execution time by using a number of tests. Comparing our prototype of QSQN-STR with other existing engines is time-consuming and left as future work.[4]

In this section, we present the mentioned comparison between our prototype of QSQN-STR, DES-DBMS and SWI-Prolog. In general, such a comparison does not reveal much about advantages of the used evaluation methods, because these systems use different programming styles and/or languages. Besides, QSQN-STR is a framework that allows every control strategy admissible w.r.t. strata's stability, and our prototype of QSQN-STR adopts the control strategy IDFS2 [1], which may be further improved. The point is not the efficiency of our prototype of QSQN-STR. The aim of our experiments and comparison is only to support the claim that QSQN-STR is a useful evaluation framework for stratified Datalog$^\neg$.

Our prototype of QSQN-STR uses extensional relations stored in a MySQL database. DES-DBMS was also implemented in Java using SQL DBMS'. SWI-Prolog is a well-known logic programming software, which can easily be connected to a MySQL database through an ODBC driver. Without using a MySQL database for storing extensional relations, SWI-Prolog runs very fast. For a fair comparison, however, we performed tests with SWI-Prolog using extensional relations stored in the same MySQL database as for QSQN-STR. For the tests with QSQN-STR, we set $T(v) = \mathit{false}$ for each $v = \mathit{filter}_{i,j} \in V$ with $\mathit{pred}(v) = \mathit{extensional}$.

Let P_1 be the stratified Datalog$^\neg$ program consisting of the following clauses, where link_1, link_2, origin and $\mathit{destination}$ are extensional predicates, $\mathit{reachable}_1$, $\mathit{reachable}_2$, $\mathit{reachable}$, query_1 and query_2 are intensional predicates, x, y and z are variables:

$$\mathit{reachable}_1(x,y) \leftarrow \mathit{link}_1(x,y), \tag{1}$$
$$\mathit{reachable}_1(x,y) \leftarrow \mathit{link}_1(x,z), \mathit{reachable}_1(z,y), \tag{2}$$
$$\mathit{reachable}_2(x,y) \leftarrow \mathit{link}_2(x,y), \tag{3}$$
$$\mathit{reachable}_2(x,y) \leftarrow \mathit{link}_2(x,z), \mathit{reachable}_2(z,y), \tag{4}$$
$$\mathit{reachable}(x,y) \leftarrow \mathit{reachable}_1(x,y), \tag{5}$$
$$\mathit{reachable}(x,y) \leftarrow \mathit{reachable}_2(x,y), \tag{6}$$
$$\mathit{query}_1(x,y) \leftarrow \mathit{origin}(x), \mathit{destination}(y), {\sim}\mathit{reachable}(x,y), \tag{7}$$
$$\mathit{query}_2(x,y) \leftarrow \mathit{origin}(x), \mathit{destination}(y), \mathit{reachable}(x,y), {\sim}\mathit{reachable}(y,x). \tag{8}$$

Let P_2 be the stratified Datalog$^\neg$ program that differs from P_1 in that the clauses (2) and (4) are replaced by the following one, with i being 1 or 2, respectively:

$$\mathit{reachable}_i(x,y) \leftarrow \mathit{reachable}_i(x,z), \mathit{link}_i(z,y).$$

Similarly, let P_3 be the stratified Datalog$^\neg$ program that differs from P_1 in that the clauses (2) and (4) are replaced by the following one, with i being 1 or 2, respectively:

$$\mathit{reachable}_i(x,y) \leftarrow \mathit{reachable}_i(x,z), \mathit{reachable}_i(z,y).$$

[2] The Datalog Education System (DES) with a DBMS via ODBC, available at http://des.sourceforge.net (see also, e.g., [7]).

[3] Available at http://www.swi-prolog.org/

[4] XSB is known as an efficient engine for in-memory Datalog$^\neg$ databases due to the suspension-resumption mechanism, advantages of WAM (Warren Abstract Machine) and other optimizations. We think that our prototype of QSQN-STR cannot compete with XSB when the computation can totally be done in the memory without accessing to the secondary storage. For a comparison with XSB, at least we want to run XSB using very large extensional relations stored on disk. However, at the moment we have a technical problem with connecting XSB to a MySQL DBMS via an ODBC driver. We could not find time for comparing our prototype of QSQN-STR with other engines like DLV [29], NP Datalog [24] and *clasp* [19]. In general, we think that those systems were designed and implemented to deal, among others, with answer set programming (ASP) and, as the main aim of ASP engines is to find an answer set (i.e., a stable model) for a given logic program, they are not goal-driven and, in general, not as efficient as expected for answering queries to stratified Datalog$^\neg$ databases. Of course, without performing experiments and comparisons, nothing can be formally stated.

Let I_1 be the extensional instance specified as follows, where n is a parameter and elements like $a_{i,j}$, o_k and d_k are constant symbols:

$$\begin{aligned} I_1(origin) &= \{o_k \mid 1 \leq k \leq n\}, \\ I_1(destination) &= \{d_k \mid 1 \leq k \leq n\}, \\ I_1(link_1) &= \{(o_k, a_{1,1}), (a_{i,1}, a_{i+1,1}), (a_{n,1}, d_k) \mid 1 \leq k \leq n, 1 \leq i < n\}, \\ I_1(link_2) &= \{(o_k, a_{1,j}), (a_{i,j}, a_{i+1,j}), (a_{n,j}, d_k) \mid 1 \leq k \leq n, 1 \leq i < n, 1 \leq j \leq n\}. \end{aligned}$$

Let I_2 be the extensional instance specified as follows:

$$\begin{aligned} I_2(origin) &= I_1(origin), \\ I_2(destination) &= I_1(destination), \\ I_2(link_1) &= I_1(link_1) \cup \{(a_{i+1,1}, a_{i,1}) \mid 1 \leq i < n\}, \\ I_2(link_2) &= I_1(link_2) \cup \{(a_{i+1,j}, a_{i,j}) \mid 1 \leq i < n, 1 \leq j \leq n\}. \end{aligned}$$

The extensional instances I_1 and I_2 are illustrated in Figures 2 and 3, respectively.
We consider the tests specified as follows.

Test	**Program**	**Extensional Instance**
Test 1	P_1	I_1
Test 2	P_1	I_2
Test 3	P_2	I_1
Test 4	P_2	I_2
Test 5	P_3	I_1
Test 6	P_3	I_2

For each of the tests, we consider the following values of n:

$$1)\ n = 20, \quad 2)\ n = 40, \quad 3)\ n = 60, \quad 4)\ n = 80, \quad 5)\ n = 100$$

and the following queries (cf. [3, Remark 2.3]):

$$a)\ query_1(x, y), \qquad b)\ query_1(o_1, d_1), \qquad c)\ query_2(x, y), \qquad d)\ query_2(o_1, d_1).$$

Our experiments were done using a computer with Windows 10 (64-bit), Intel® Core™ i5-6500 CPU 3.20 GHz and 8 GB RAM. Figures 4–9 show a comparison between our prototype of QSQN-STR, SWI-Prolog and DES-DBMS w.r.t. the execution time for the mentioned tests. The experimental results of SWI-Prolog are shown only for the tests for which SWI-Prolog can terminate properly.[5] For each of the mentioned engines, each test was executed 10 times and the average value of execution time in milliseconds was taken. To give a better data visualization, the execution times are shown after being converted by $\log_{10}$. Detailed instructions for verifying our experiments are included in [2].

The results presented in Figures 4–9 show that our prototype of QSQN-STR outperforms DES-DBMS by a few orders of magnitude in term of execution time for all of the tests. It is competitive with SWI-Prolog for the tests for which SWI-Prolog can terminate properly.

[5] SWI-Prolog uses SLDNF-resolution, which can have infinite derivations even for Datalog, and for such cases SWI-Prolog terminates with the communication "out of local stack".

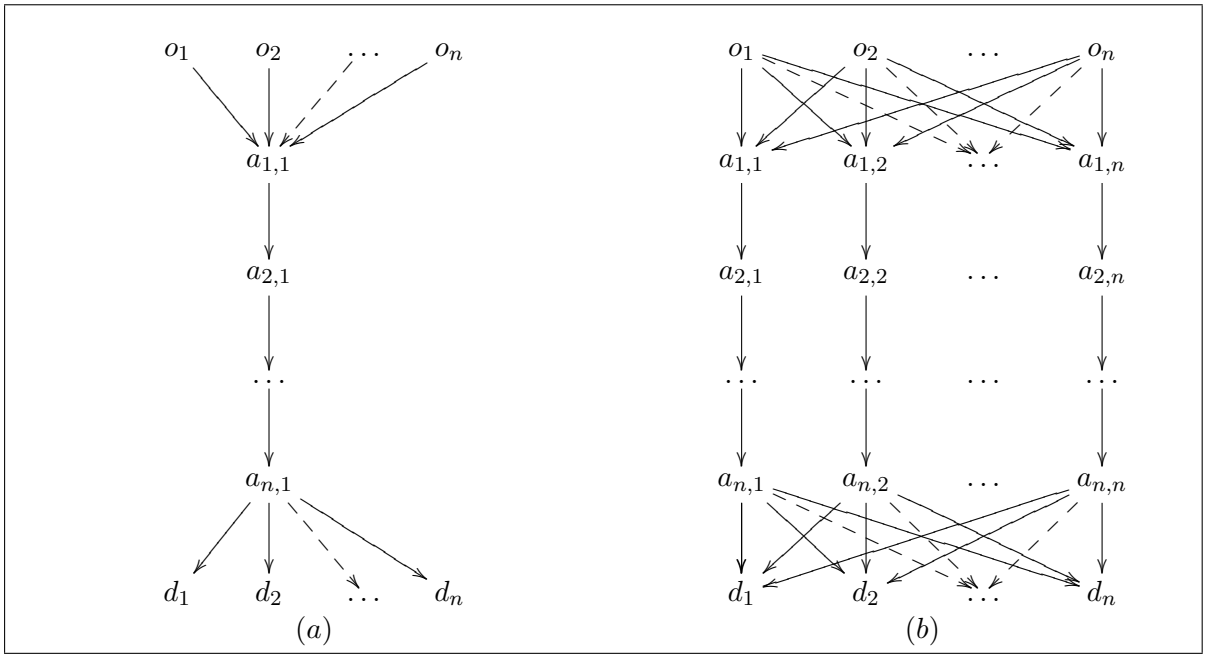


Fig. 2. The extensional instance I_1: (a) $I_1(link_1)$, and (b) $I_1(link_2)$

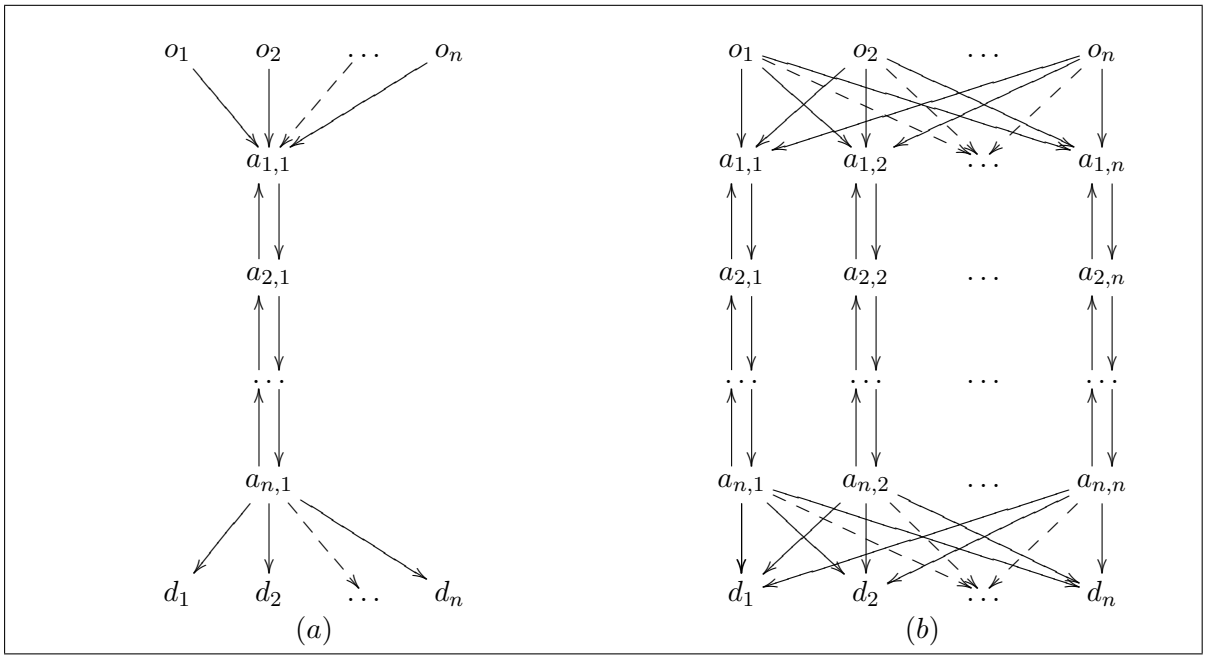


Fig. 3. The extensional instance I_2: (a) $I_2(link_1)$, and (b) $I_2(link_2)$

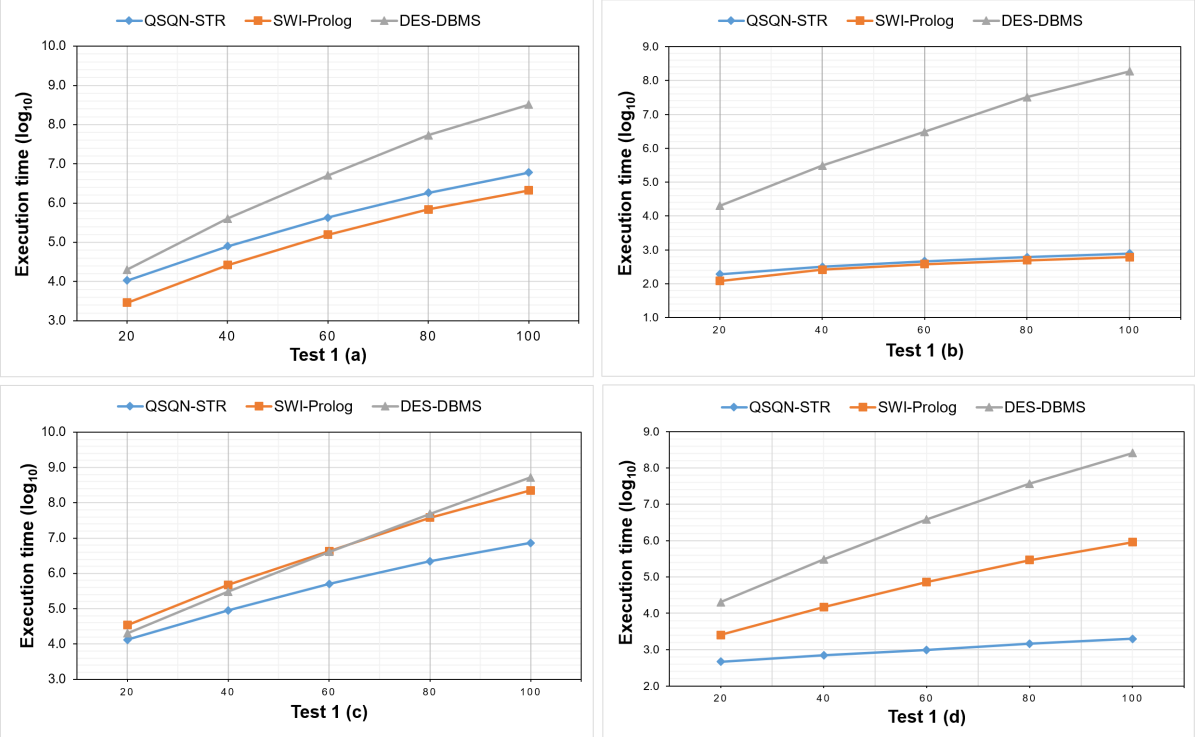


Fig. 4. Experimental results for Test 1

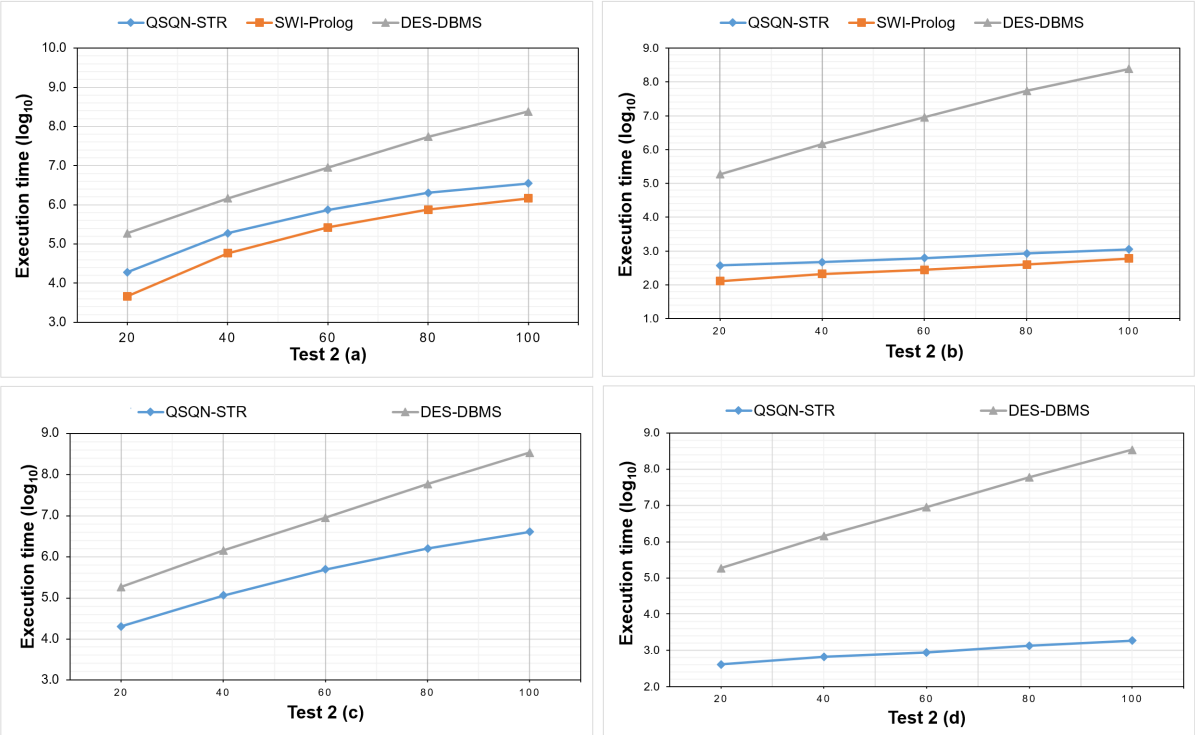


Fig. 5. Experimental results for Test 2

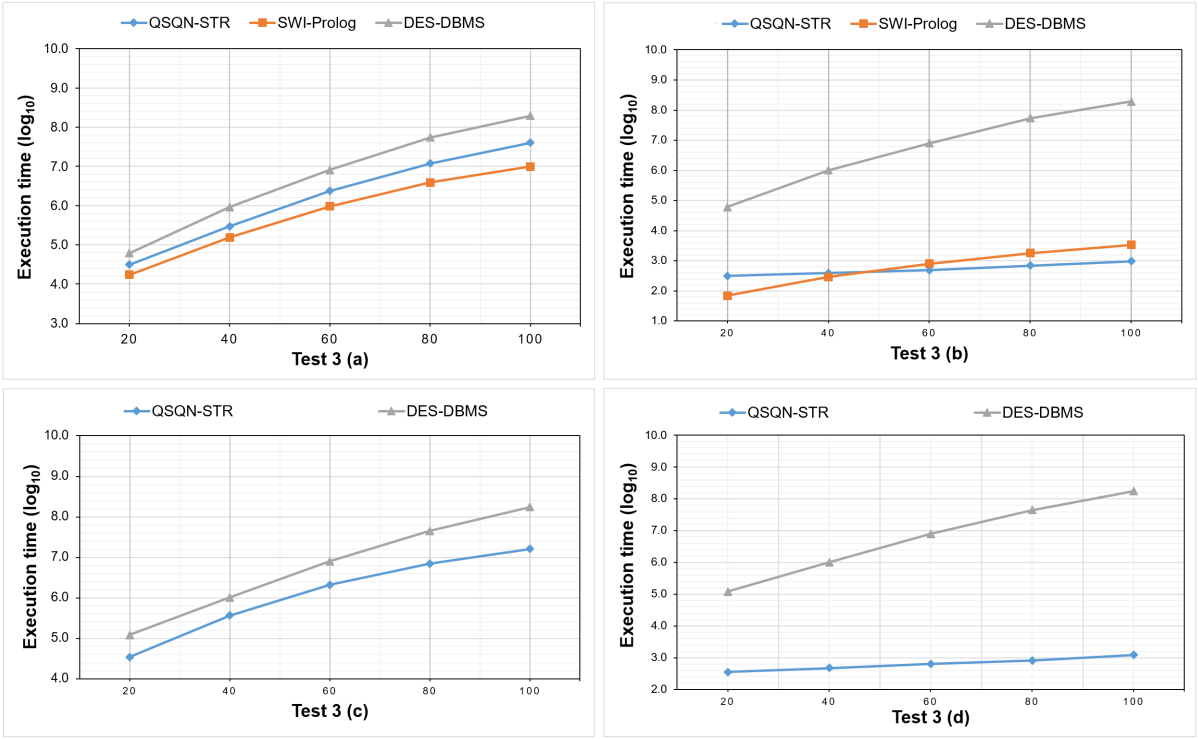


Fig. 6. Experimental results for Test 3

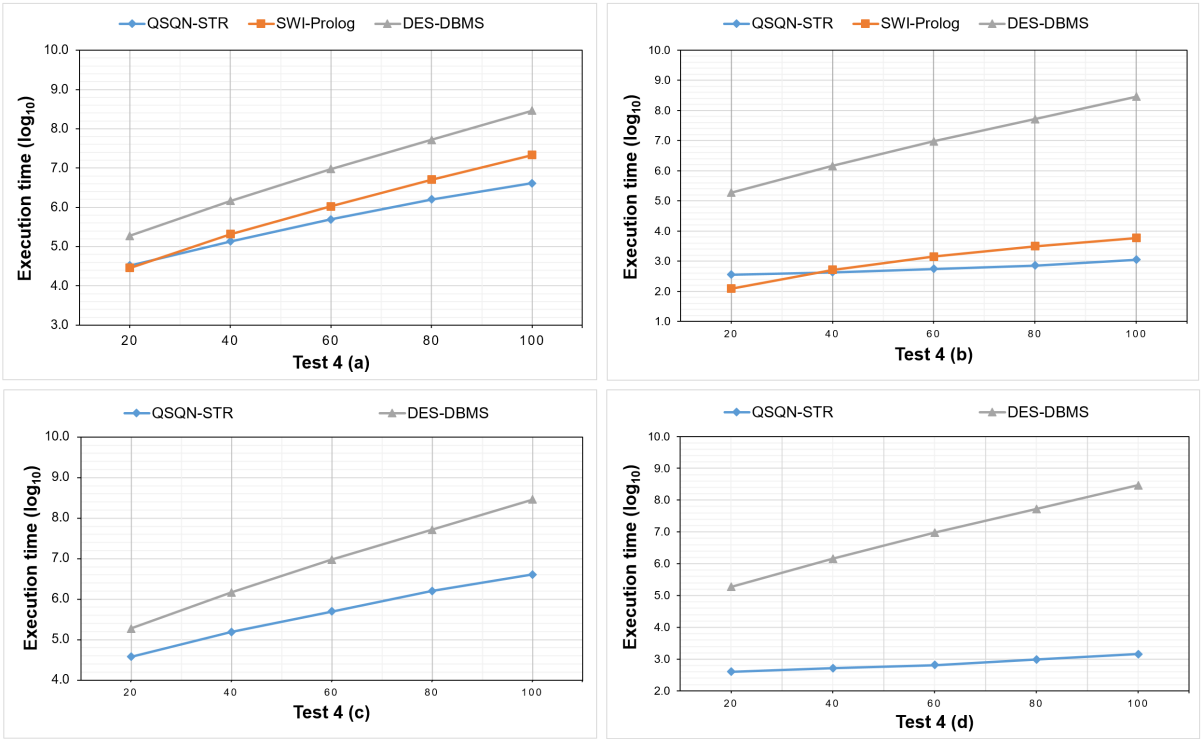


Fig. 7. Experimental results for Test 4

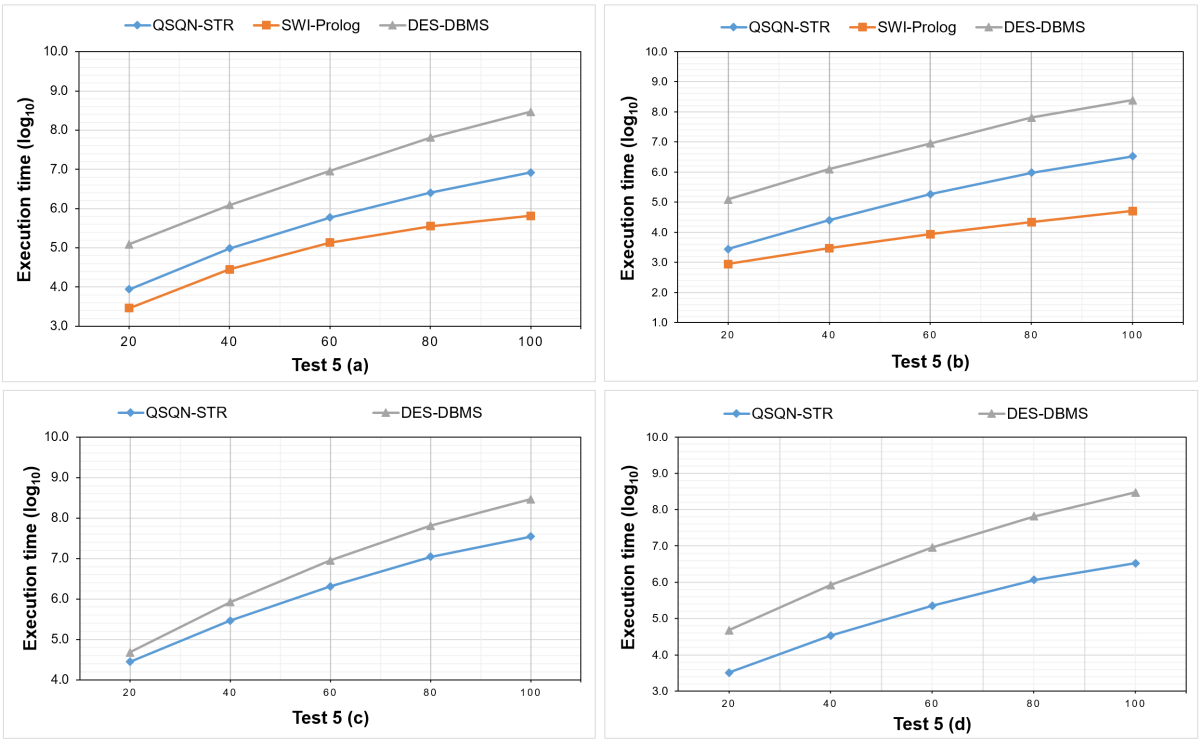


Fig. 8. Experimental results for Test 5

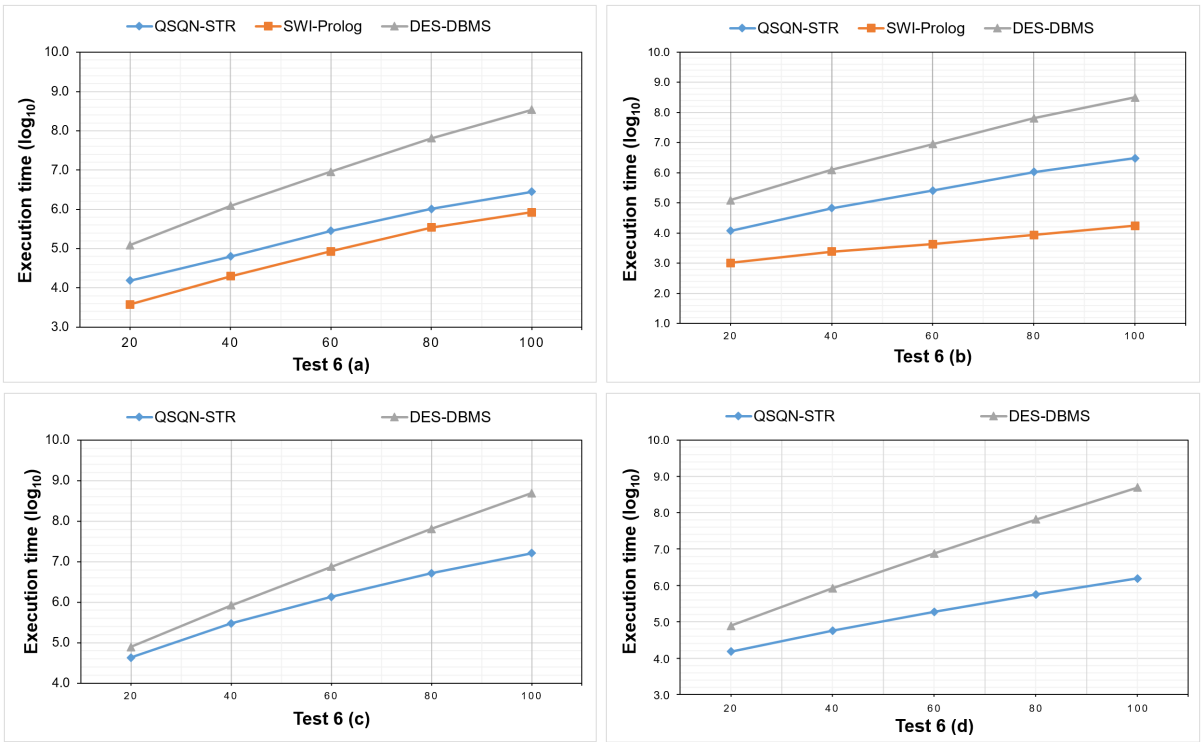


Fig. 9. Experimental results for Test 6

References

1. Cao, S. T.: Methods for Evaluating Queries to Horn Knowledge Bases in First-Order Logic. Ph.D. thesis, University of Warsaw, 2016.
2. Cao, S. T.: An Implementation in Java of the QSQN-STR Evaluation Method. Available at: http://mimuw.edu.pl/~sonct/QSQN-STR18.zip, 2018.
3. Cao, S. T.—Nguyen, L. A.: Incorporating Stratified Negation into Query-Subquery Nets for Evaluating Queries to Stratified Deductive Databases. To appear in Computing and Informatics, 2018, doi: 10.1007/978-3-319-17996-4_32.
4. Cao, S. T.—Nguyen, L. A.: Online Appendix: A Demonstration of the QSQN-STR Method for evaluating Queries to Stratified Datalog$^\neg$. Available at: http://mimuw.edu.pl/~nguyen/QSQN-STR-demonstration.pdf, 2018.
5. Gelfond, M.—Lifschitz, V.: The Stable Model Semantics for Logic Programming. Proceedings of ICLP/SLP 1988, pp. 1070–1080, MIT Press, 1988.
6. Lloyd, J. W.: Foundations of Logic Programming. 2$^{\text{nd}}$ edition, Springer, 1987, doi: 10.1007/978-3-642-83189-8.
7. Sáenz-Pérez, F.: DES: A Deductive Database System. Electronic Notes in Theoretical Computer Science, Vol. 271, 2011, pp. 63–78, doi: 10.1016/j.entcs.2011.02.011.
8. Sagonas, K. F.—Swift, T.—Warren, D. S.: XSB as an Efficient Deductive Database Engine. Proceedings of SIGMOD Conference 1994, pp. 442–453, ACM Press, 1994.
9. Staab, S.: Completeness of the SLD-Resolution. Slides of a Course on Advanced Data Modeling. Available at: https://west.uni-koblenz.de/files/SS08/adm08/db2-ss08-slides9.ppt, 2008.

D Pseudocode of the QSQN-STR Method for Stratified Datalog$^\neg$

Algorithm 1: evaluating a query $q(\overline{x})$ to a stratified Datalog$^\neg$ database (P, I).

Input: a stratified Datalog$^\neg$ database (P, I), a stratification $P = P_1 \cup \ldots \cup P_K$ of P and a query $q(\overline{x})$.

Output: the set of all correct answers for the query $q(\overline{x})$ on (P, I).

let (V, E, T) be a QSQN-STR structure of P;
// T can be chosen arbitrarily or appropriately
set C so that (V, E, T, C) is an empty QSQ-STR-net of P;
let $\overline{x}'$ be a fresh variant of $\overline{x}$;
$\mathit{tuples}(\mathit{input\_q}) := \{\overline{x}'\}$;
foreach $(\mathit{input\_q}, v) \in E$ **do** $\mathit{unprocessed}(\mathit{input\_q}, v) := \{\overline{x}'\}$;
while *there exists* $(u, v) \in E$ *such that* `active-edge`(u, v) *holds* **do**
    select any edge $(u, v) \in E$ such that `active-edge`(u, v) holds and the selection satisfies the admissibility w.r.t. strata's stability;
    `fire`$'(u, v)$;
return $\mathit{tuples}(\mathit{ans\_q})$;

Function active-edge(u, v)

Global data: a QSQ-STR-net (V, E, T, C) of P.
Input: an edge $(u, v) \in E$.
Output: *true* if there are data to transfer through the edge (u, v), and $false$ otherwise.

if u *is pre_filter*$_i$ *or post_filter*$_i$ **then return** $false$;
else if u *is input_p or ans_p* **then return** *unprocessed*$(u, v) \neq \emptyset$;
else if u *is filter*$_{i,j}$ *and kind*$(u) =$ *extensional* **then**
    return $T(u) = true \wedge$ *unprocessed_subqueries*$(u) \neq \emptyset$;
else // u `is of the form` *filter*$_{i,j}$ `and` *kind*$(u) =$ *intensional*
    let $p =$ *pred*(u);
    if $v =$ *input_p* **then return** *unprocessed_subqueries*$_2(u) \neq \emptyset$;
    else return *unprocessed_subqueries*$(u) \neq \emptyset \vee$ *unprocessed_tuples*$(u) \neq \emptyset$;

Procedure add-subquery$(\bar{t}, \delta, \Gamma, v)$

Purpose: add the subquery $(\bar{t}, \delta)$ to Γ, but keep in Γ only the most general subqueries w.r.t. v.

if *no subquery in* Γ *is more general than* $(\bar{t}, \delta)$ *w.r.t.* v **then**
    delete from Γ all subqueries less general than $(\bar{t}, \delta)$ w.r.t. v;
    add $(\bar{t}, \delta)$ to Γ;

Procedure add-tuple$(\bar{t}, \Gamma)$

Purpose: add a fresh variant of the tuple $\bar{t}$ to Γ, but keep in Γ only the most general tuples.

let $\bar{t}'$ be a fresh variant of $\bar{t}$;
if $\bar{t}'$ *is not an instance of any tuple from* Γ **then**
    delete from Γ all tuples that are instances of $\bar{t}'$;
    add $\bar{t}'$ to Γ;

Procedure transfer$'(D, u, v)$

Global data: a stratified Datalog$^\neg$ database (P, I) and a QSQ-STR-net (V, E, T, C) of P.
Input: data D to transfer through the edge $(u, v) \in E$.

if $D = \emptyset$ **then return**;
else if v *is post_filter*$_i$ **then** `transfer`$'(\{\bar{t} \mid (\bar{t}, \varepsilon) \in D\}, v,$ *succ*$(v))$;
else if u *is ans_p* **then** *unprocessed_tuples*$(v) :=$ *unprocessed_tuples*$(v) \cup D$;
else if v *is ans_p* **then** `transfer`$_1(D, u, v)$;
else if v *is input_p* **then** `transfer`$_2(D, u, v)$;
else if u *is input_p* **then** `transfer`$'_3(D, u, v)$;
else if v *is filter*$_{i,j}$, *kind*$(v) =$ *extensional and* $T(v) =$ *false* **then**
    if *neg*$(v) =$ *false* **then** `transfer`$'_4(D, u, v)$;
    else `transfer`$'_{4b}(D, u, v)$;
else `transfer`$_5(D, u, v)$;

Procedure transfer$_1(D, u, v)$

// $v = ans\_p$, $u = post\_filter_i$, D is a set of tuples of constants

foreach $\bar{t} \in D - tuples(v)$ **do**
  add $\bar{t}$ to $tuples(v)$;
  foreach $(v, w) \in E$ **do** add $\bar{t}$ to $unprocessed(v, w)$;

Procedure transfer$_2(D, u, v)$

// $v = input\_p$, $u = filter_{i,j}$ and $kind(u) = intensional$

foreach $\bar{t} \in D$ **do**
  let $\bar{t}'$ be a fresh variant of $\bar{t}$;
  if $\bar{t}'$ *is not an instance of any tuple from* $tuples(v)$ **then**
    delete from $tuples(v)$ all tuples that are instances of $\bar{t}'$;
    add $\bar{t}'$ to $tuples(v)$;
    foreach $(v, w) \in E$ **do**
      delete from $unprocessed(v, w)$ all tuples that are instances of $\bar{t}'$;
      add $\bar{t}'$ to $unprocessed(v, w)$;

Procedure transfer$'_3(D, u, v)$

// u is $input\_p$ and $v = pre\_filter_i$

$\Gamma := \emptyset$;
foreach $\bar{t} \in D$ **do**
  if $p(\bar{t})$ *and* $atom(v)$ *are unifiable by an mgu* γ **then**
    add-subquery$(\bar{t}\gamma, \gamma_{|post_vars(v)}, \Gamma, succ(v))$;
transfer$'(\Gamma, v, succ(v))$;

Procedure transfer$'_4(D, u, v)$

// $v = filter_{i,j}$, $kind(v) = extensional$, $T(v) =$ *false* and $neg(v) =$ *false*

let $p = pred(v)$ and set $\Gamma := \emptyset$;
foreach $(\bar{t}, \delta) \in D$ *and* $\bar{t}' \in I(p)$ **do**
  if $atom(v)\delta$ *and* $p(\bar{t}')$ *are unifiable by an mgu* γ **then**
    add-subquery$(\bar{t}\gamma, (\delta\gamma)_{|post_vars(v)}, \Gamma, succ(v))$;
transfer$'(\Gamma, v, succ(v))$;

Procedure transfer$'_{4b}(D, u, v)$

// $v = filter_{i,j}$, $kind(v) = extensional$, $T(v) =$ *false*, $neg(v) =$ *true*

let $p = pred(v)$ and set $\Gamma := \emptyset$;
foreach $(\bar{t}, \delta) \in D$ **do**
  if $atom(v)\delta \notin \{p(\bar{t}') \mid \bar{t}' \in I(p)\}$ **then**
    add-subquery$(\bar{t}, \delta_{|post_vars(u)}, \Gamma, succ(v))$;
transfer$'(\Gamma, v, succ(v))$;

Procedure transfer$_5(D, u, v)$

/* $v = \mathit{filter}_{i,j}$, ($\mathit{kind}(v) = \mathit{extensional}$ and $T(v) =$ `true` or $\mathit{kind}(v) = \mathit{intensional}$) and u is not of the form $\mathit{ans\_p}$ */

1 **foreach** $(\bar{t}, \delta) \in D$ **do**
2 **if** *no subquery in* $\mathit{subqueries}(v)$ *is more general than* $(\bar{t}, \delta)$ *w.r.t.* v **then**
3 delete from $\mathit{subqueries}(v)$ and $\mathit{unprocessed\_subqueries}(v)$ all subqueries less general than $(\bar{t}, \delta)$ w.r.t. v;
4 add $(\bar{t}, \delta)$ to both $\mathit{subqueries}(v)$ and $\mathit{unprocessed\_subqueries}(v)$;
5 **if** $\mathit{kind}(v) = \mathit{intensional}$ **then**
6 delete from $\mathit{unprocessed\_subqueries}_2(v)$ all subqueries less general than $(\bar{t}, \delta)$ w.r.t. v;
7 add $(\bar{t}, \delta)$ to $\mathit{unprocessed\_subqueries}_2(v)$;

Procedure fire$'(u, v)$

Global data: a stratified Datalog$^\neg$ database (P, I) and a QSQ-STR-net (V, E, T, C) of P.

Input: an edge $(u, v) \in E$ such that `active-edge`(u, v) holds.

1 **if** u *is* $\mathit{ans\_p}$ **then**
2 `transfer'`$(\mathit{unprocessed}(u, v), u, v)$;
3 $\mathit{unprocessed}(u, v) := \emptyset$;
4 **else if** u *is* $\mathit{input\_p}$ **then**
5 `transfer'`$(\mathit{unprocessed}(u, v) - \mathit{tuples}(\mathit{ans\_p}), u, v)$;
6 $\mathit{unprocessed}(u, v) := \emptyset$;
7 **else if** v *is* $\mathit{input\_p}$ **then** `fire'`$_1(u, v)$;
8 **else if** u *is* $\mathit{filter}_{i,j}$ *and* $\mathit{neg}(u) = \mathit{false}$ **then**
9 **if** $\mathit{kind}(u) = \mathit{extensional}$ **then** `fire'`$_2(u, v)$;
10 **else** `fire'`$_3(u, v)$;
11 **else if** u *is* $\mathit{filter}_{i,j}$ *and* $\mathit{neg}(u) = \mathit{true}$ **then** `fire'`$_4(u, v)$;

Procedure fire$'_1(u, v)$

// $v = \mathit{input\_p}$, $u = \mathit{filter}_{i,j}$ and $\mathit{kind}(u) = \mathit{intensional}$

1 let $p = \mathit{pred}(u)$ and set $\Gamma := \emptyset$;
2 **foreach** $(\bar{t}, \delta) \in \mathit{unprocessed\_subqueries}_2(u)$ **do**
3 let $p(\bar{t}') = \mathit{atom}(u)\delta$;
4 `add-tuple`$(\bar{t}', \Gamma)$;
5 $\mathit{unprocessed\_subqueries}_2(u) := \emptyset$;
6 `transfer'`(Γ, u, v);

Procedure $\text{fire}'_2(u, v)$

// $u = \mathit{filter}_{i,j}$, $\mathit{neg}(u) =$ `false`, $\mathit{kind}(u) = \mathit{extensional}$ `and` $T(u) =$ `true`

let $p = \mathit{pred}(u)$ and set $\Gamma := \emptyset$;
foreach $(\bar{t}, \delta) \in \mathit{unprocessed\_subqueries}(u)$ *and* $\bar{t}' \in I(p)$ **do**
  if $\mathit{atom}(u)\delta$ *and* $p(\bar{t}')$ *are unifiable by an mgu* γ **then**
    `add-subquery`$(\bar{t}\gamma, (\delta\gamma)_{|\mathit{post\_vars}(u)}, \Gamma, v)$;
$\mathit{unprocessed\_subqueries}(u) := \emptyset$;
`transfer`$'(\Gamma, u, v)$;

Procedure $\text{fire}'_3(u, v)$

// $u = \mathit{filter}_{i,j}$, $\mathit{neg}(u) =$ `false`, $\mathit{kind} = \mathit{intensional}$ `and` $v = \mathit{succ}(u)$

let $p = \mathit{pred}(u)$ and set $\Gamma := \emptyset$;
foreach $(\bar{t}, \delta) \in \mathit{unprocessed\_subqueries}(u)$ *and* $\bar{t}' \in \mathit{tuples}(\mathit{ans\_}p)$ **do**
  if $\mathit{atom}(u)\delta$ *and* $p(\bar{t}')$ *are unifiable by an mgu* γ **then**
    `add-subquery`$(\bar{t}\gamma, (\delta\gamma)_{|\mathit{post\_vars}(u)}, \Gamma, v)$;
$\mathit{unprocessed\_subqueries}(u) := \emptyset$;
foreach $(\bar{t}, \delta) \in \mathit{subqueries}(u)$ *and* $\bar{t}' \in \mathit{unprocessed\_tuples}(u)$ **do**
  if $\mathit{atom}(u)\delta$ *and* $p(\bar{t}')$ *are unifiable by an mgu* γ **then**
    `add-subquery`$(\bar{t}\gamma, (\delta\gamma)_{|\mathit{post\_vars}(u)}, \Gamma, v)$;
$\mathit{unprocessed\_tuples}(u) := \emptyset$;
`transfer`$'(\Gamma, u, v)$;

Procedure $\text{fire}'_4(u, v)$

// $u = \mathit{filter}_{i,j}$, $\mathit{neg}(u) =$ `true` `and` $v = \mathit{succ}(u)$

let $p = \mathit{pred}(u)$ and set $\Gamma := \emptyset$;
let R be $I(p)$ if $\mathit{kind}(u) = \mathit{extensional}$, and $\mathit{tuples}(\mathit{ans\_}p)$ otherwise;
foreach $(\bar{t}, \delta) \in \mathit{unprocessed\_subqueries}(u)$ **do**
  if $\mathit{atom}(u)\delta \notin \{p(\bar{t}') \mid \bar{t}' \in R\}$ **then**
    `add-subquery`$(\bar{t}, \delta_{|\mathit{post\_vars}(u)}, \Gamma, v)$;
$\mathit{unprocessed\_subqueries}(u) := \emptyset$;
`transfer`$'(\Gamma, u, v)$;

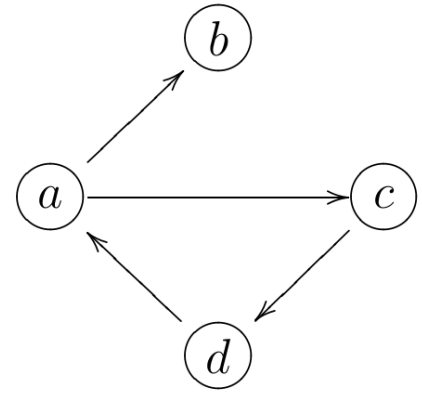
Online Appendix

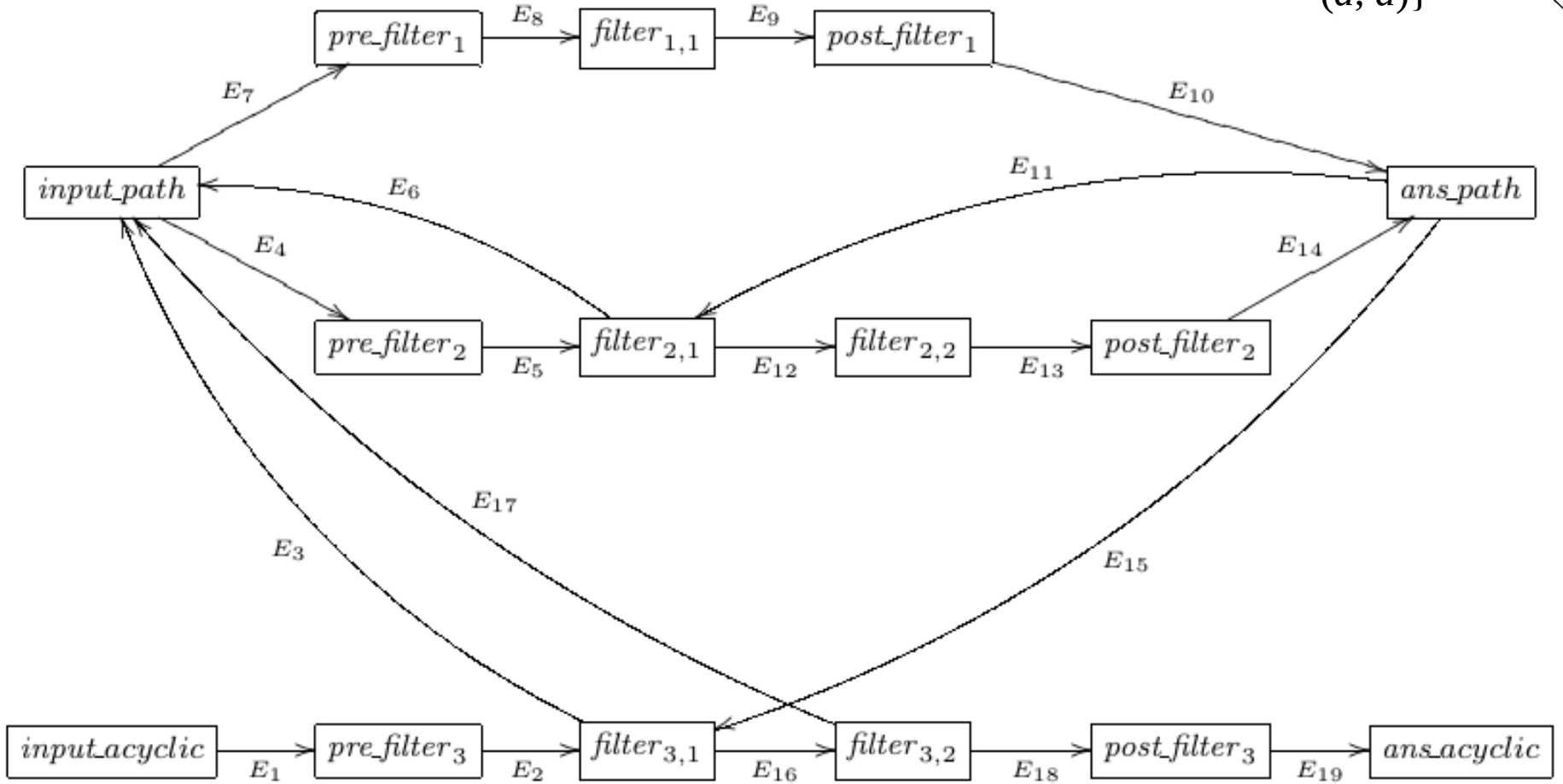
A Demonstration of the QSQN-STR Method for Evaluating Queries to Stratified Datalog¬

(this should be displayed using the presentation mode)

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$

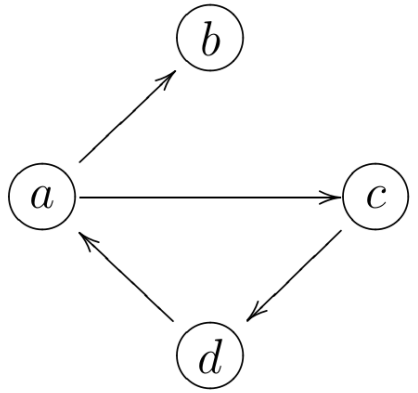
edge
{(*a*, *b*),
(*a*, *c*),
(*c*, *d*),
(*d*, *a*)}

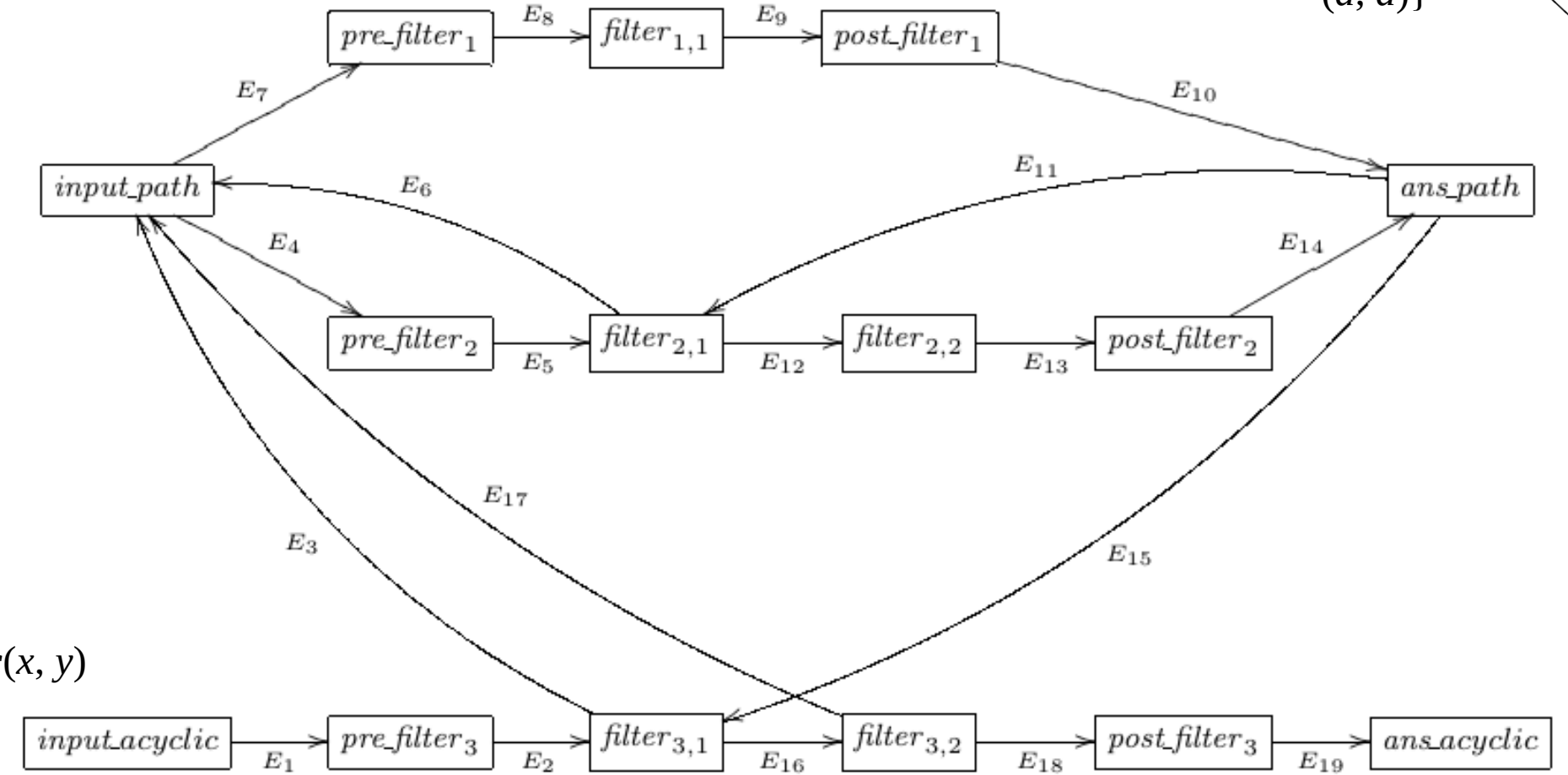




$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$

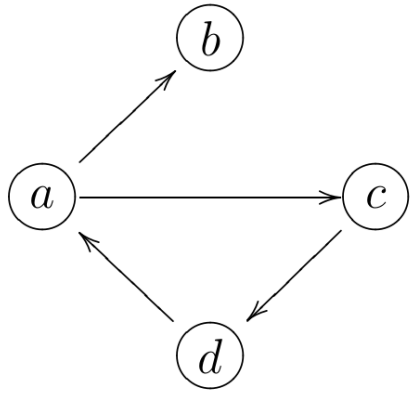
edge
{(*a*, *b*),
(*a*, *c*),
(*c*, *d*),
(*d*, *a*)}

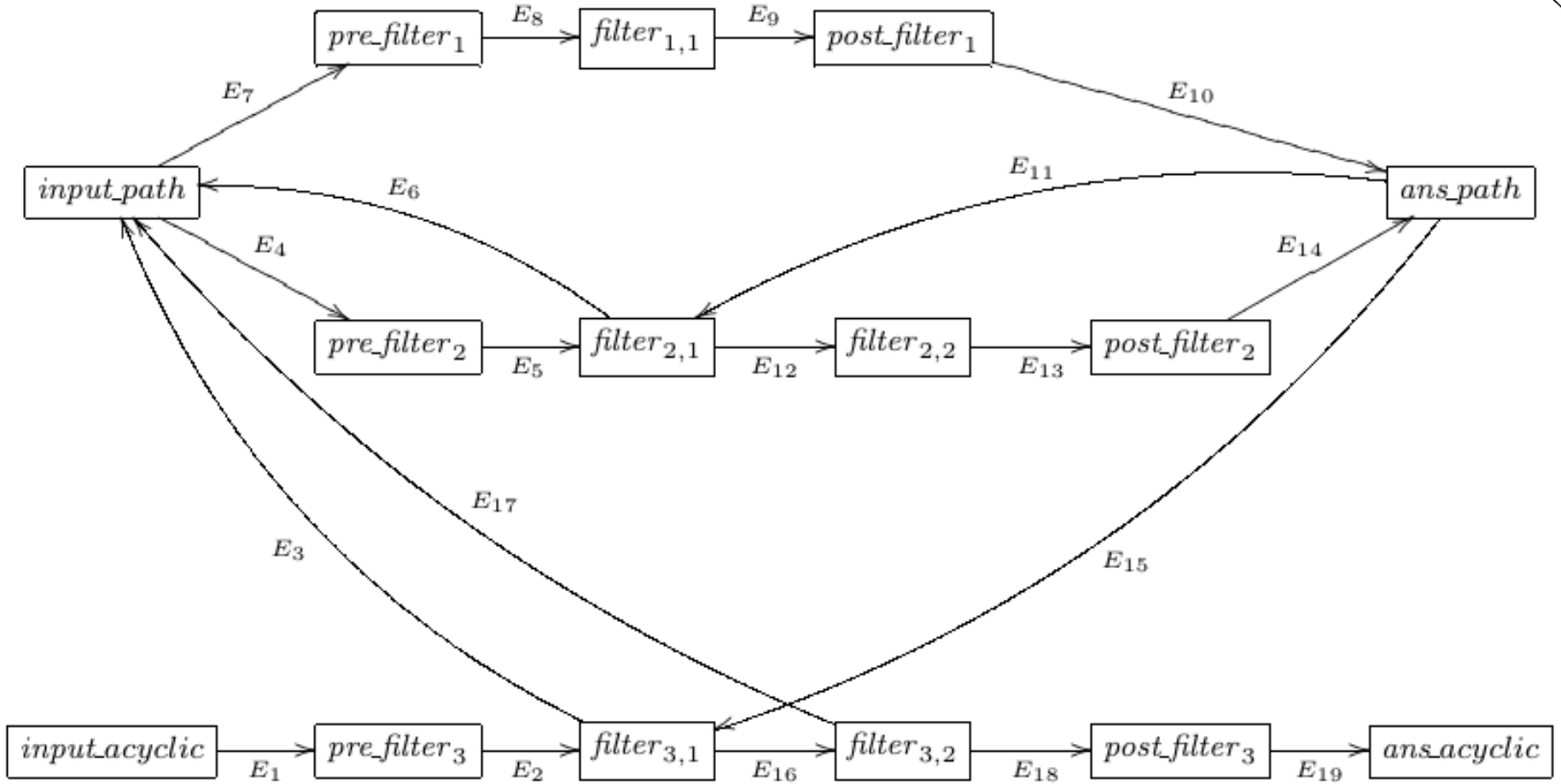




Query: *acyclic*(*x*, *y*)

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$

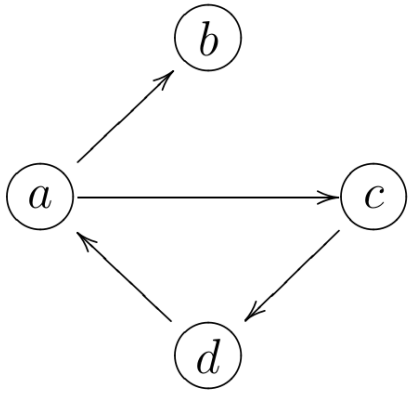


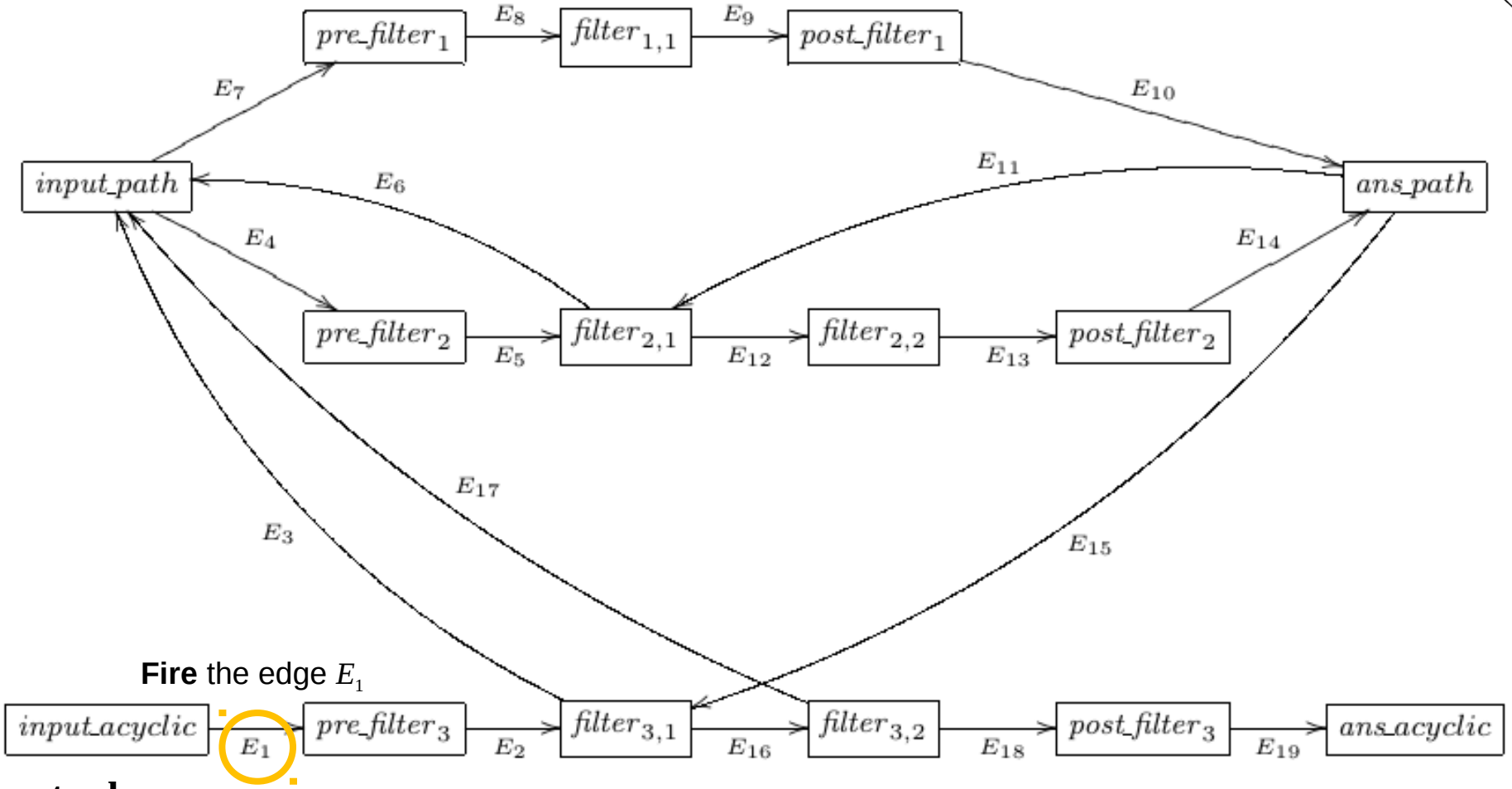


tuples

(x_1, y_1)

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$



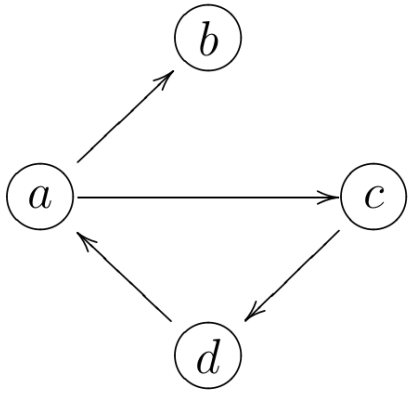


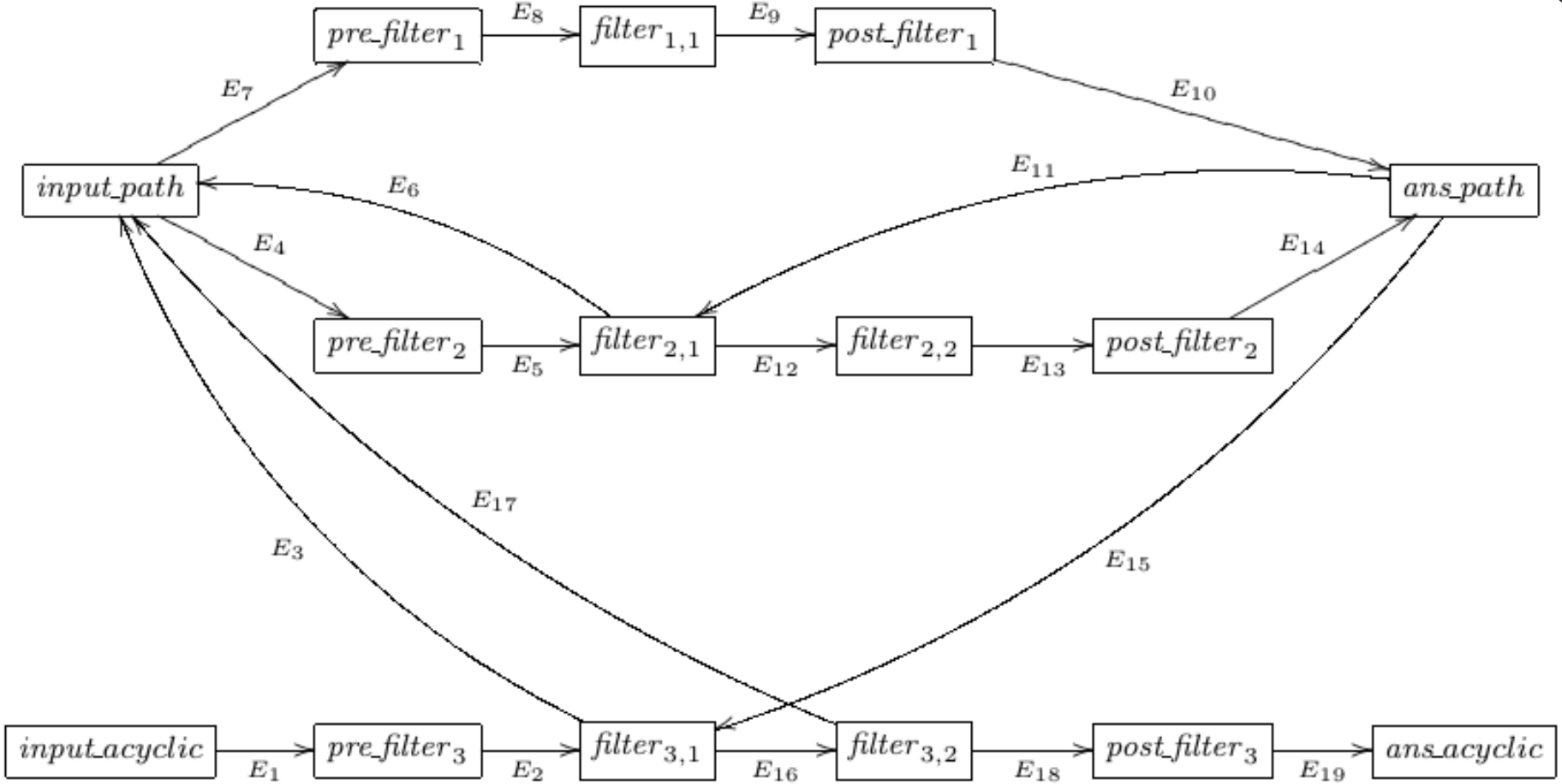
Fire the edge E_1

tuples

(x_1, y_1)

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$

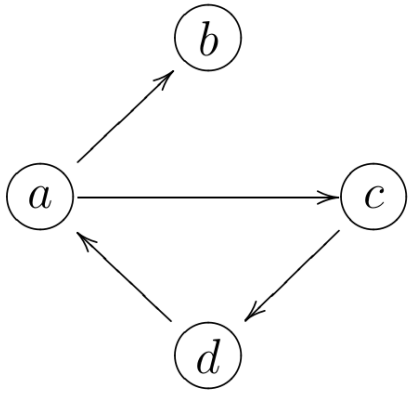


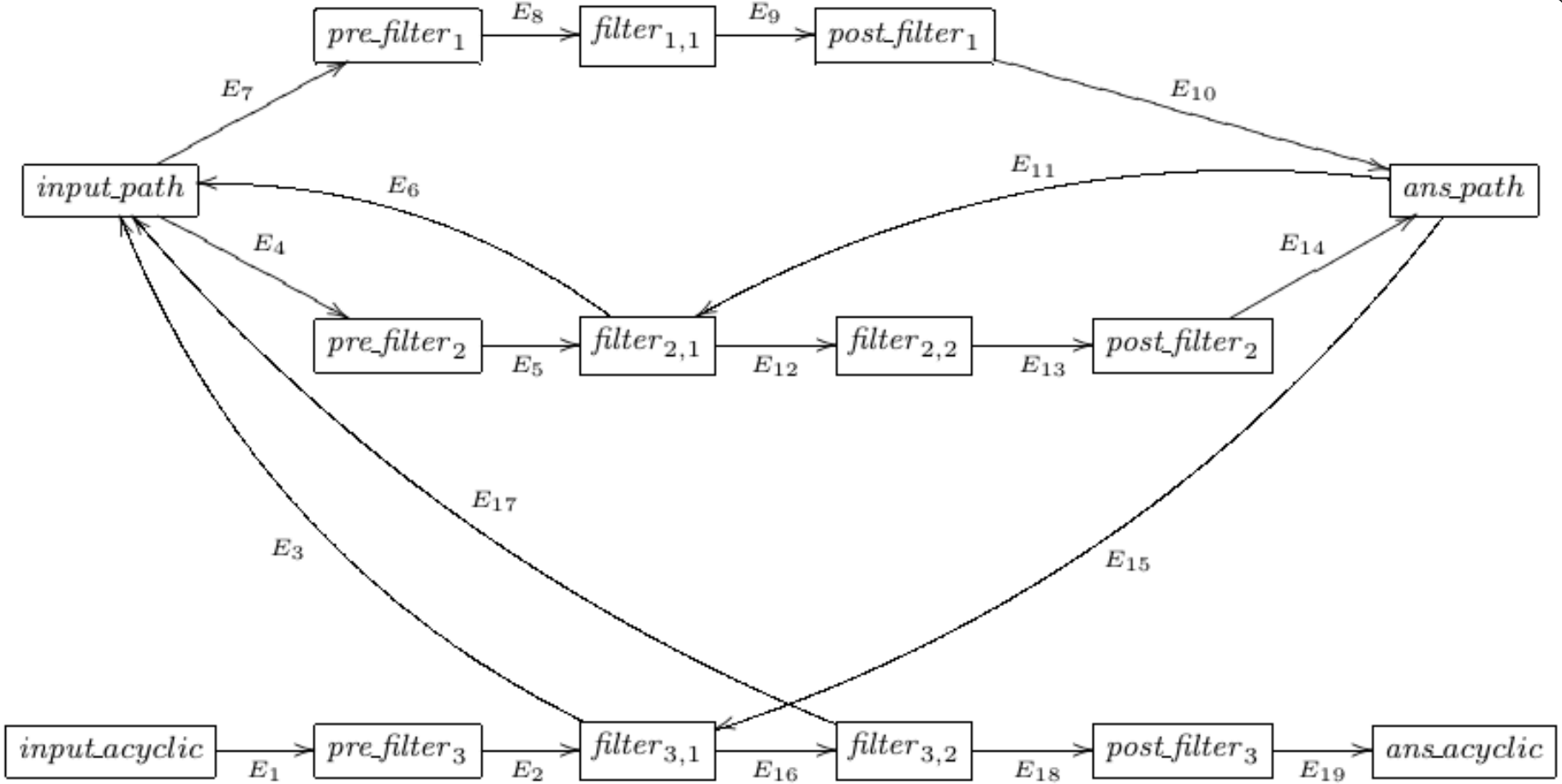


tuples

(x_1, y_1) $\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





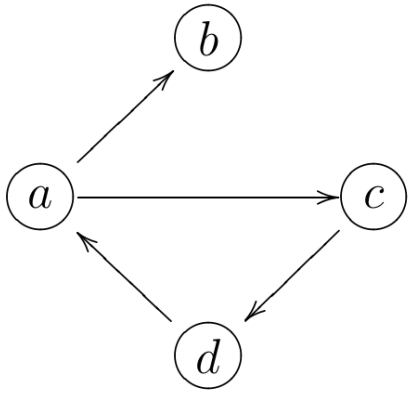
tuples

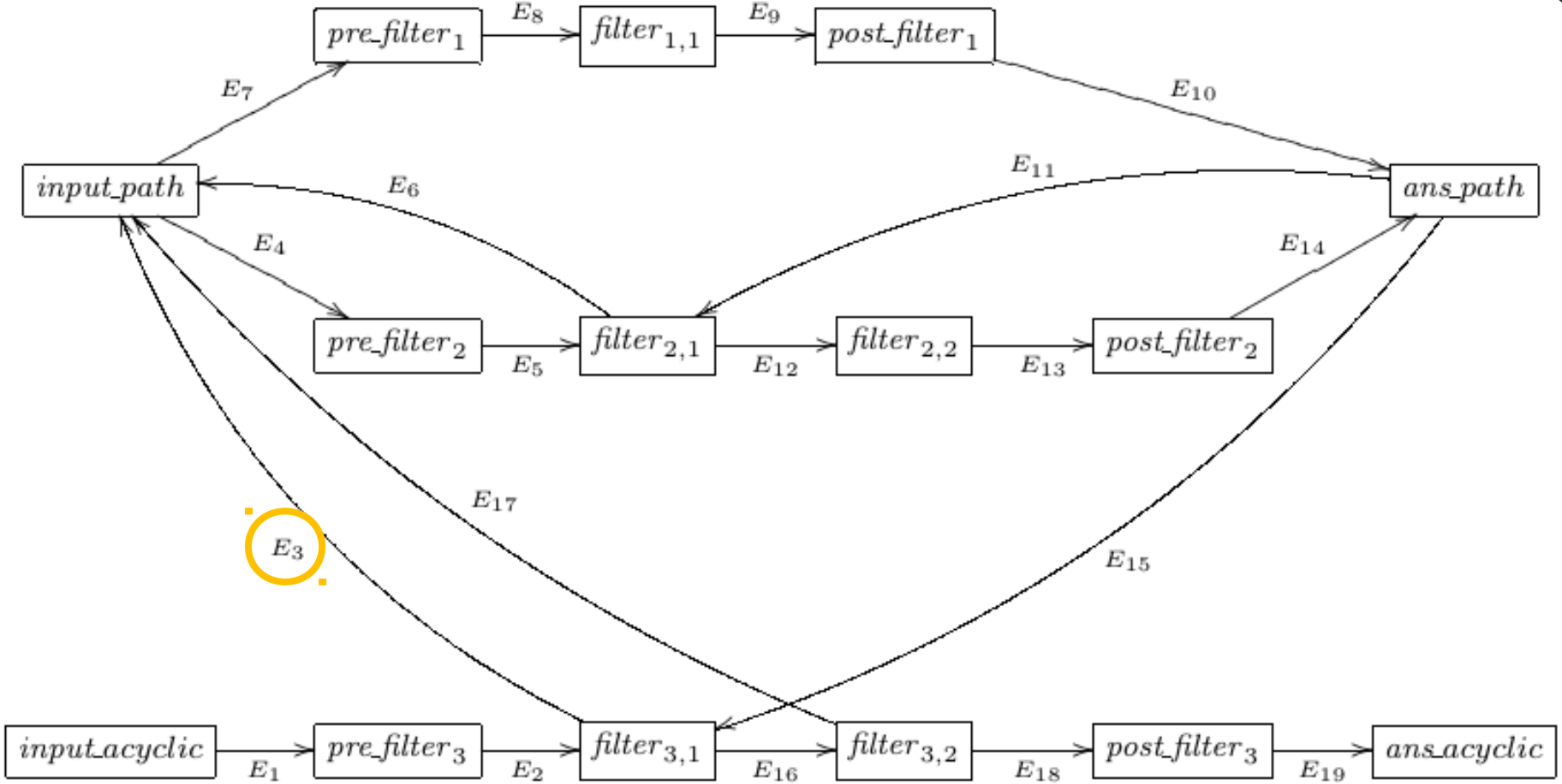
(x_1, y_1)

subqueries

$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





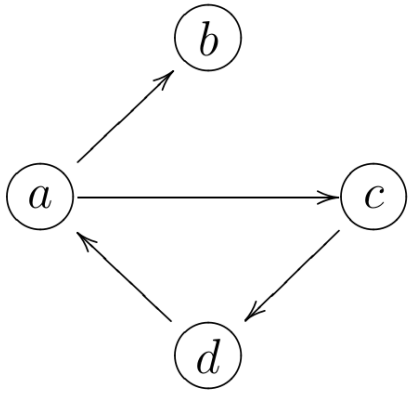
tuples

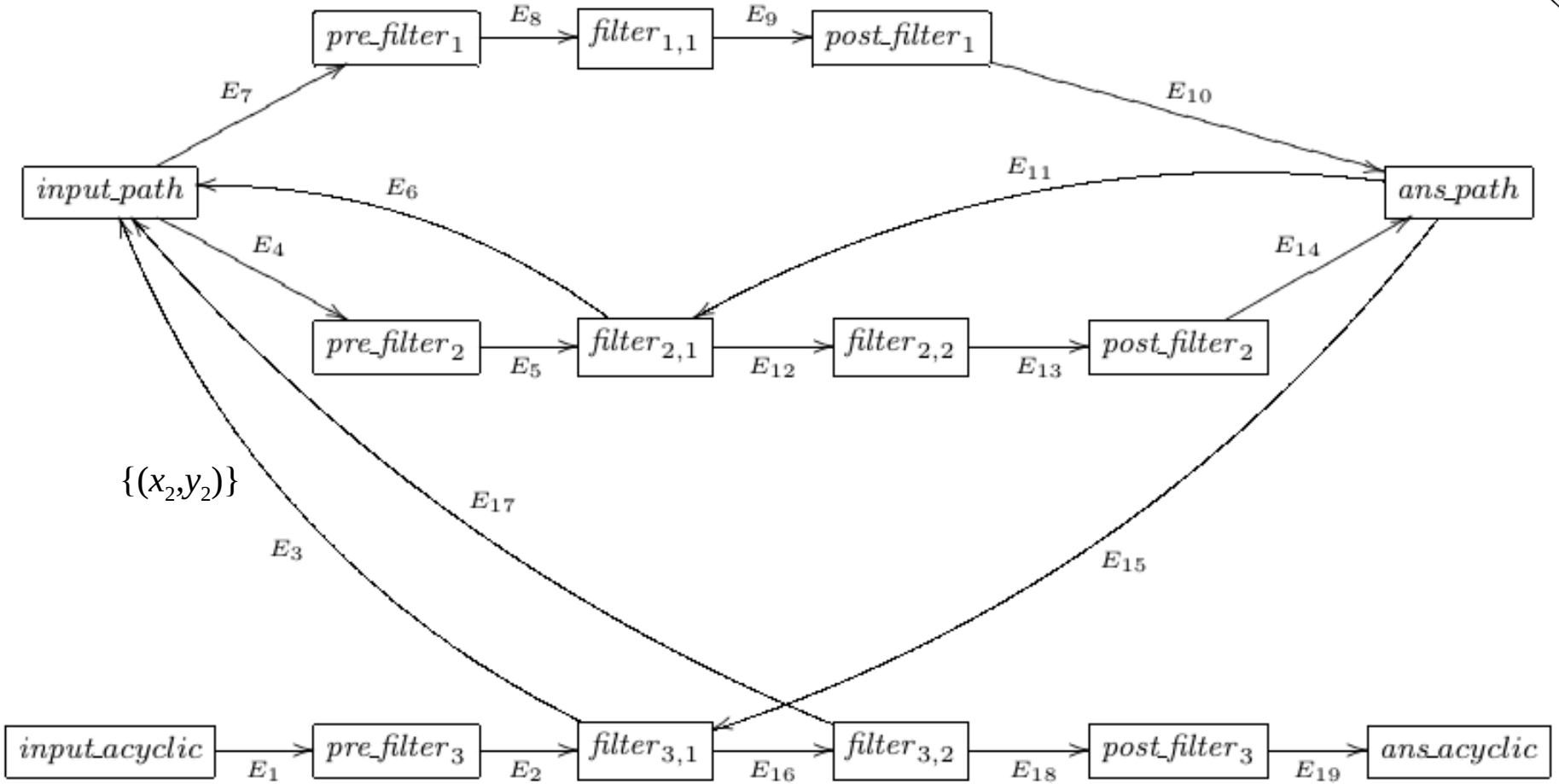
(x_1, y_1)

subqueries

$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





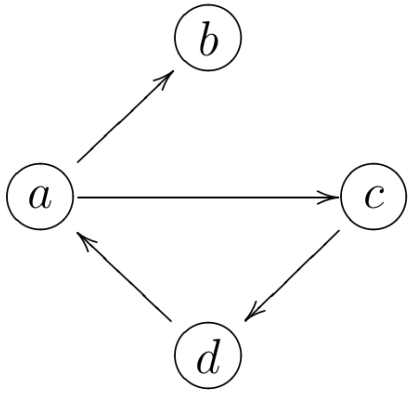
tuples

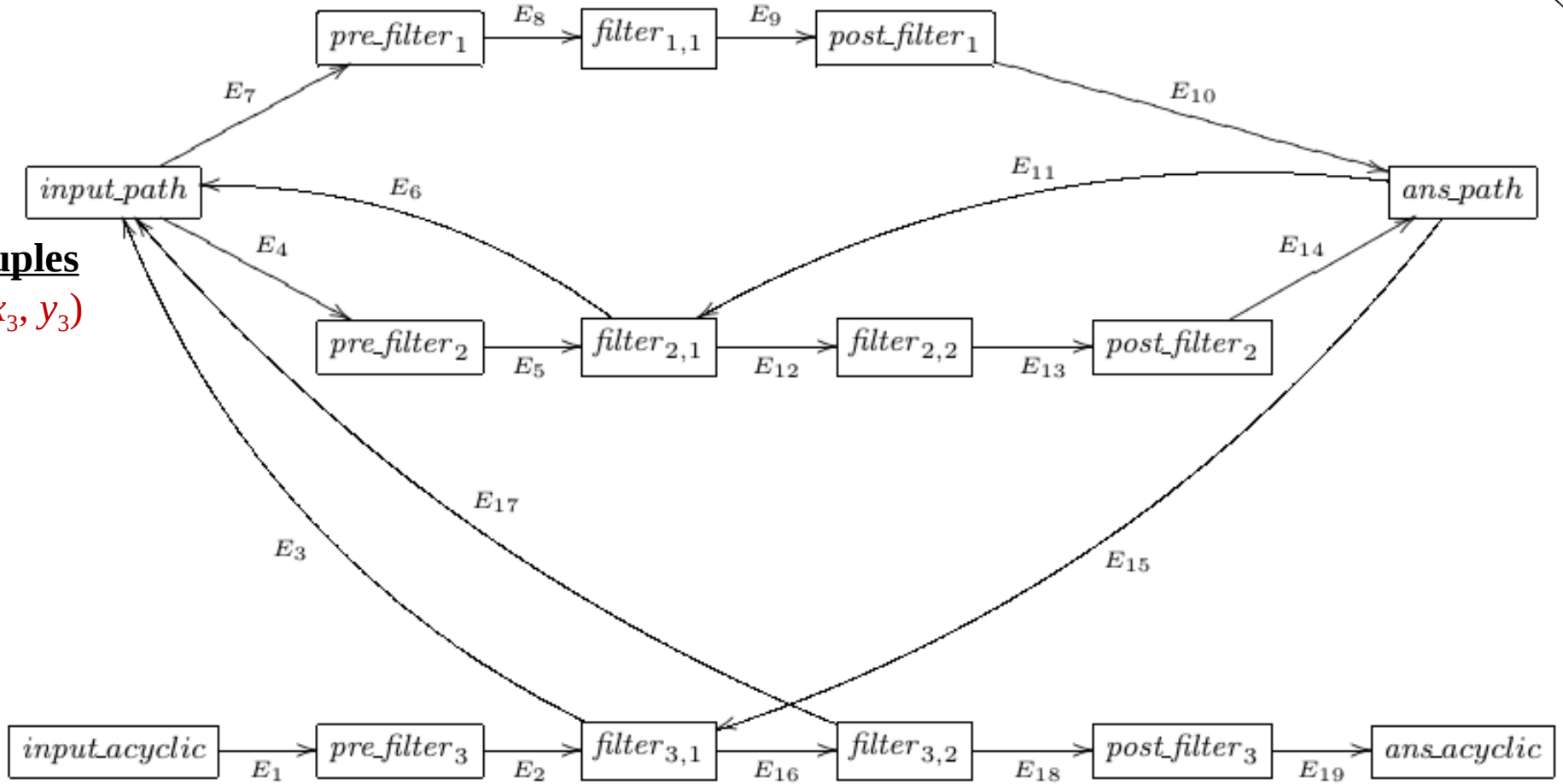
(x_1, y_1)

subqueries

$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$



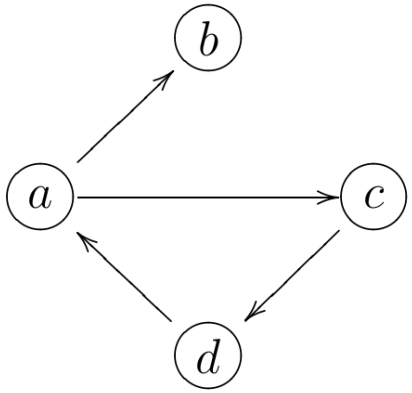


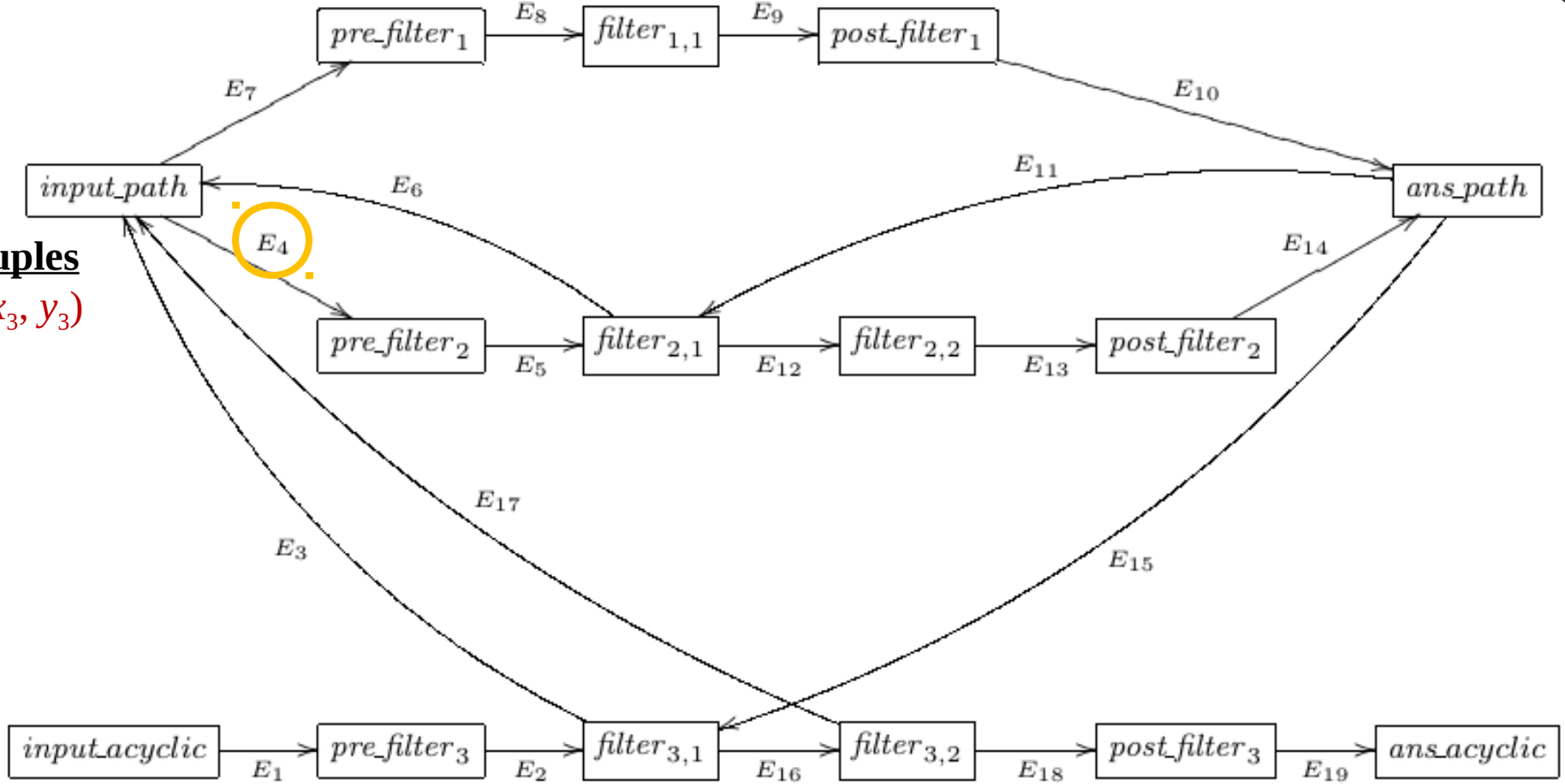
tuples
(x_3, y_3)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$



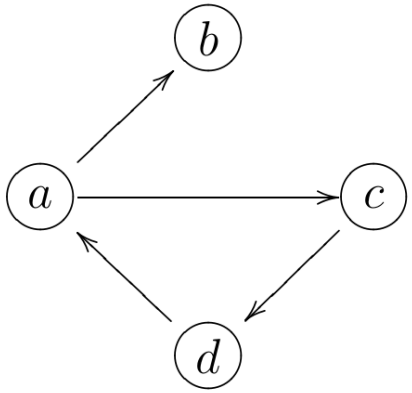


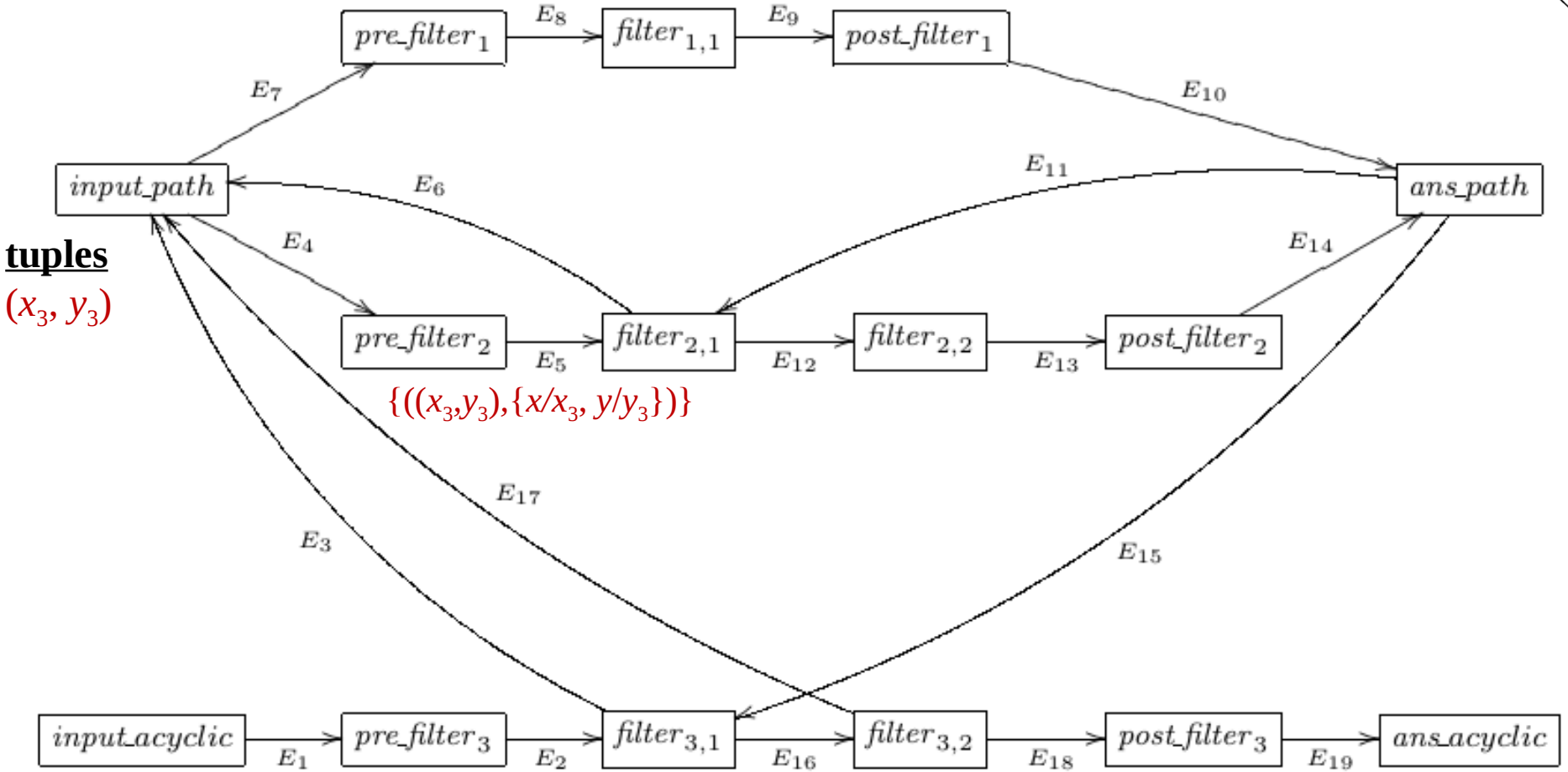
tuples
(x_3, y_3)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





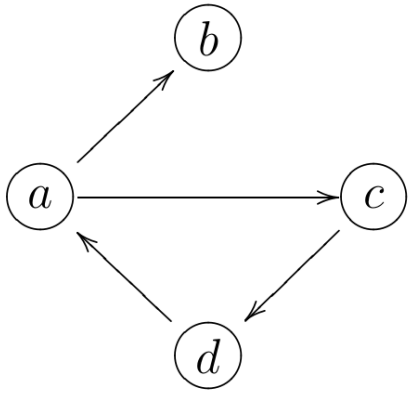
tuples
(x_3, y_3)

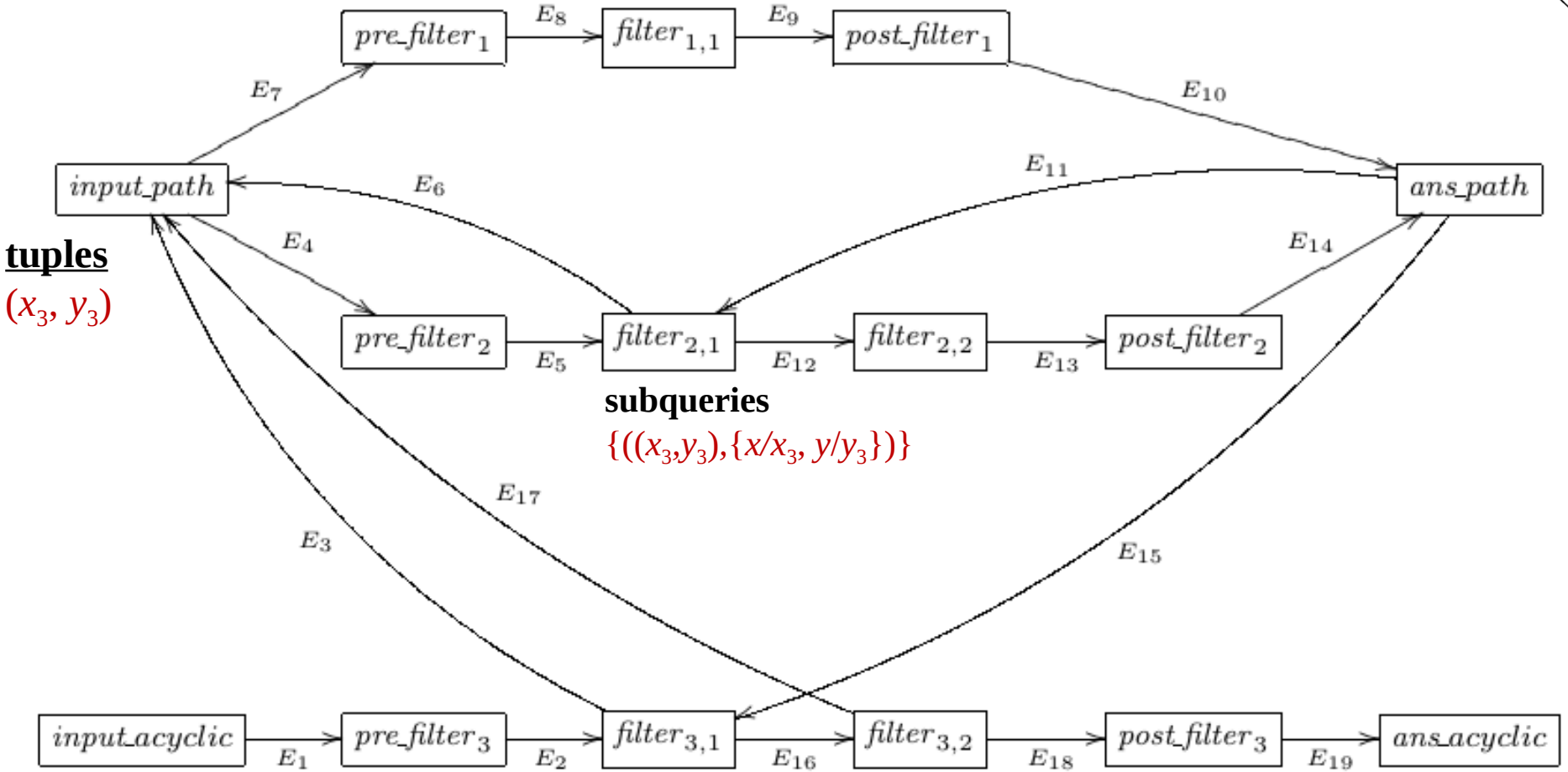
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





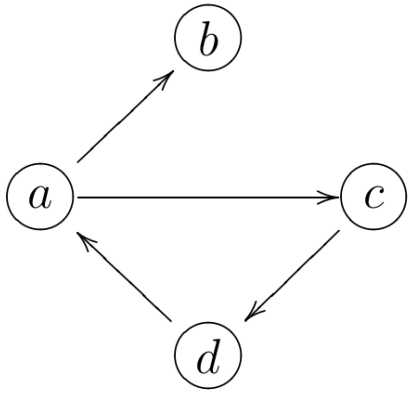
tuples
(x_3, y_3)

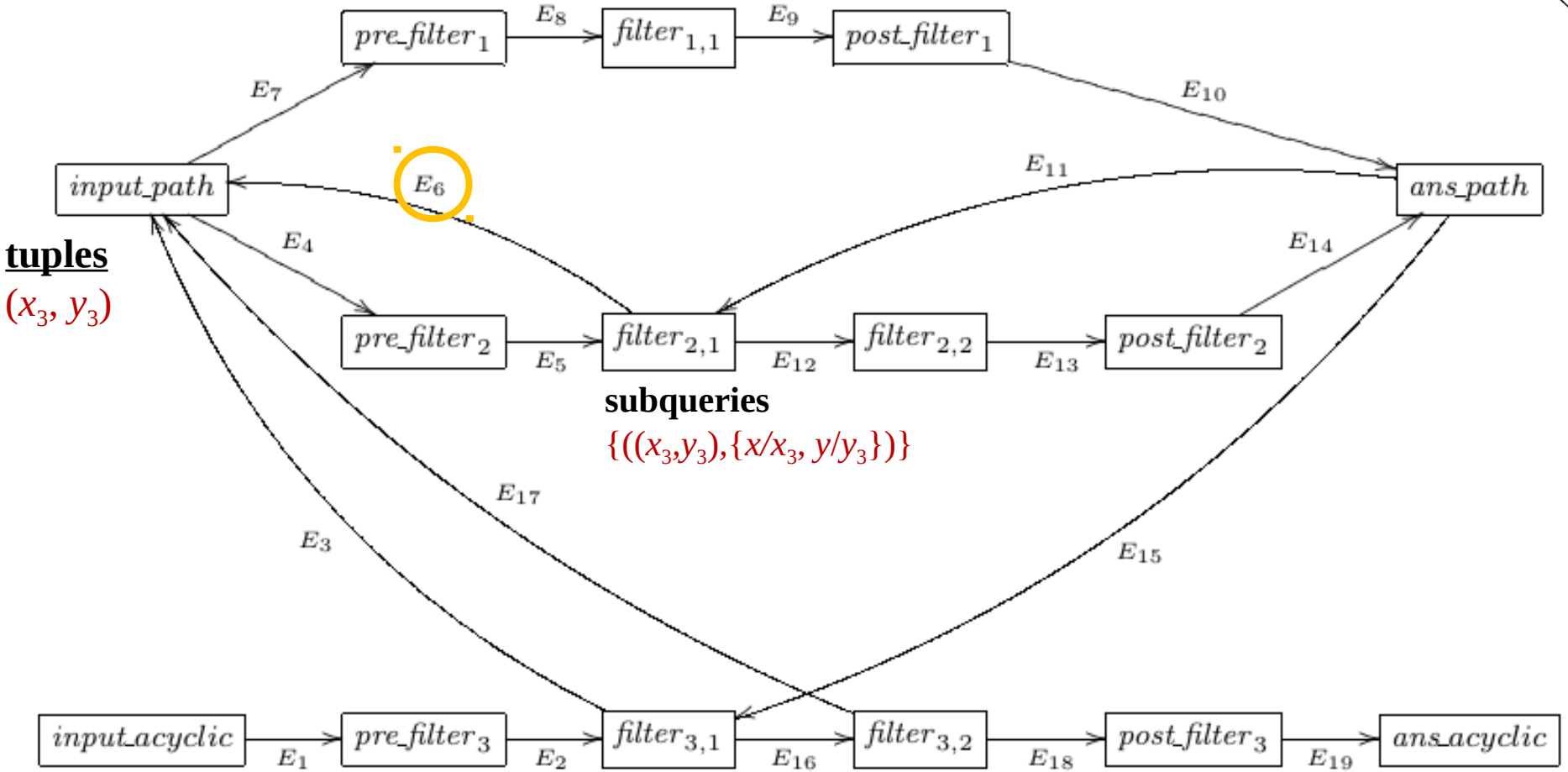
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





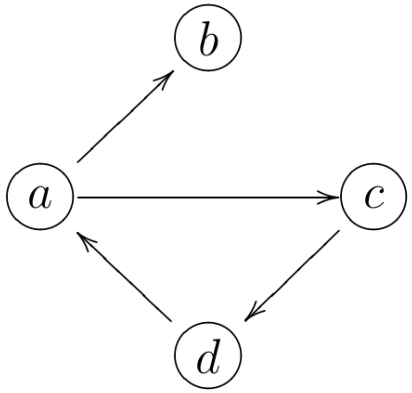
tuples
(x_3, y_3)

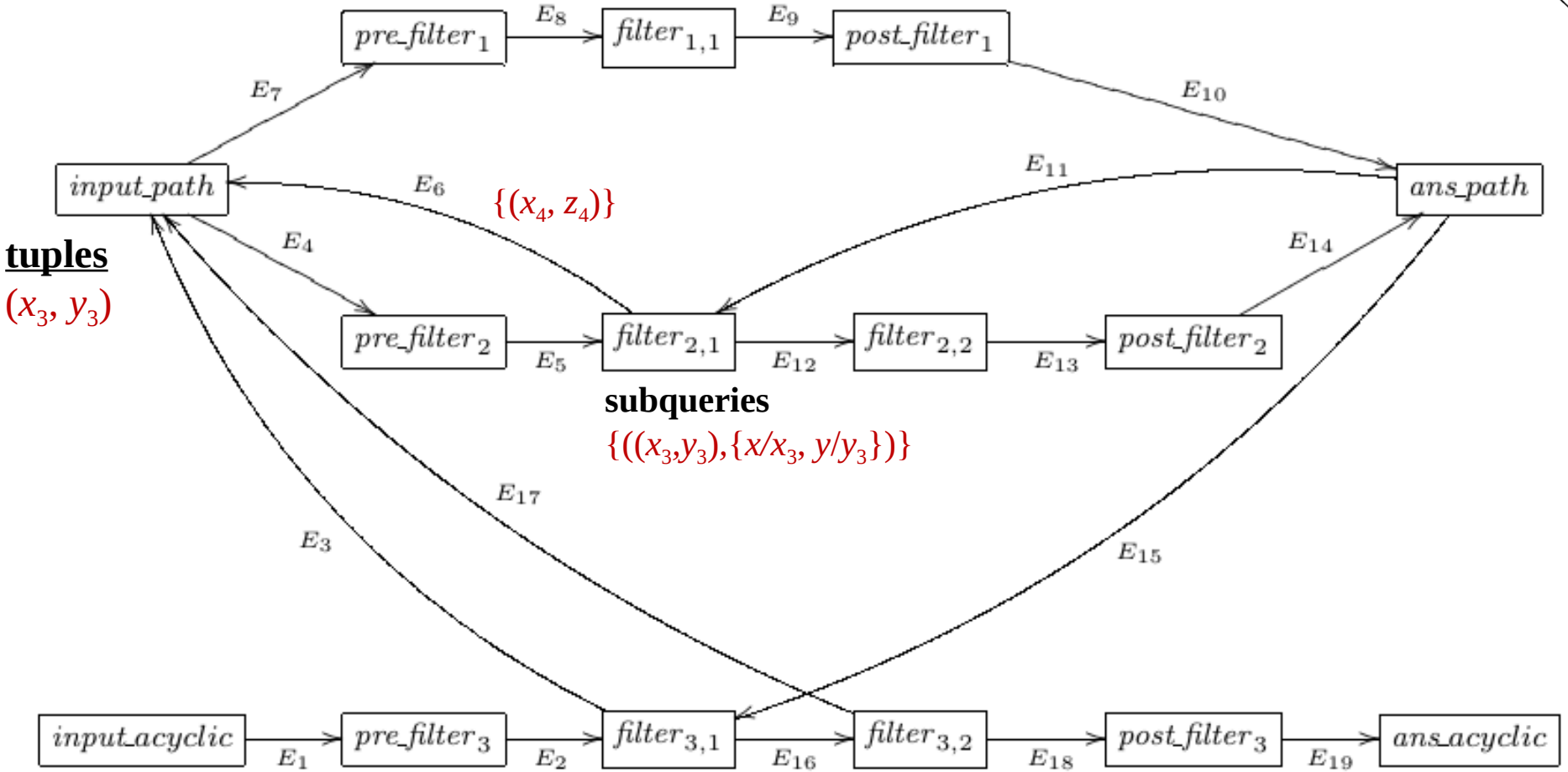
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





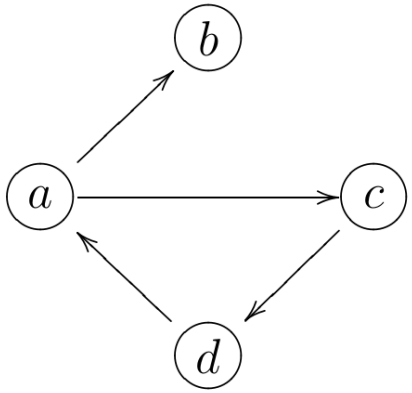
tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

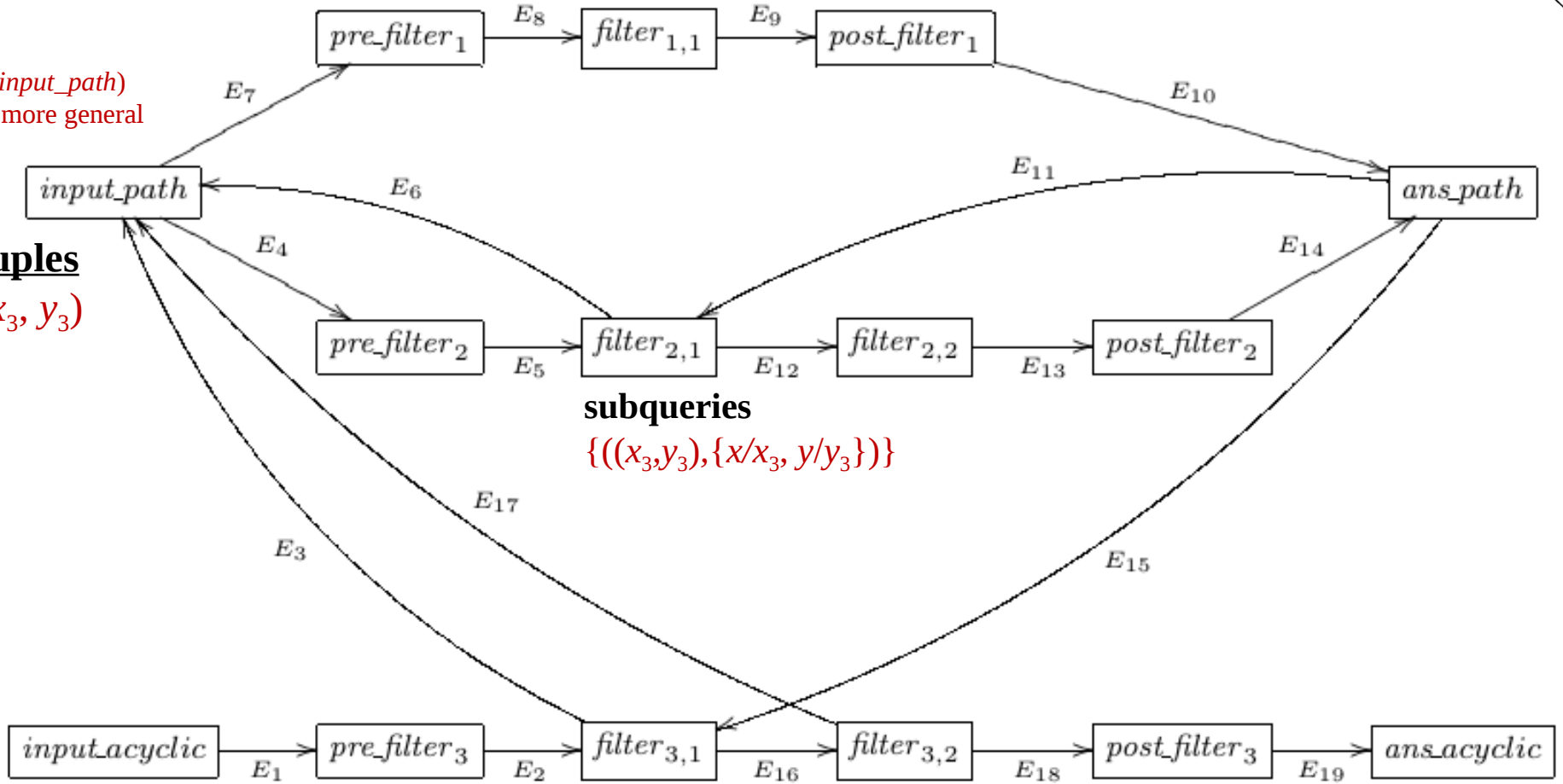
tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$



no tuple is added to tuples(*input_path*) because the tuple (x_3, y_3) is more general than any other tuples



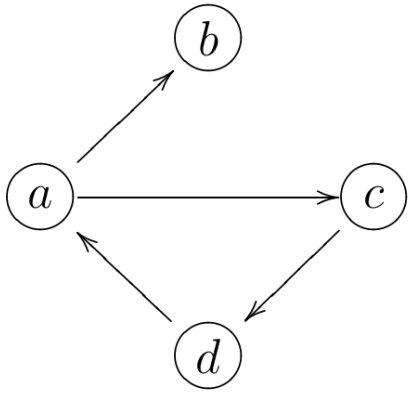
tuples
(x_3, y_3)

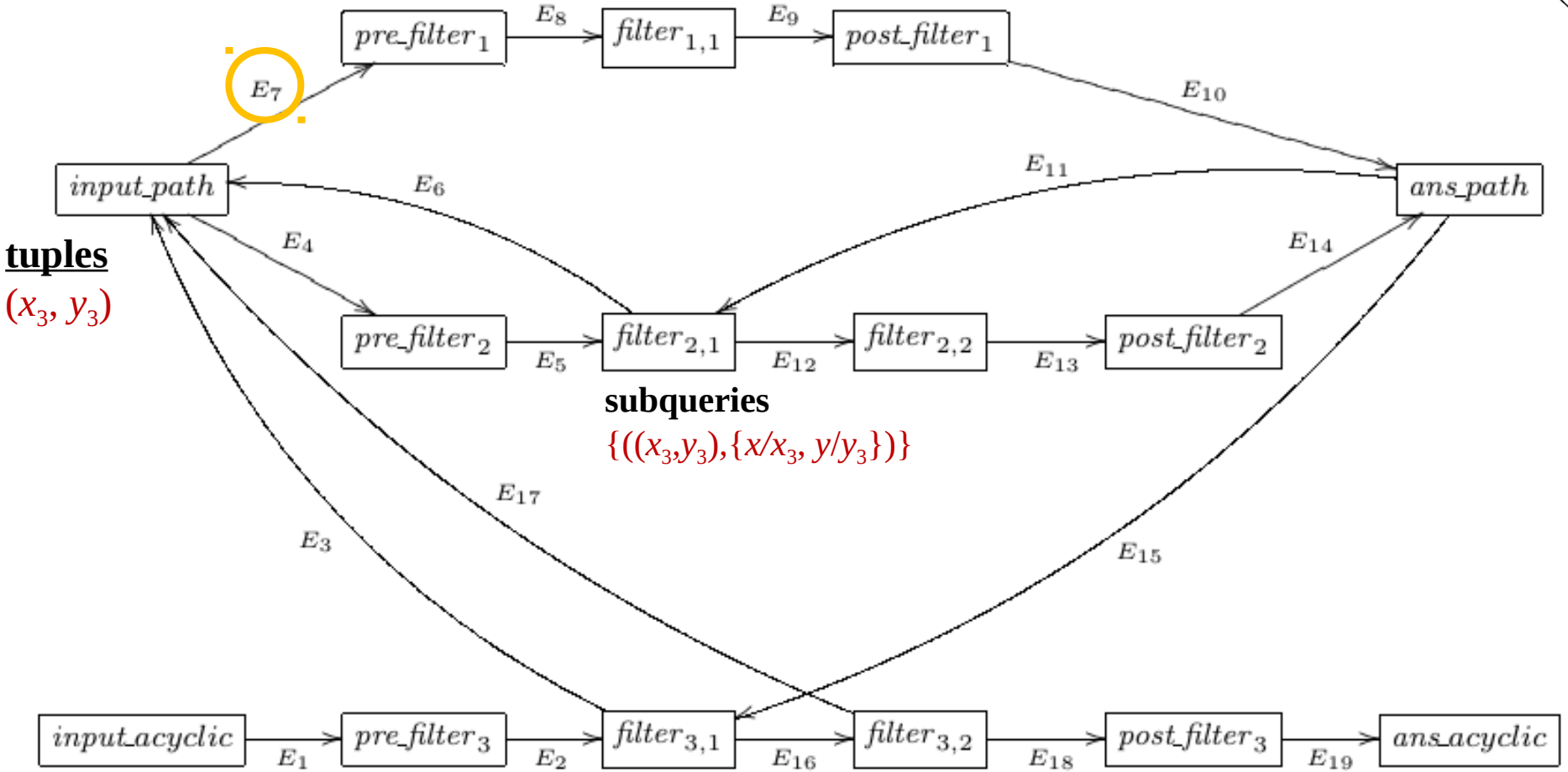
subqueries
{((x_3,y_3),{x/x_3, y/y_3})}

tuples
(x_1, y_1)

subqueries
{((x_1,y_1),{x/x_1, y/y_1})}

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





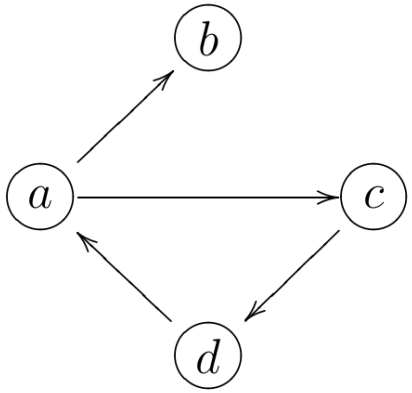
tuples
(x_3, y_3)

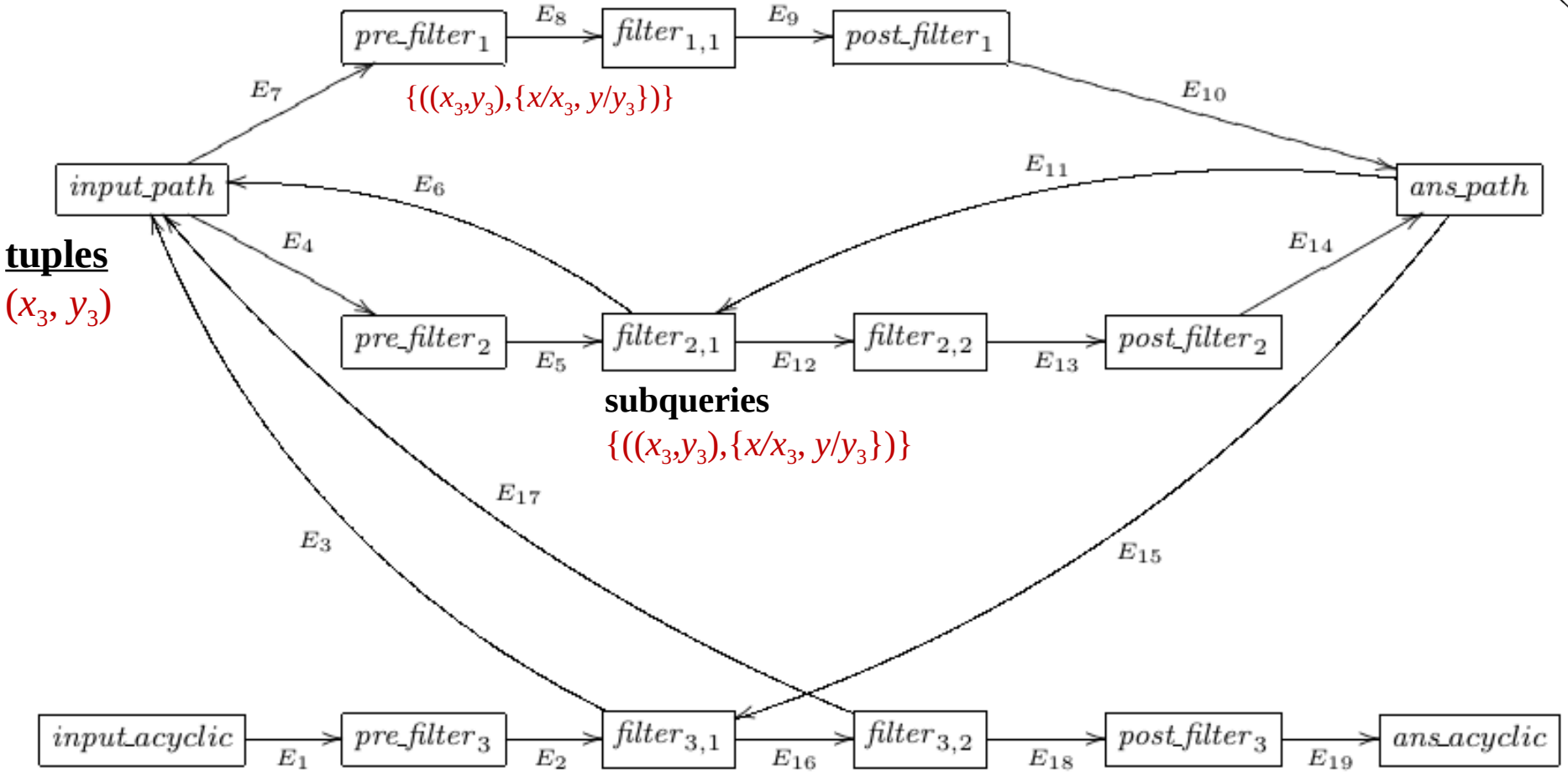
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





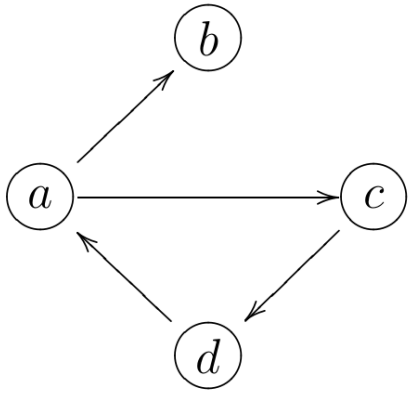
tuples
(x_3, y_3)

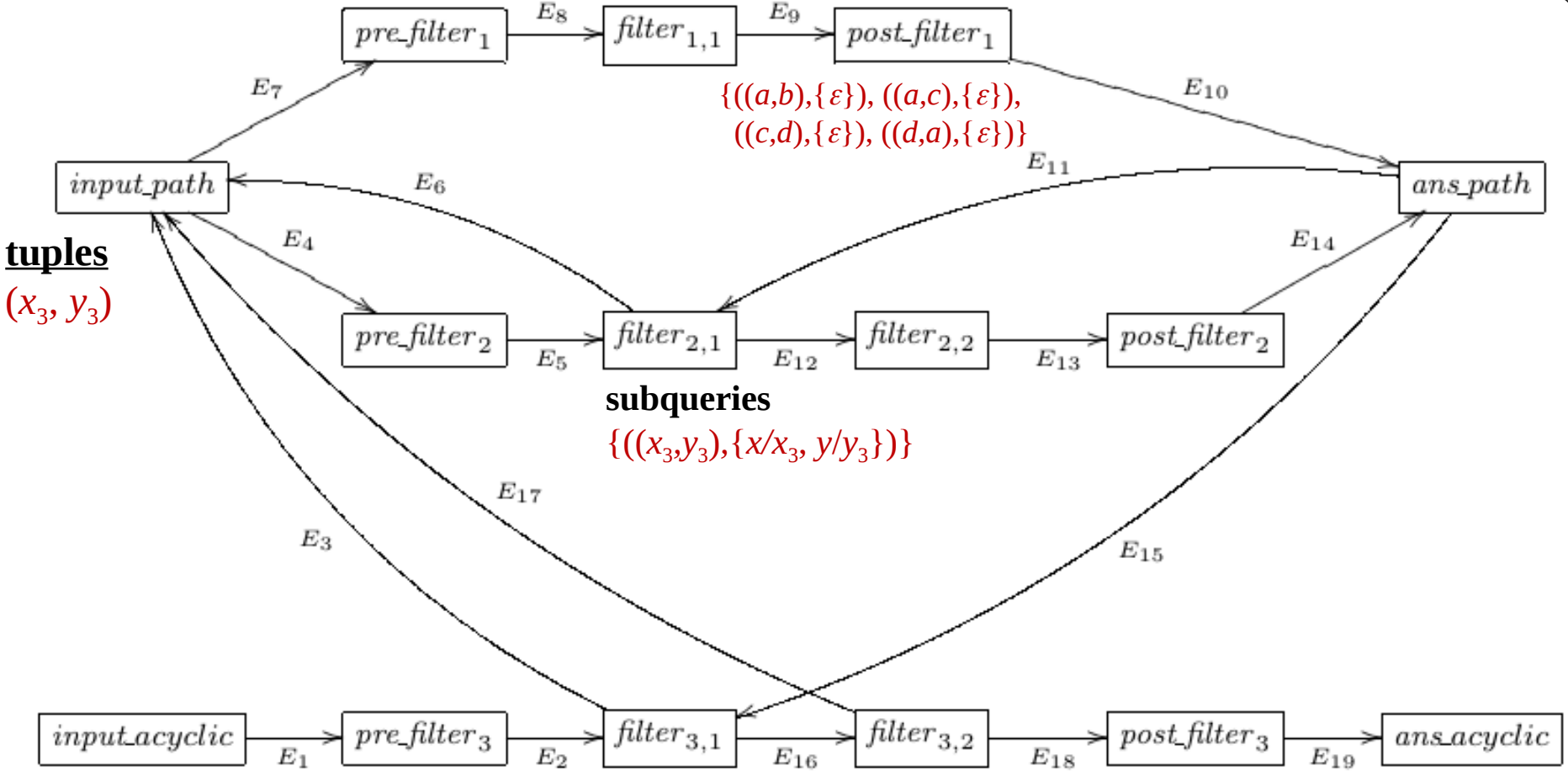
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$



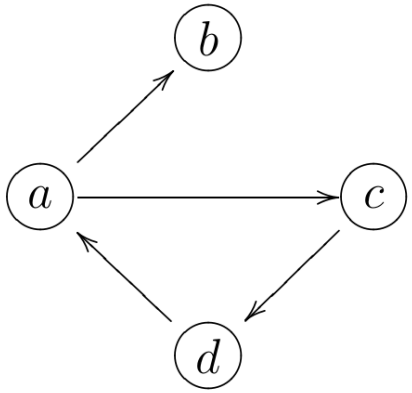


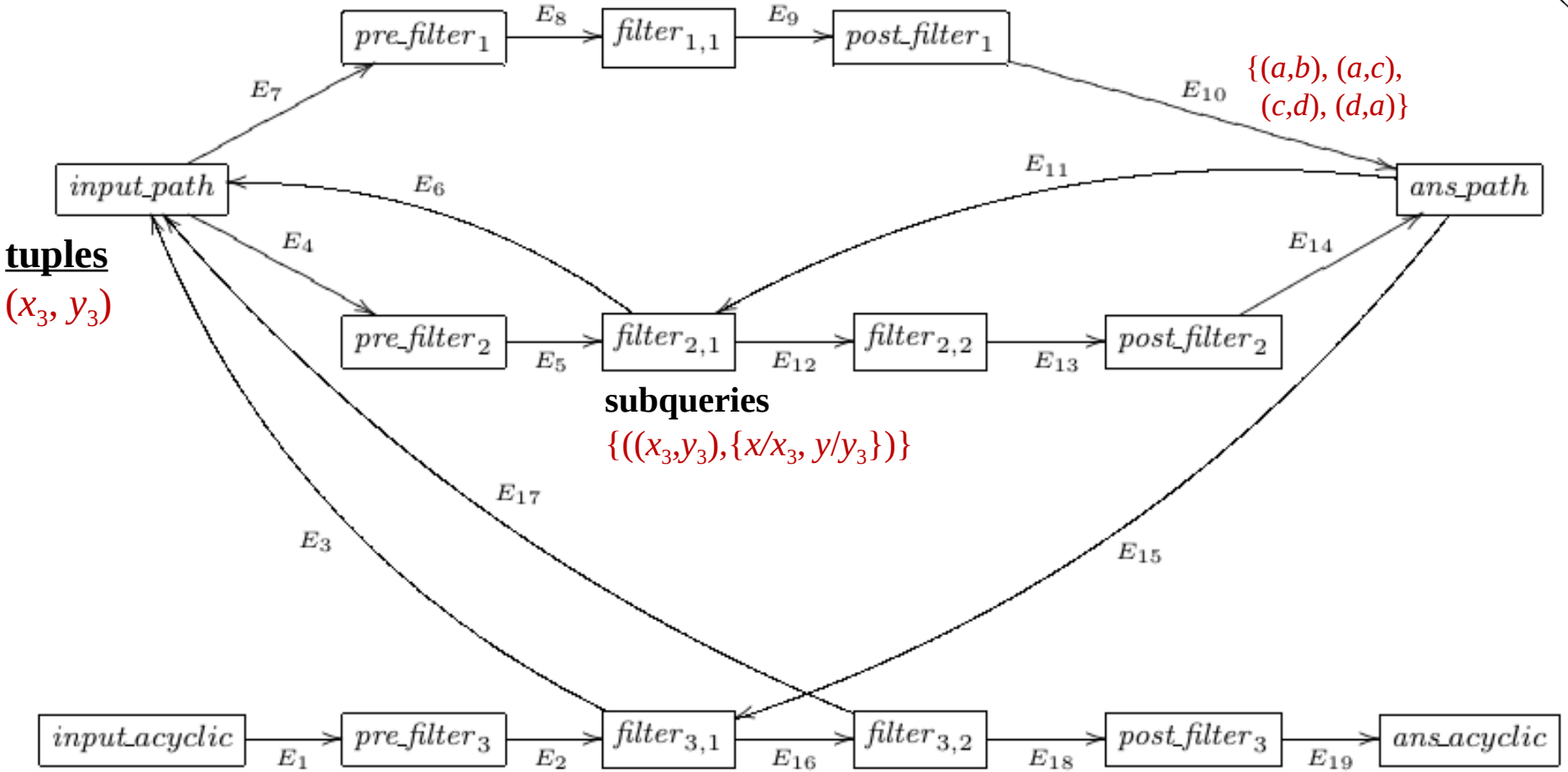
tuples
(x_3, y_3)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





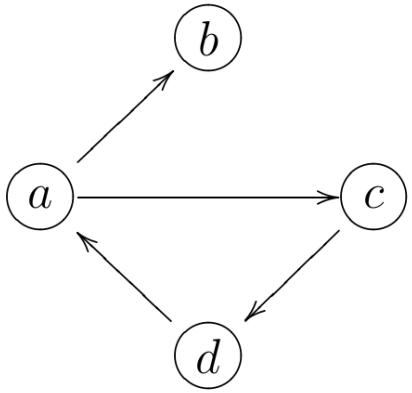
tuples
(x_3, y_3)

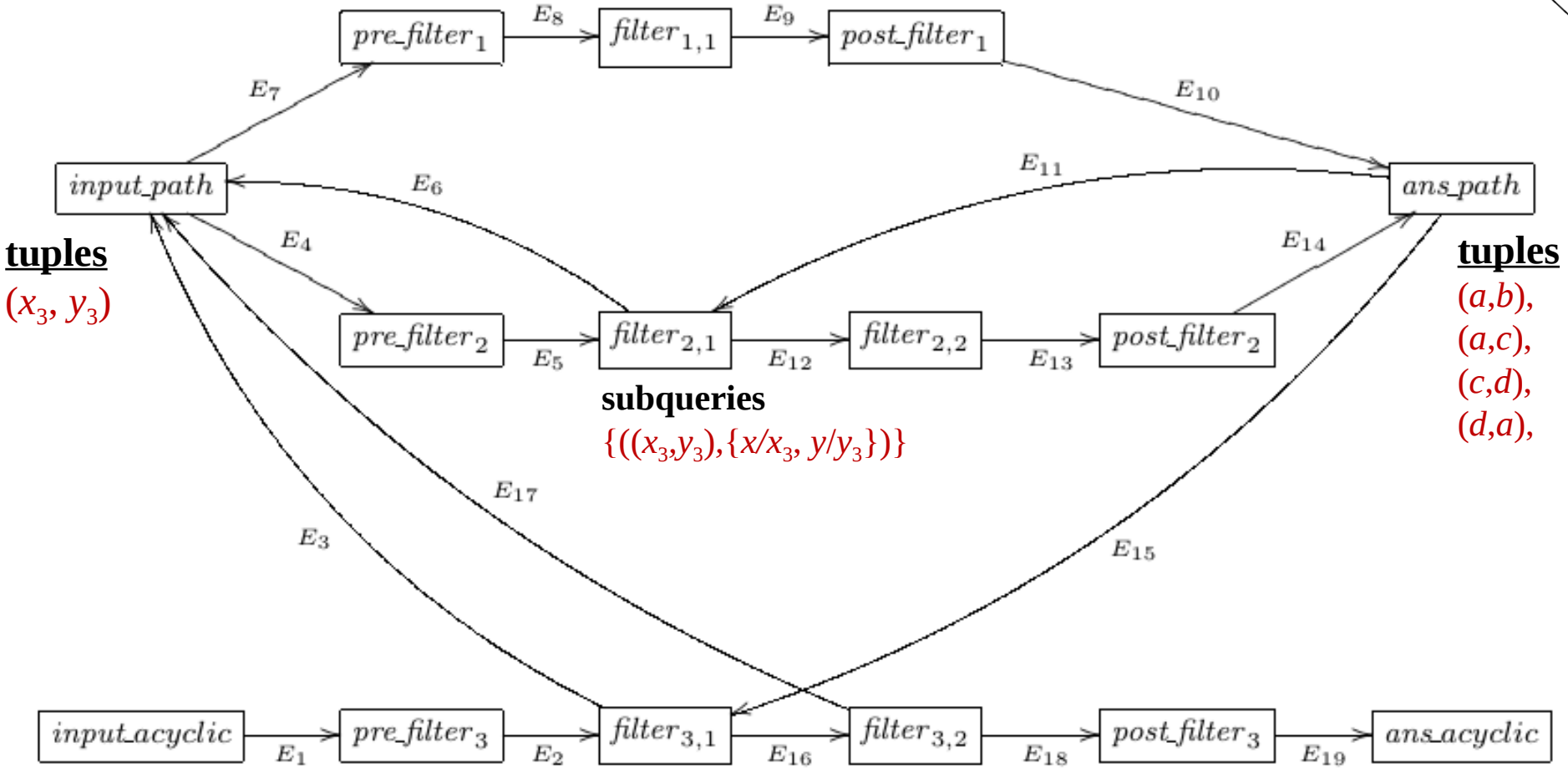
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

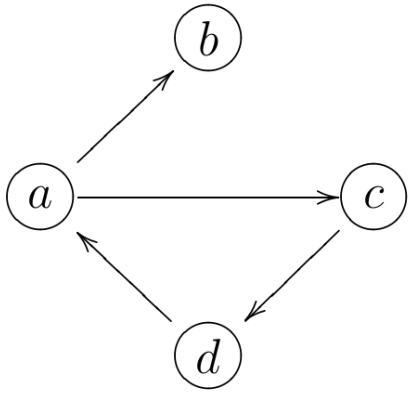
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

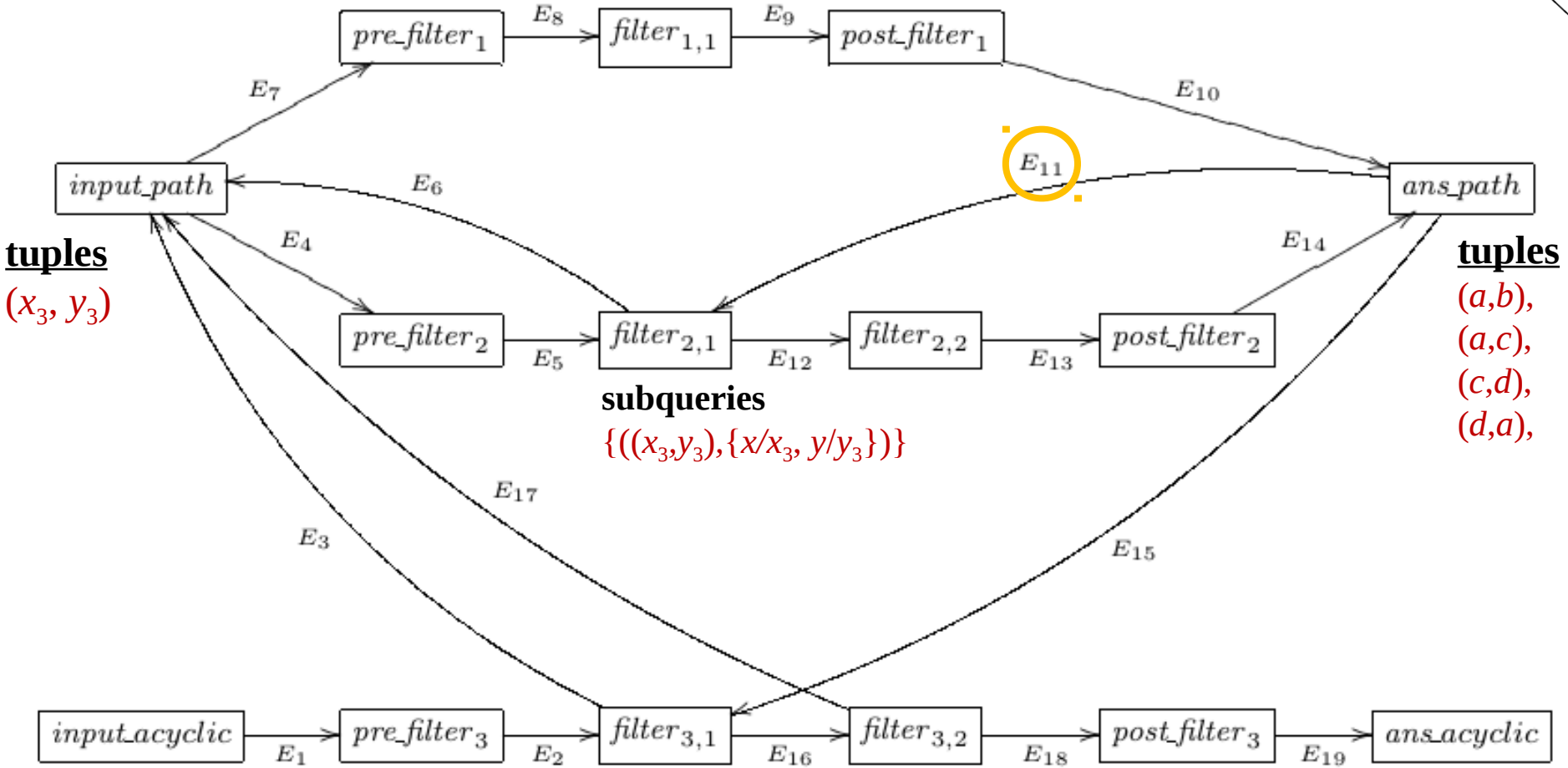
tuples
(a,b),
(a,c),
(c,d),
(d,a),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

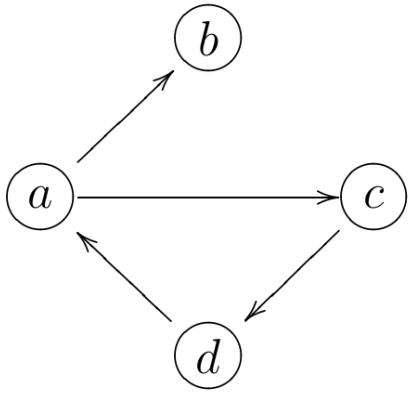
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

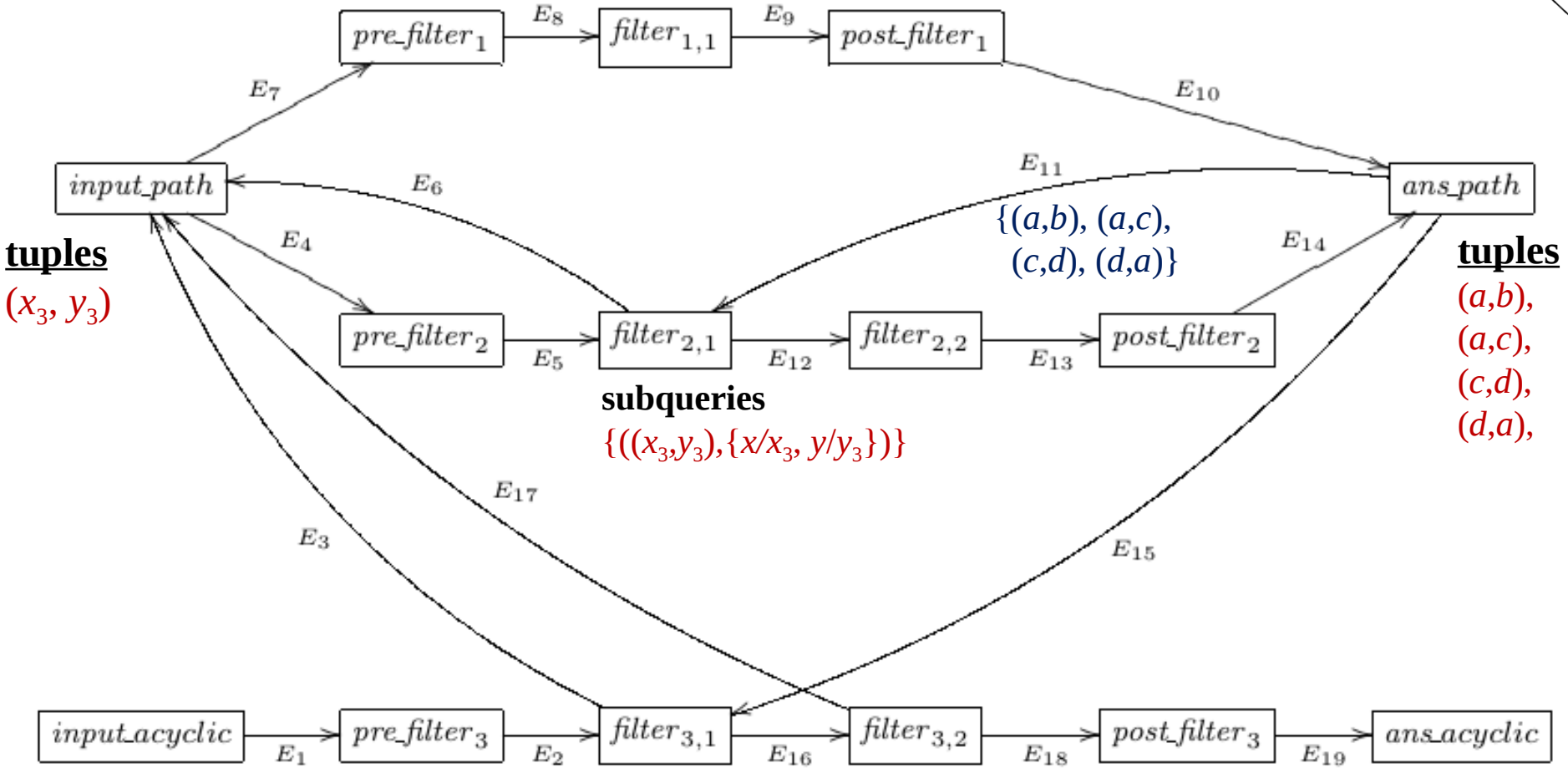
tuples
(a,b),
(a,c),
(c,d),
(d,a),

tuples
(x_1, y_1)

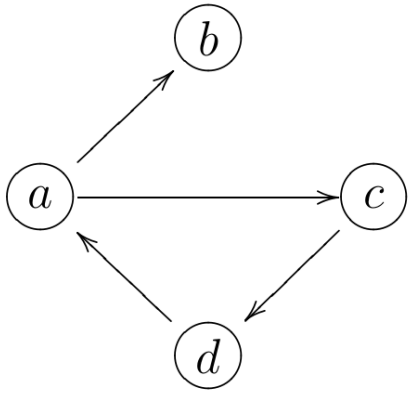
subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

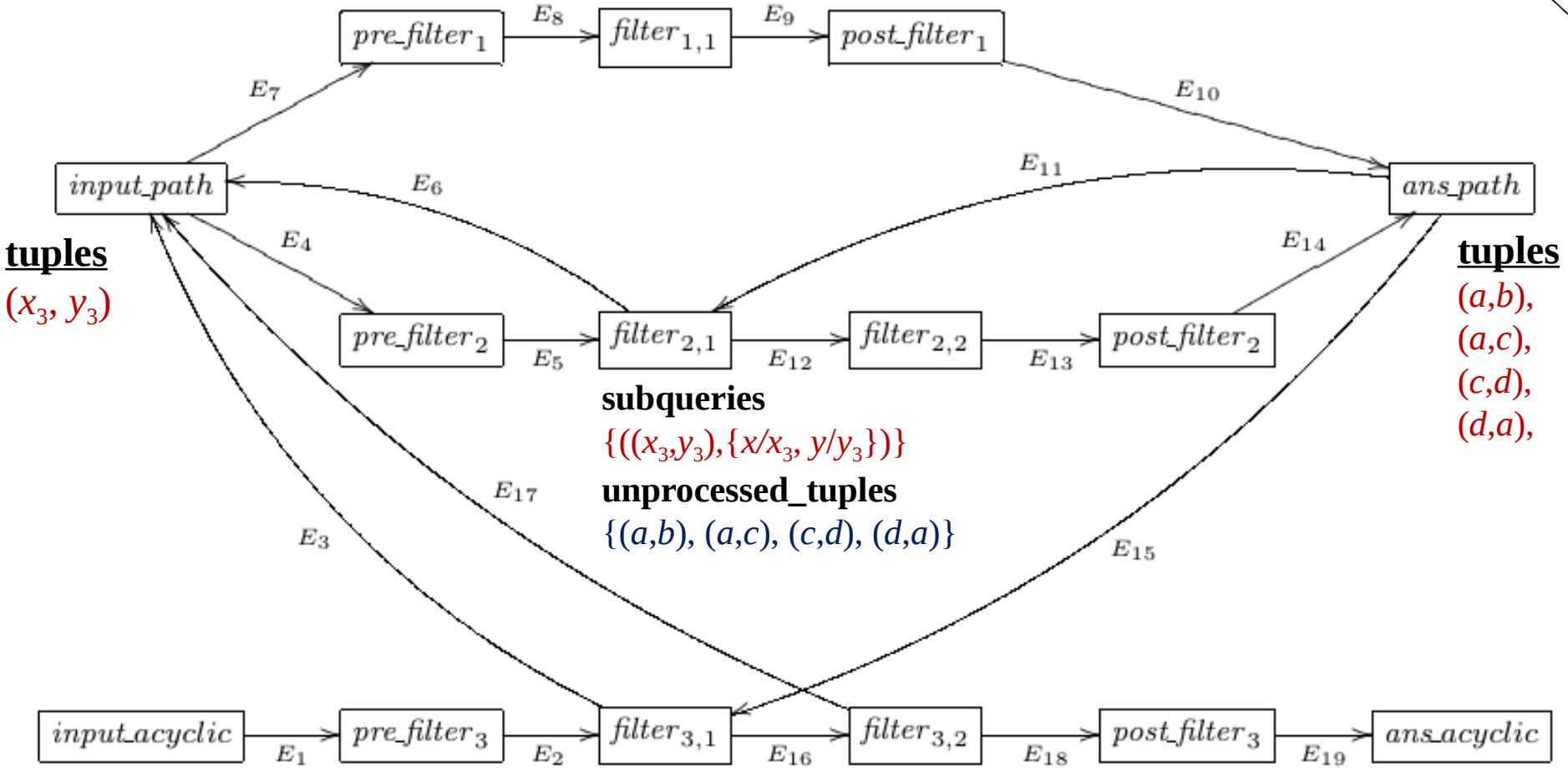
$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

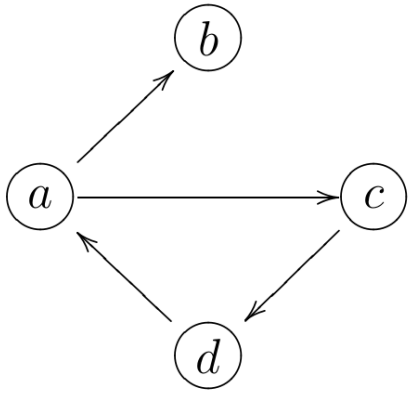
unprocessed_tuples
$\{(a,b), (a,c), (c,d), (d,a)\}$

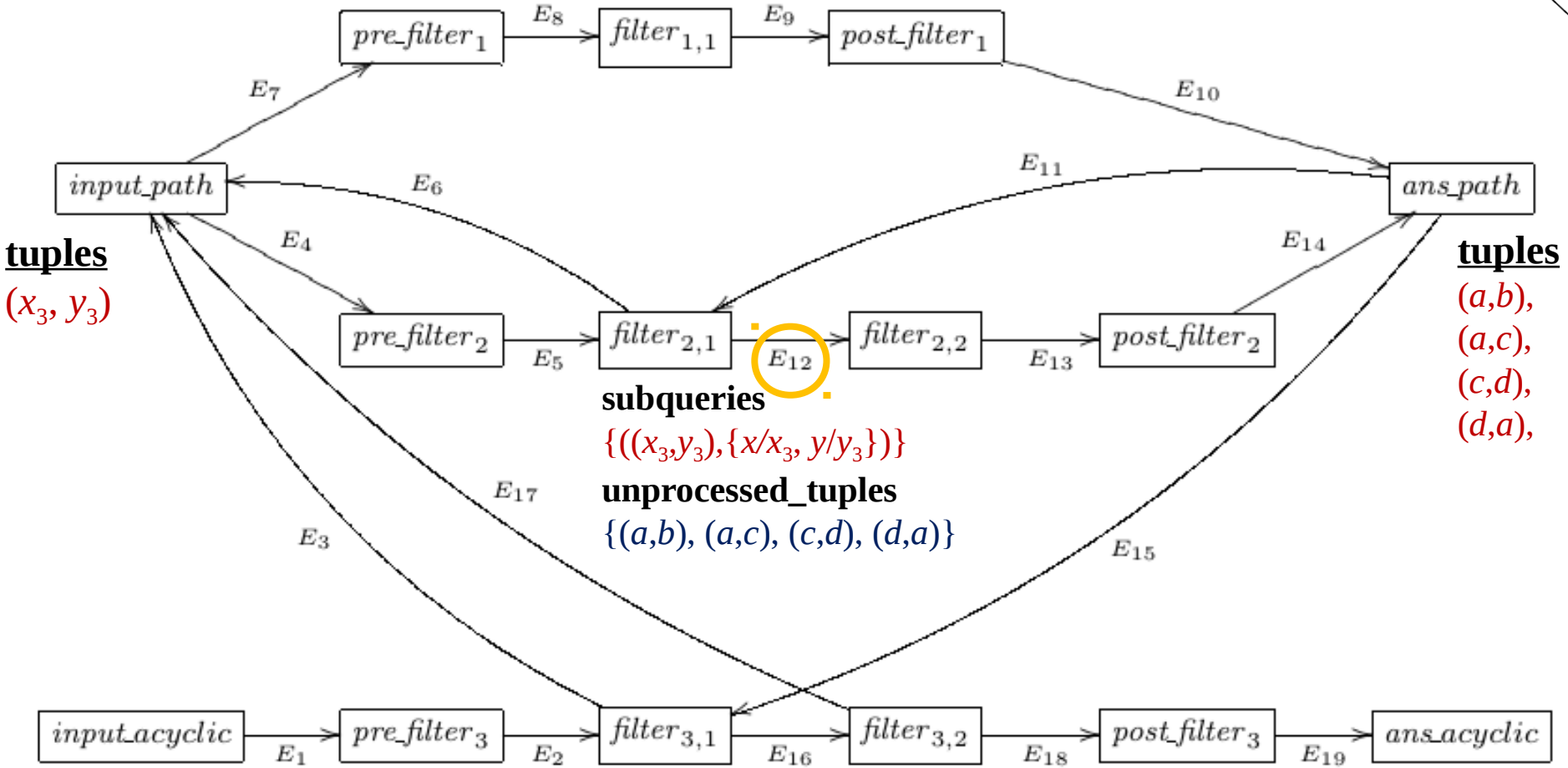
tuples
(a,b),
(a,c),
(c,d),
(d,a),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

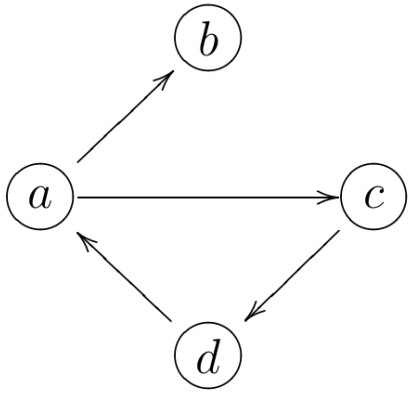
unprocessed_tuples
$\{(a,b), (a,c), (c,d), (d,a)\}$

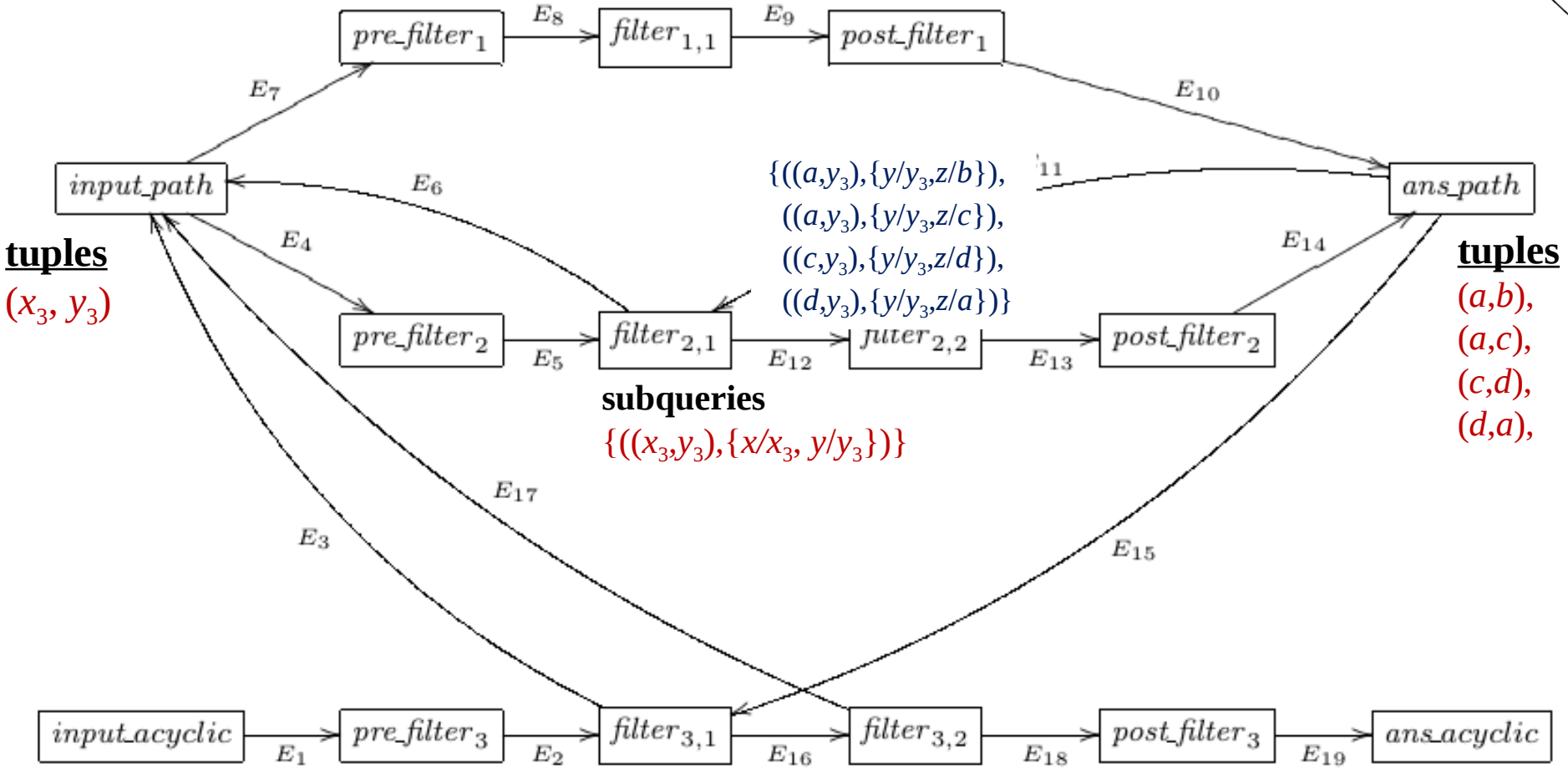
tuples
(a,b),
(a,c),
(c,d),
(d,a),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

$\{((a,y_3),\{y/y_3,z/b\}),$
$((a,y_3),\{y/y_3,z/c\}),$
$((c,y_3),\{y/y_3,z/d\}),$
$((d,y_3),\{y/y_3,z/a\})\}$

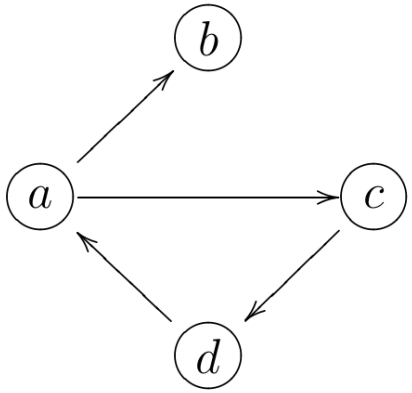
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

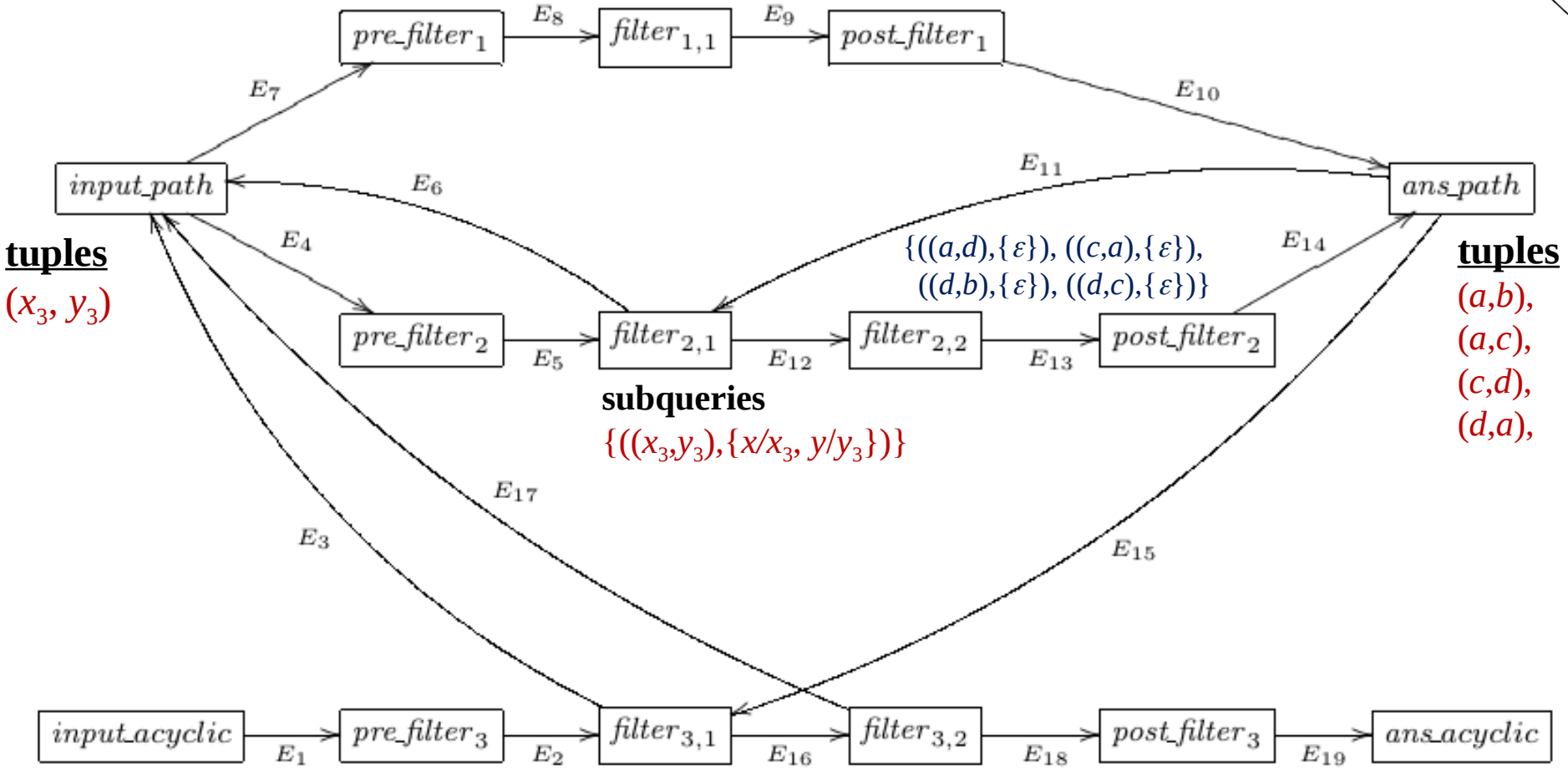
tuples
$(a,b),$
$(a,c),$
$(c,d),$
$(d,a),$

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

$\{((a,d),\{\varepsilon\}), ((c,a),\{\varepsilon\}), ((d,b),\{\varepsilon\}), ((d,c),\{\varepsilon\})\}$

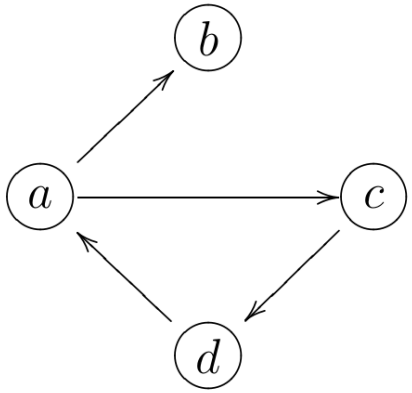
tuples
(a,b),
(a,c),
(c,d),
(d,a),

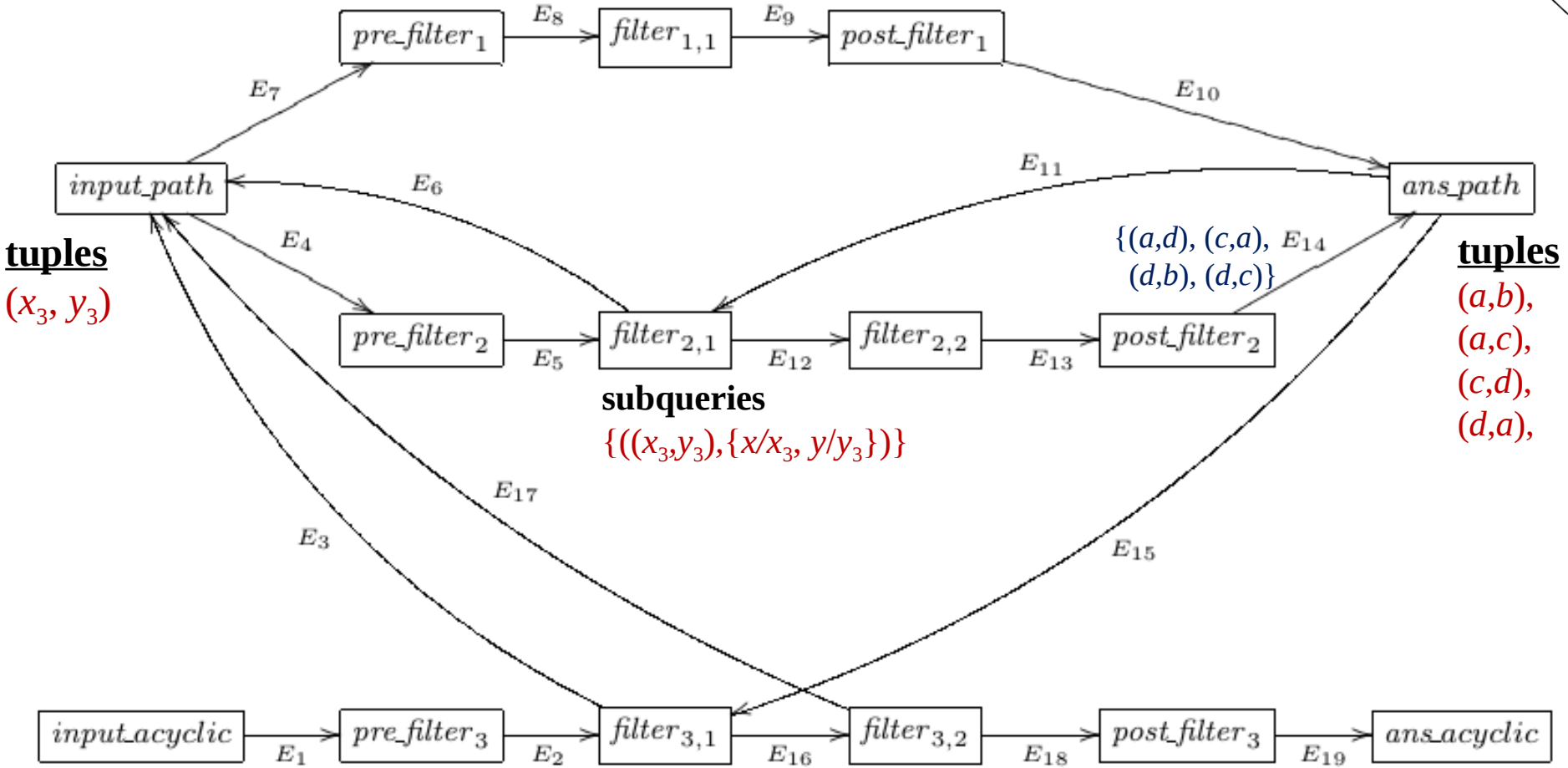
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

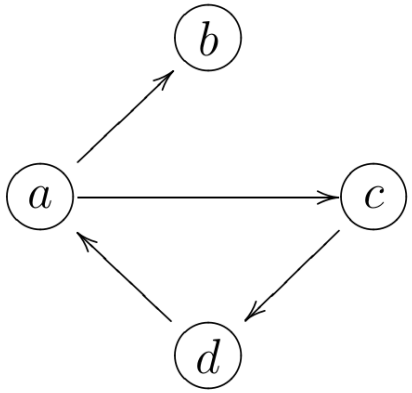
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

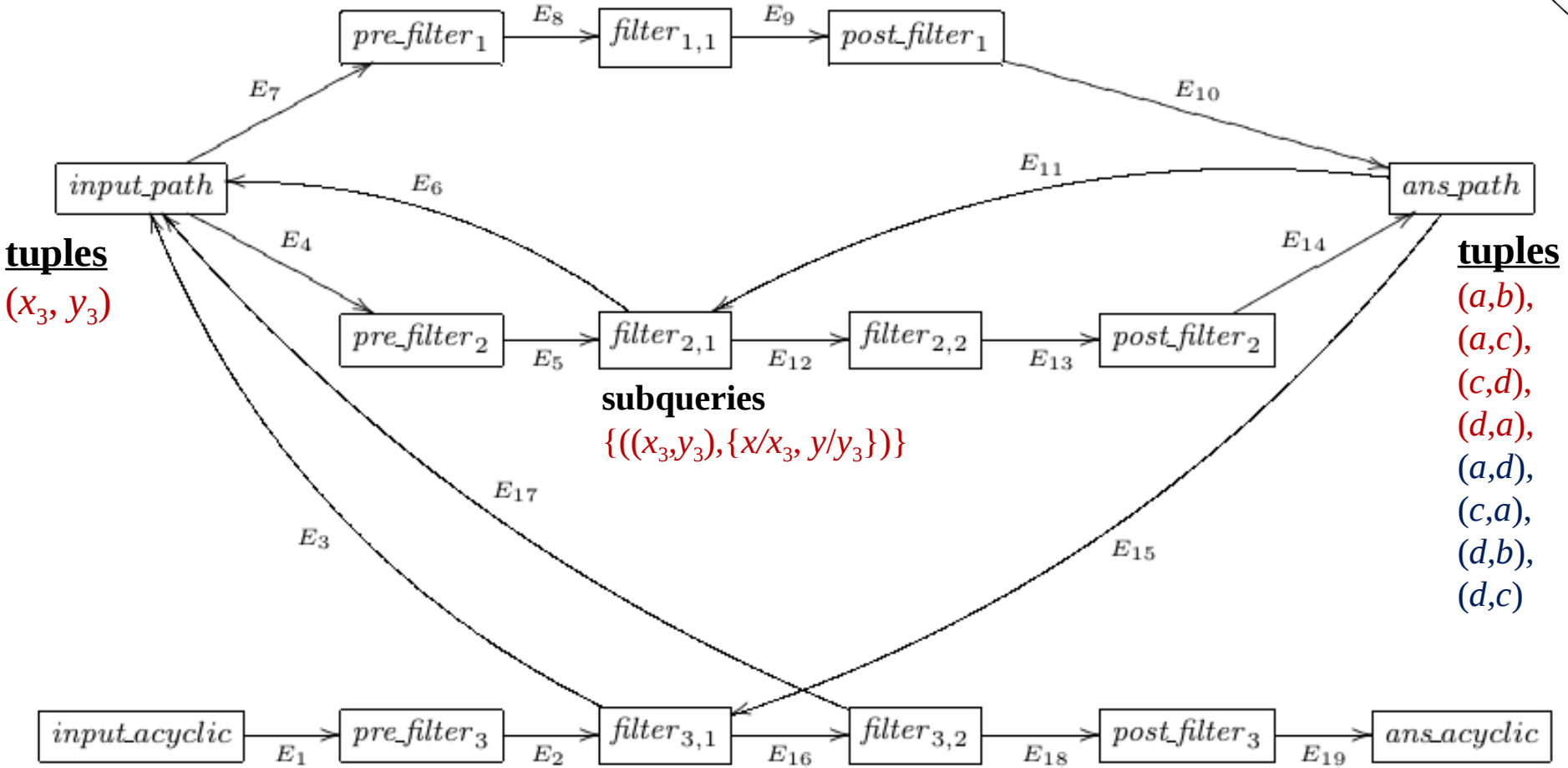
tuples
(a,b),
(a,c),
(c,d),
(d,a),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

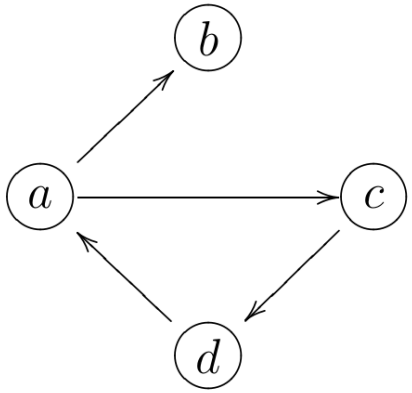
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

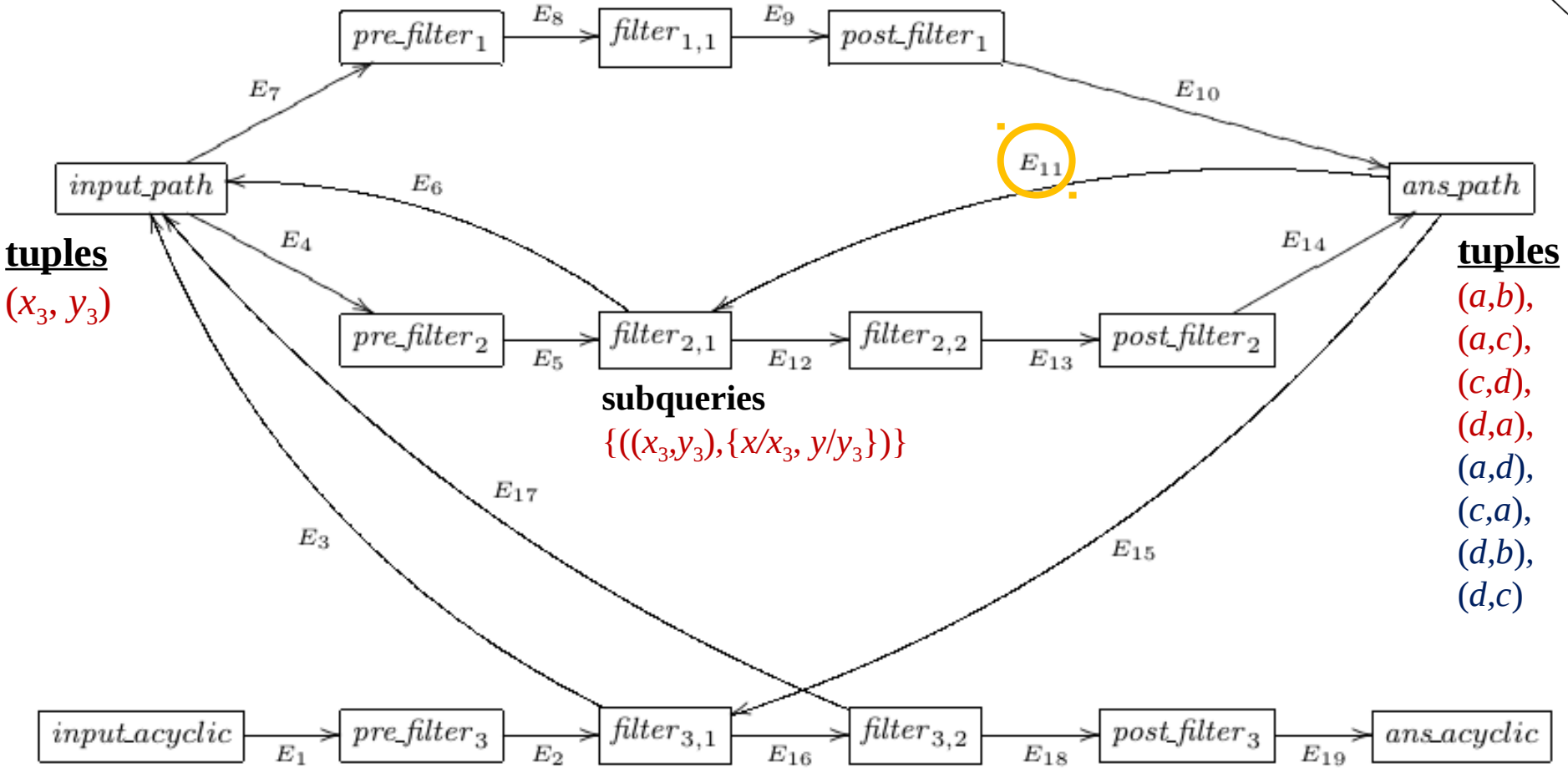
tuples
(a,b),
(a,c),
(c,d),
(d,a),
(a,d),
(c,a),
(d,b),
(d,c)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

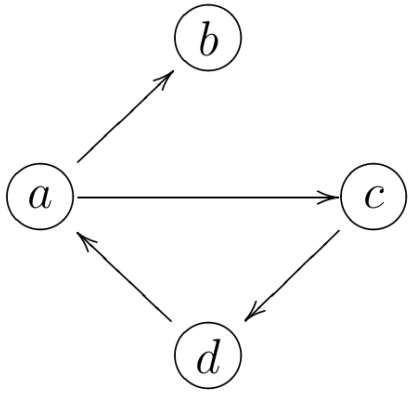
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

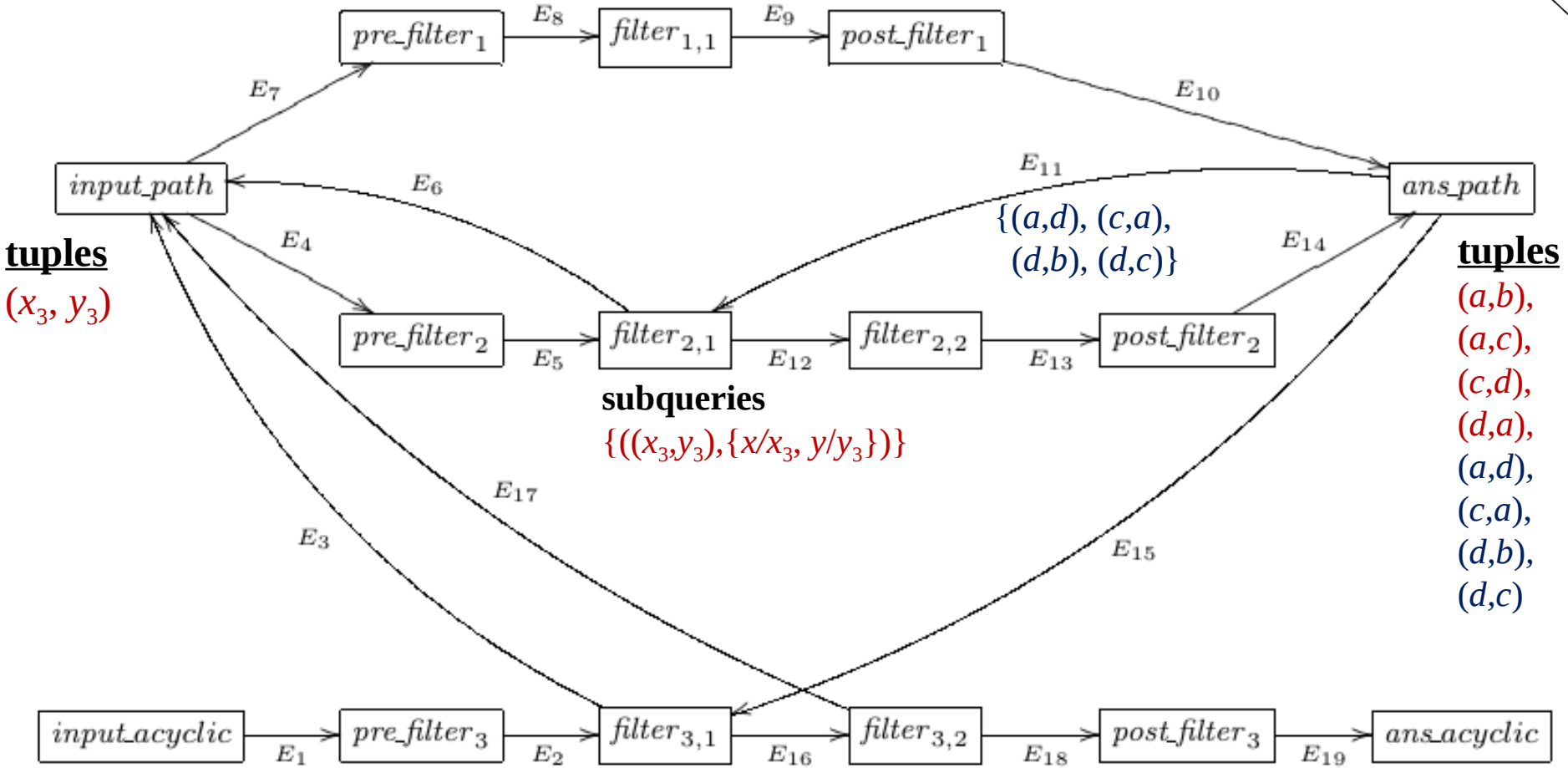
tuples
(a,b),
(a,c),
(c,d),
(d,a),
(a,d),
(c,a),
(d,b),
(d,c)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

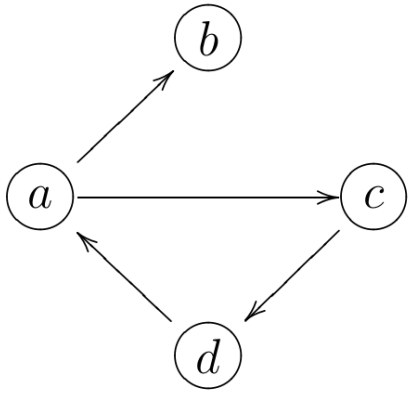
$\{(a,d), (c,a), (d,b), (d,c)\}$

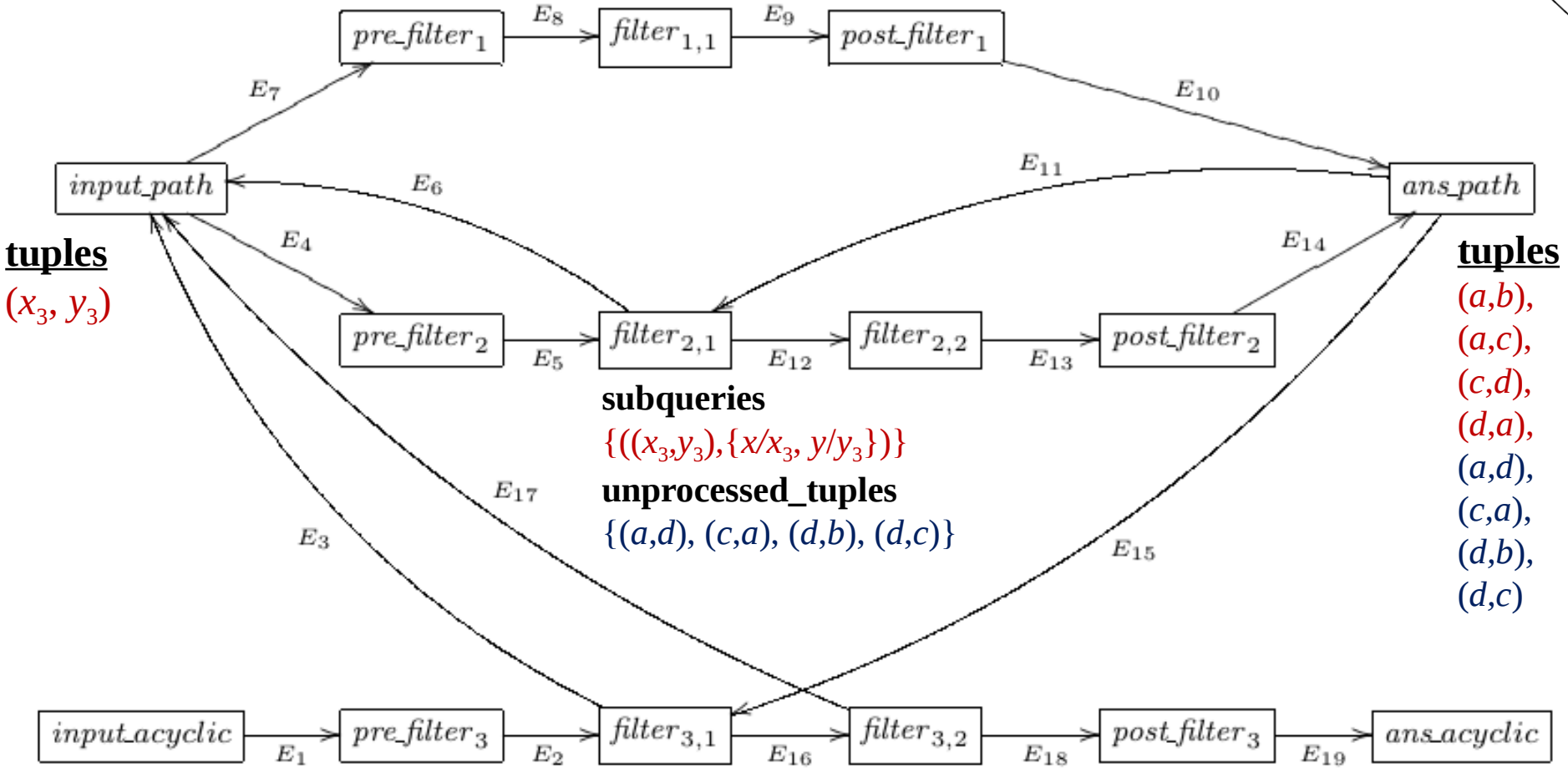
tuples
(a,b),
(a,c),
(c,d),
(d,a),
(a,d),
(c,a),
(d,b),
(d,c)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

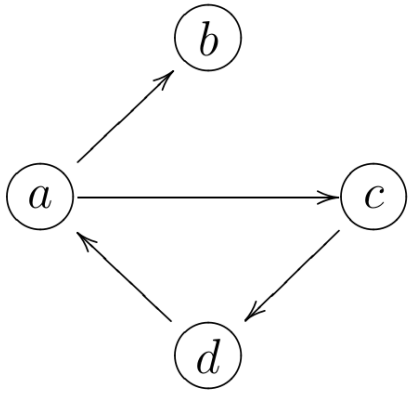
unprocessed_tuples
$\{(a,d), (c,a), (d,b), (d,c)\}$

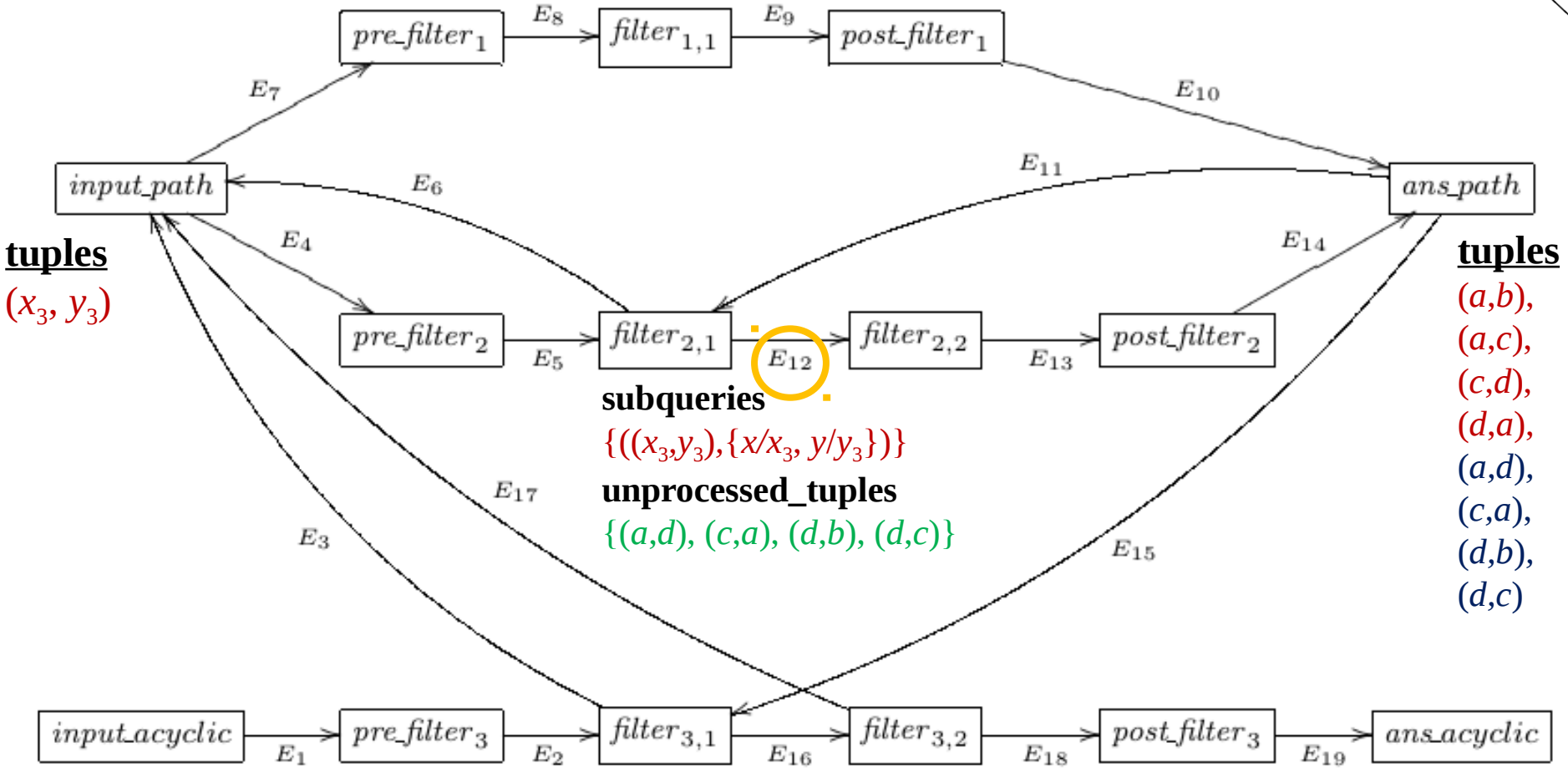
tuples
(a,b),
(a,c),
(c,d),
(d,a),
(a,d),
(c,a),
(d,b),
(d,c)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

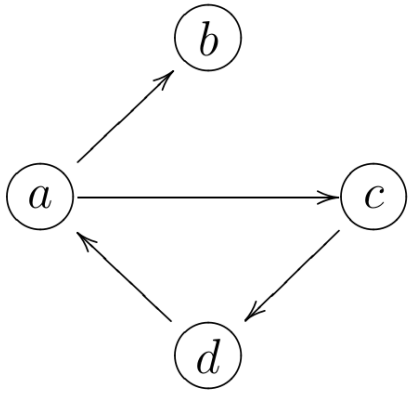
unprocessed_tuples
{(*a*,*d*), (*c*,*a*), (*d*,*b*), (*d*,*c*)}

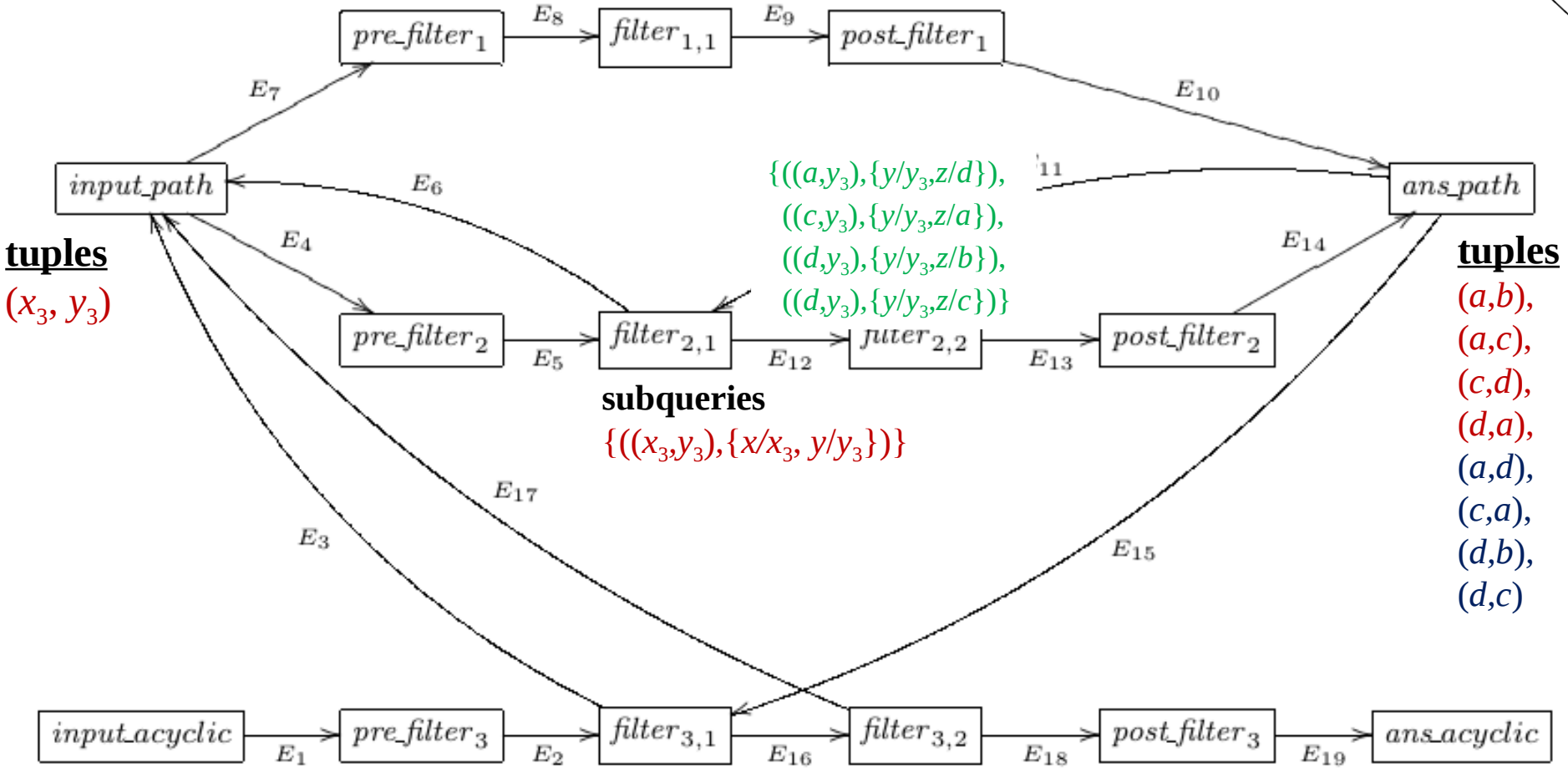
tuples
(*a*,*b*),
(*a*,*c*),
(*c*,*d*),
(*d*,*a*),
(*a*,*d*),
(*c*,*a*),
(*d*,*b*),
(*d*,*c*)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

$\{((a,y_3),\{y/y_3,z/d\}),$
$((c,y_3),\{y/y_3,z/a\}),$
$((d,y_3),\{y/y_3,z/b\}),$
$((d,y_3),\{y/y_3,z/c\})\}$

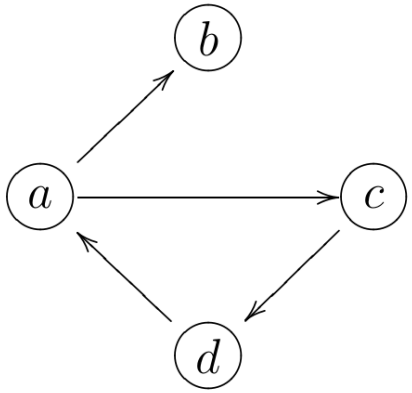
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

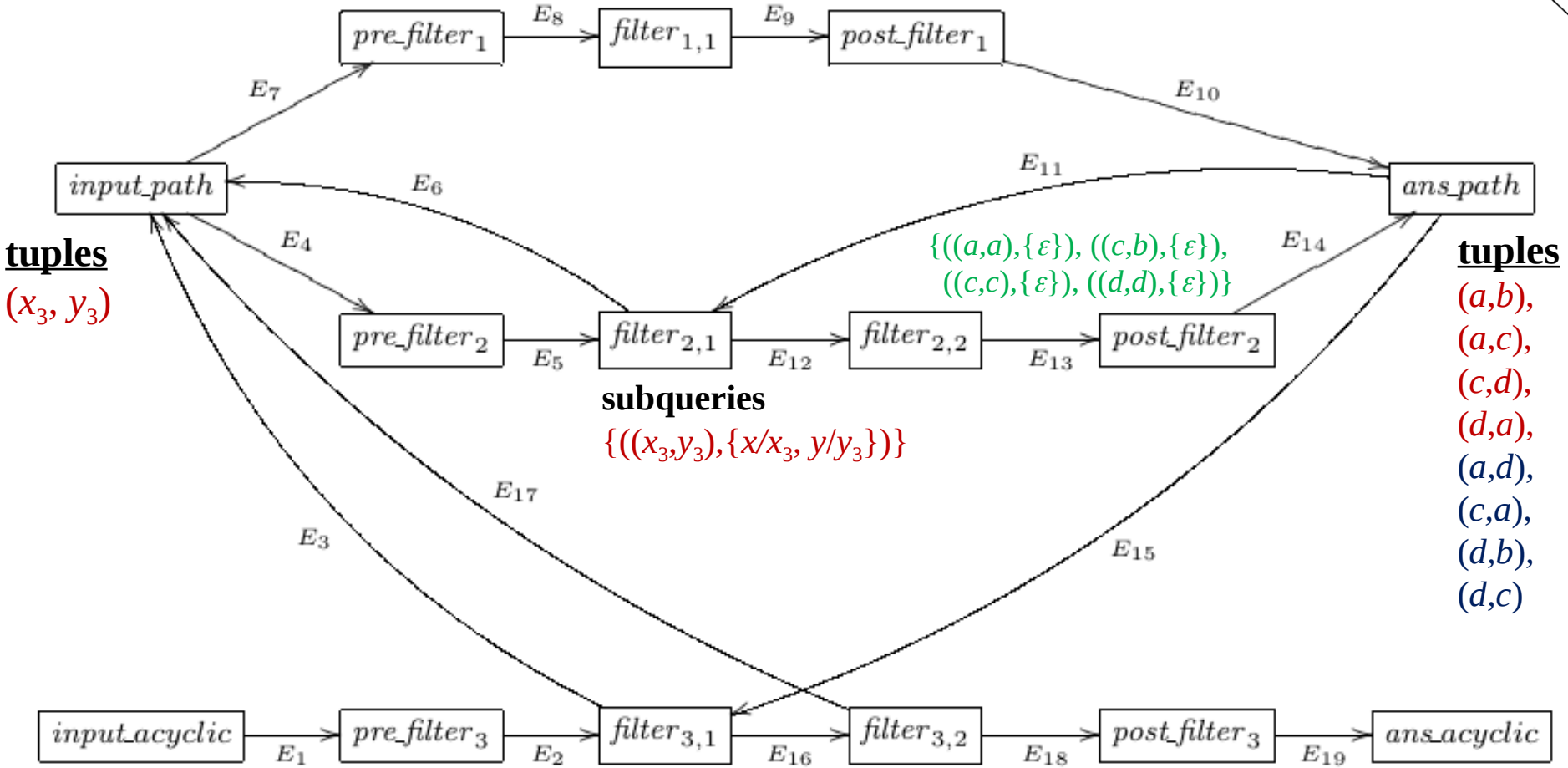
tuples
(a,b),
(a,c),
(c,d),
(d,a),
(a,d),
(c,a),
(d,b),
(d,c)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

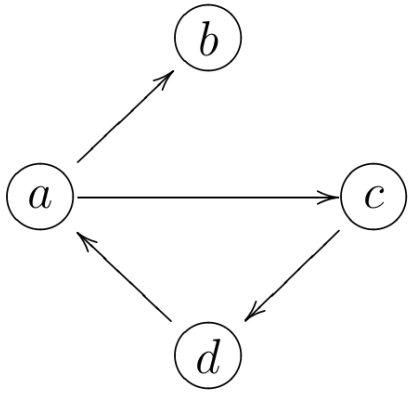
$\{((a,a),\{\varepsilon\}), ((c,b),\{\varepsilon\}),$
$((c,c),\{\varepsilon\}), ((d,d),\{\varepsilon\})\}$

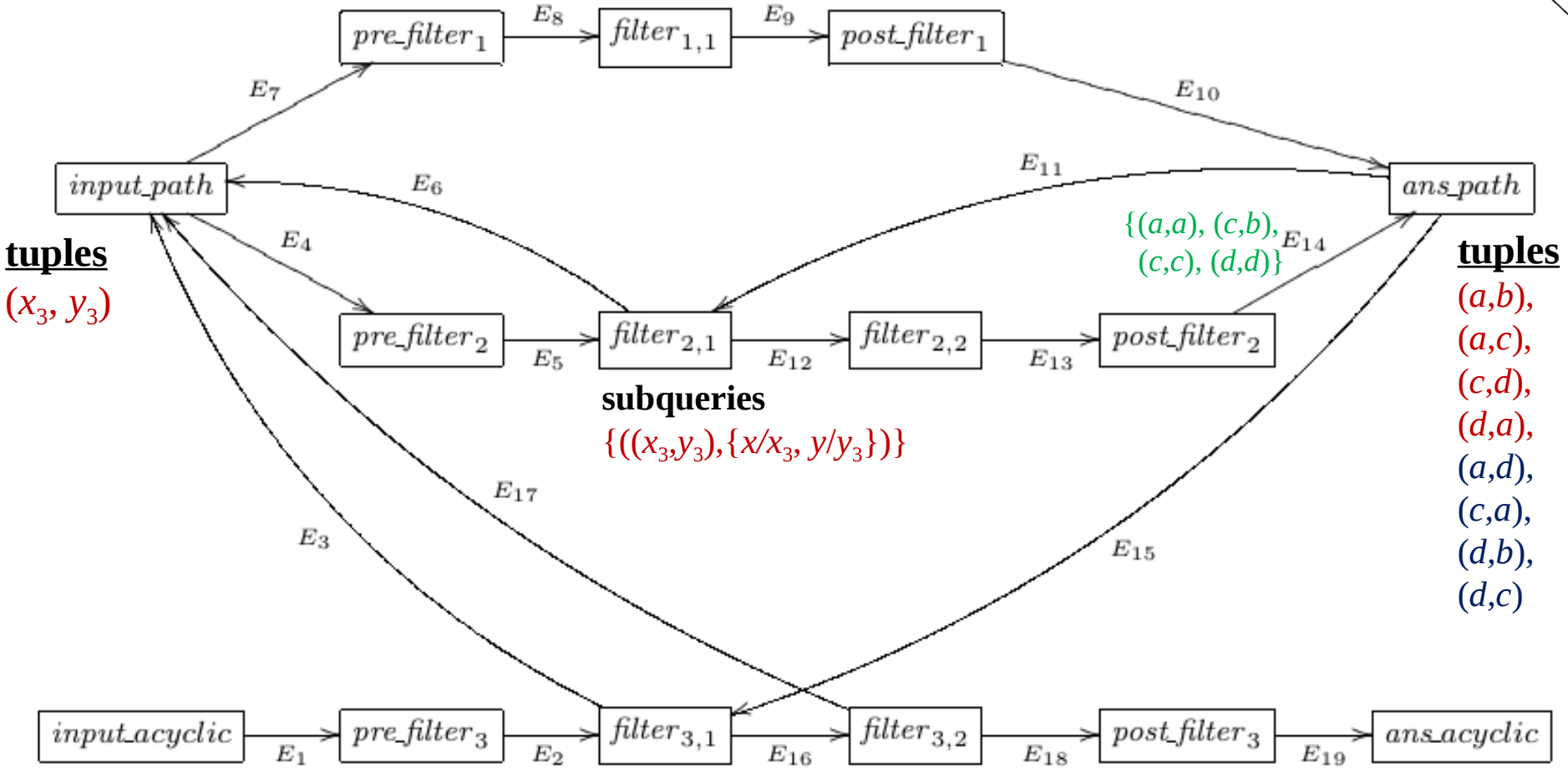
tuples
(a,b),
(a,c),
(c,d),
(d,a),
(a,d),
(c,a),
(d,b),
(d,c)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

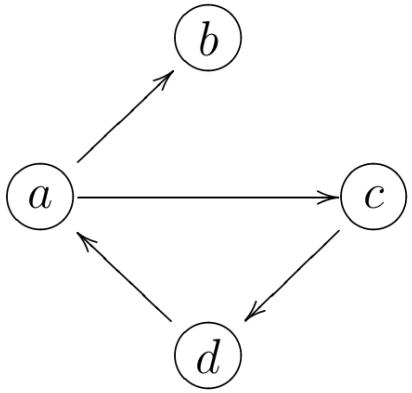
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

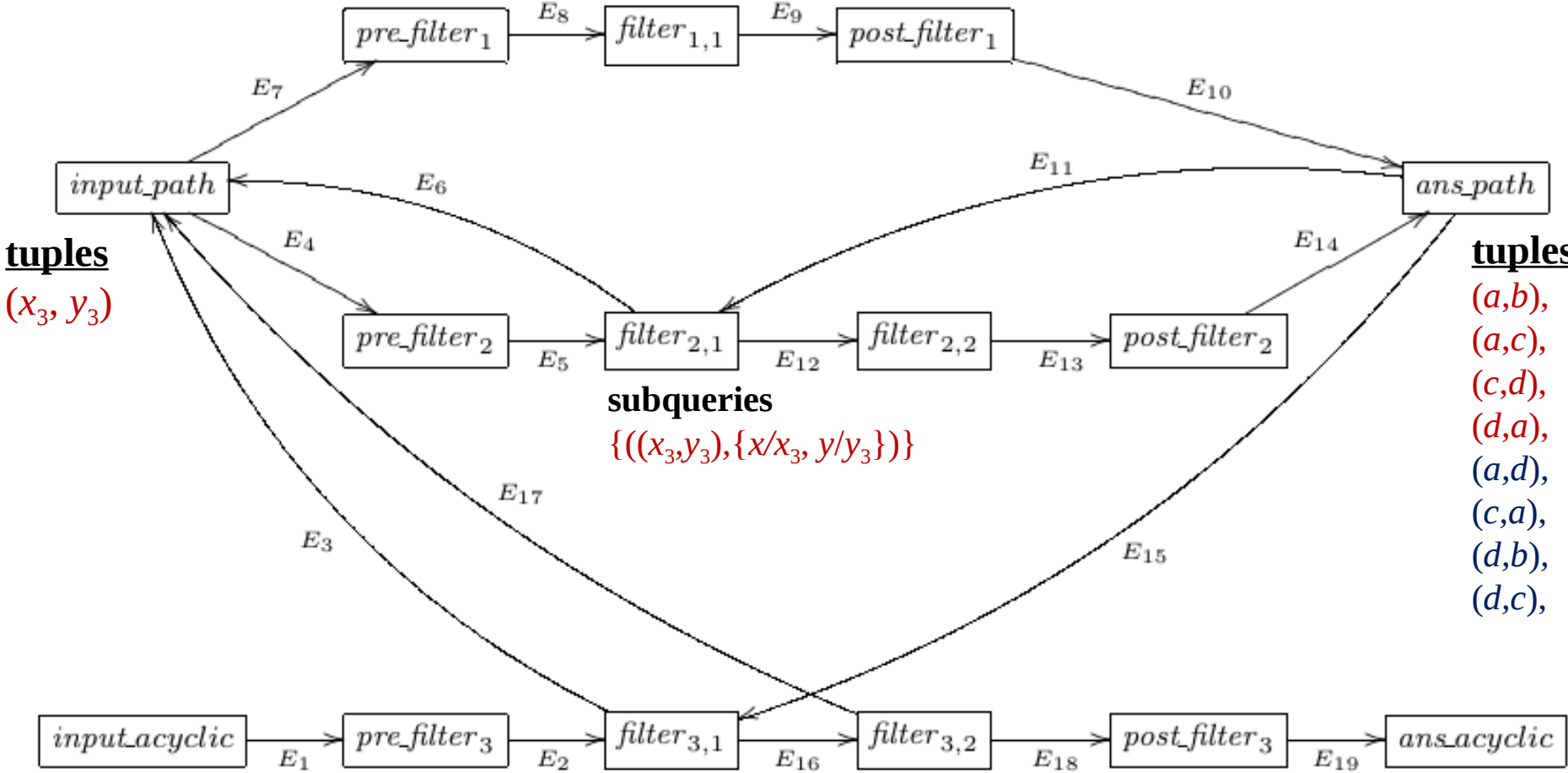
tuples
(a,b),
(a,c),
(c,d),
(d,a),
(a,d),
(c,a),
(d,b),
(d,c)

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

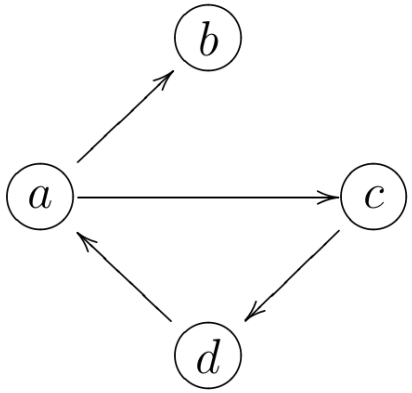
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

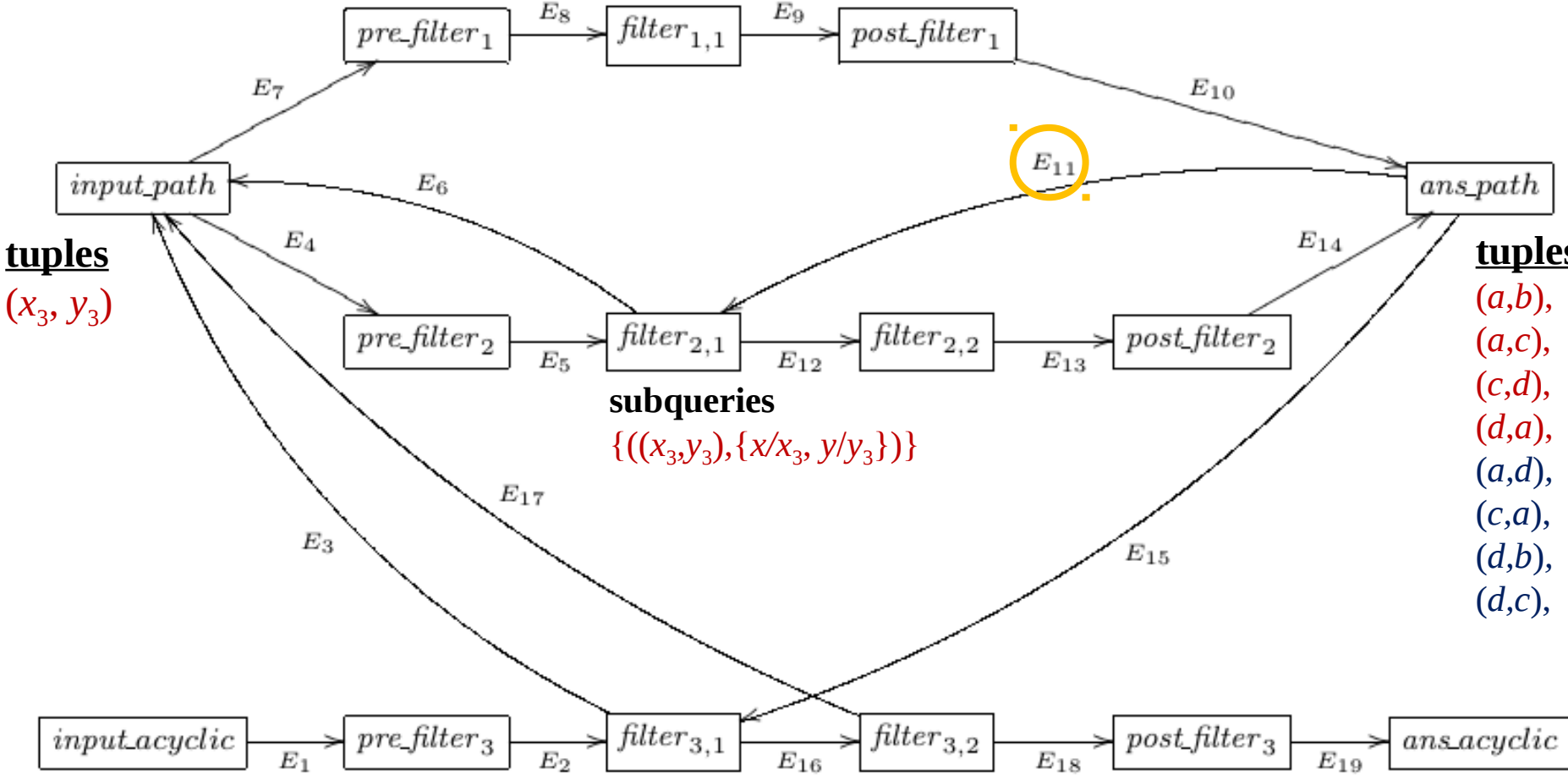
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

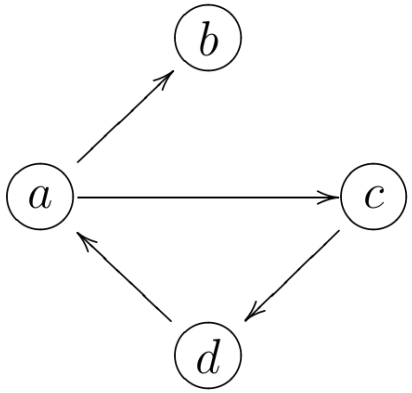
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

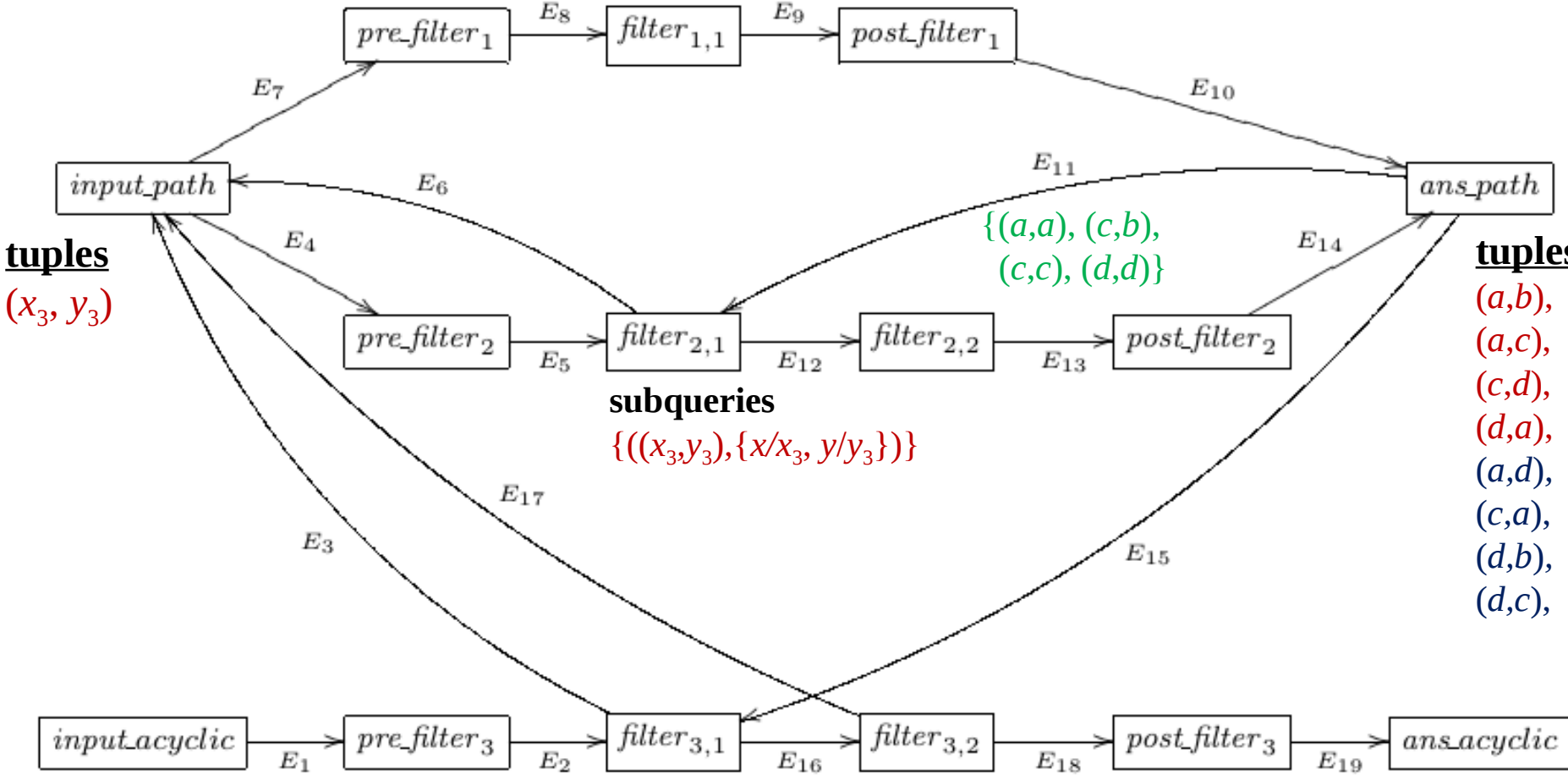
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

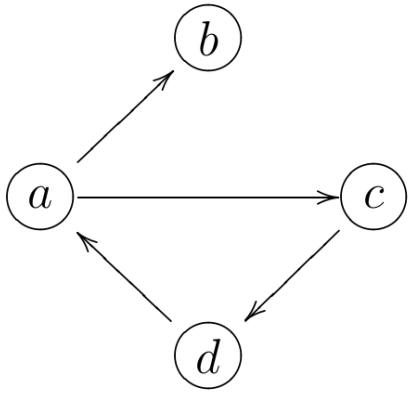
{(a,a), (c,b), (c,c), (d,d)}

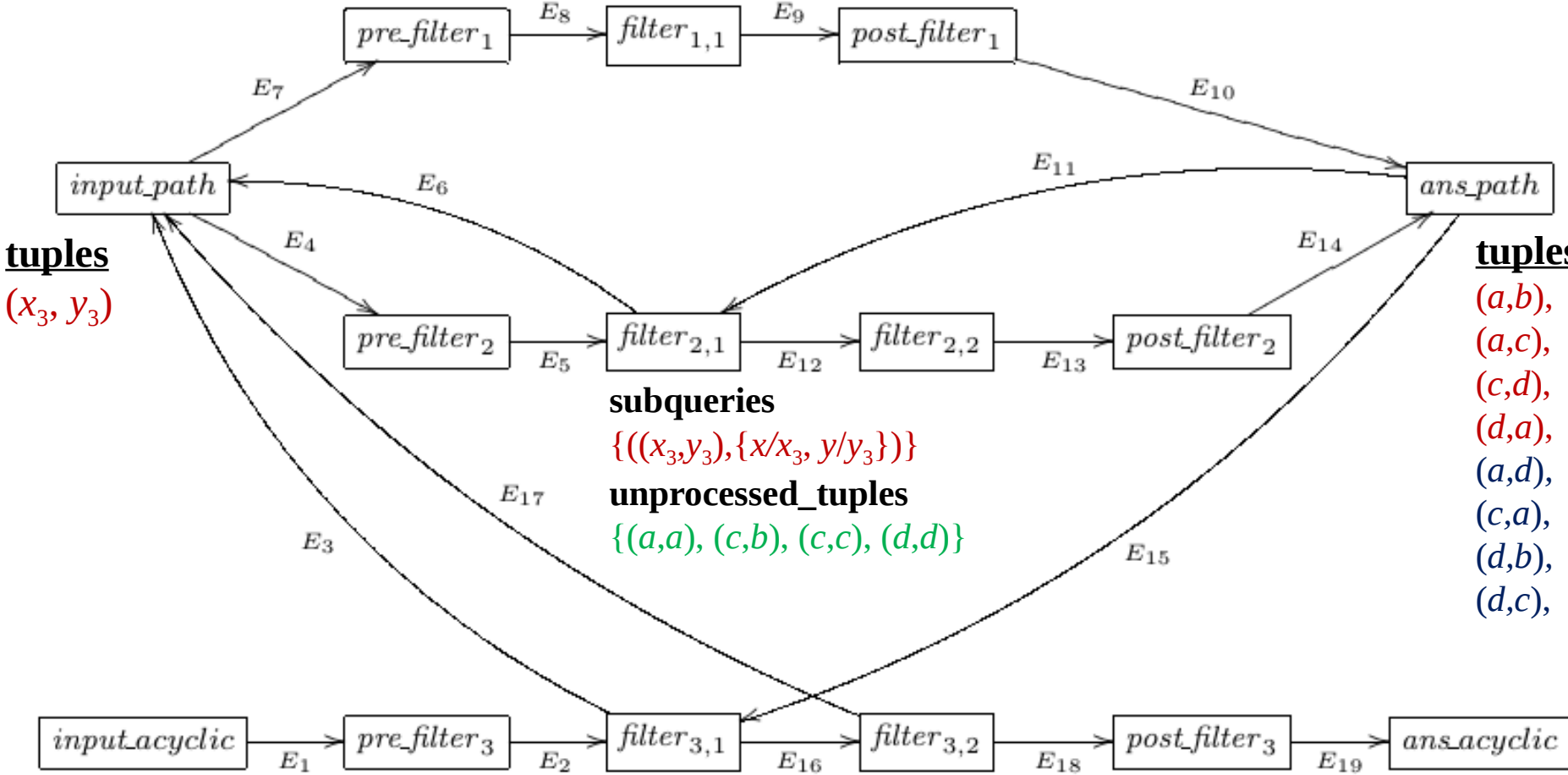
tuples
(a,b), (a,a), (a,c), (c,b), (c,d), (c,c), (d,a), (d,d) (a,d), (c,a), (d,b), (d,c),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

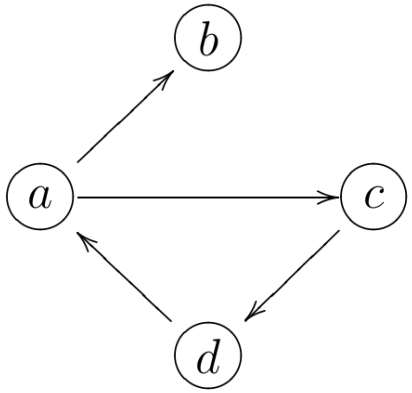
unprocessed_tuples
$\{(a,a), (c,b), (c,c), (d,d)\}$

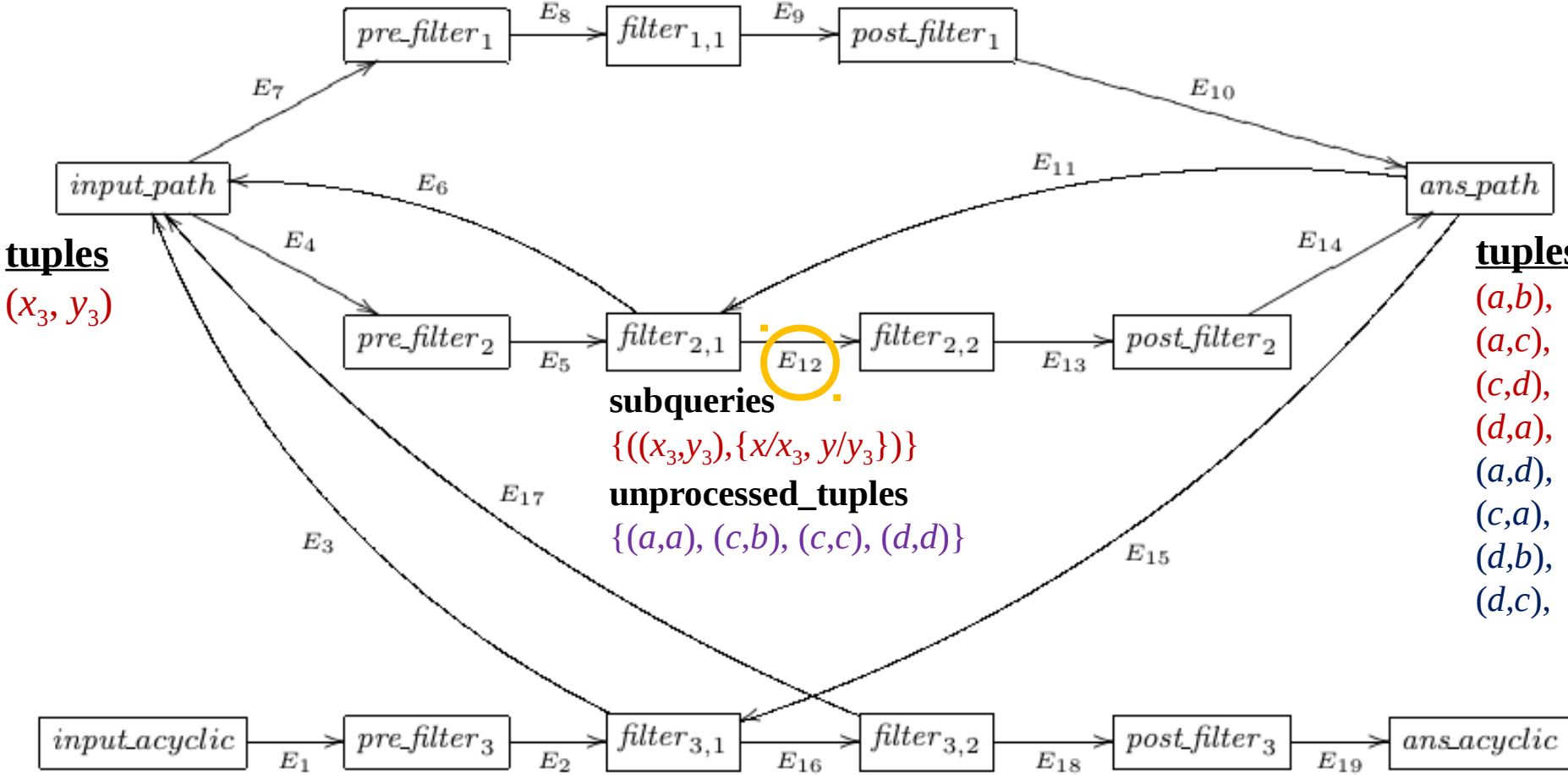
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

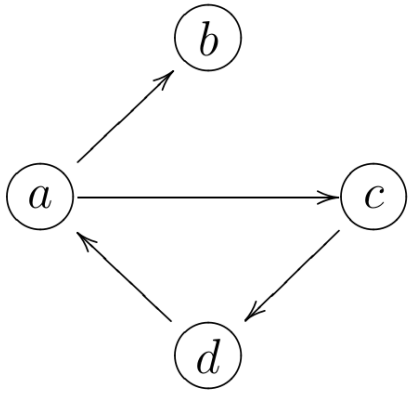
unprocessed_tuples
$\{(a,a), (c,b), (c,c), (d,d)\}$

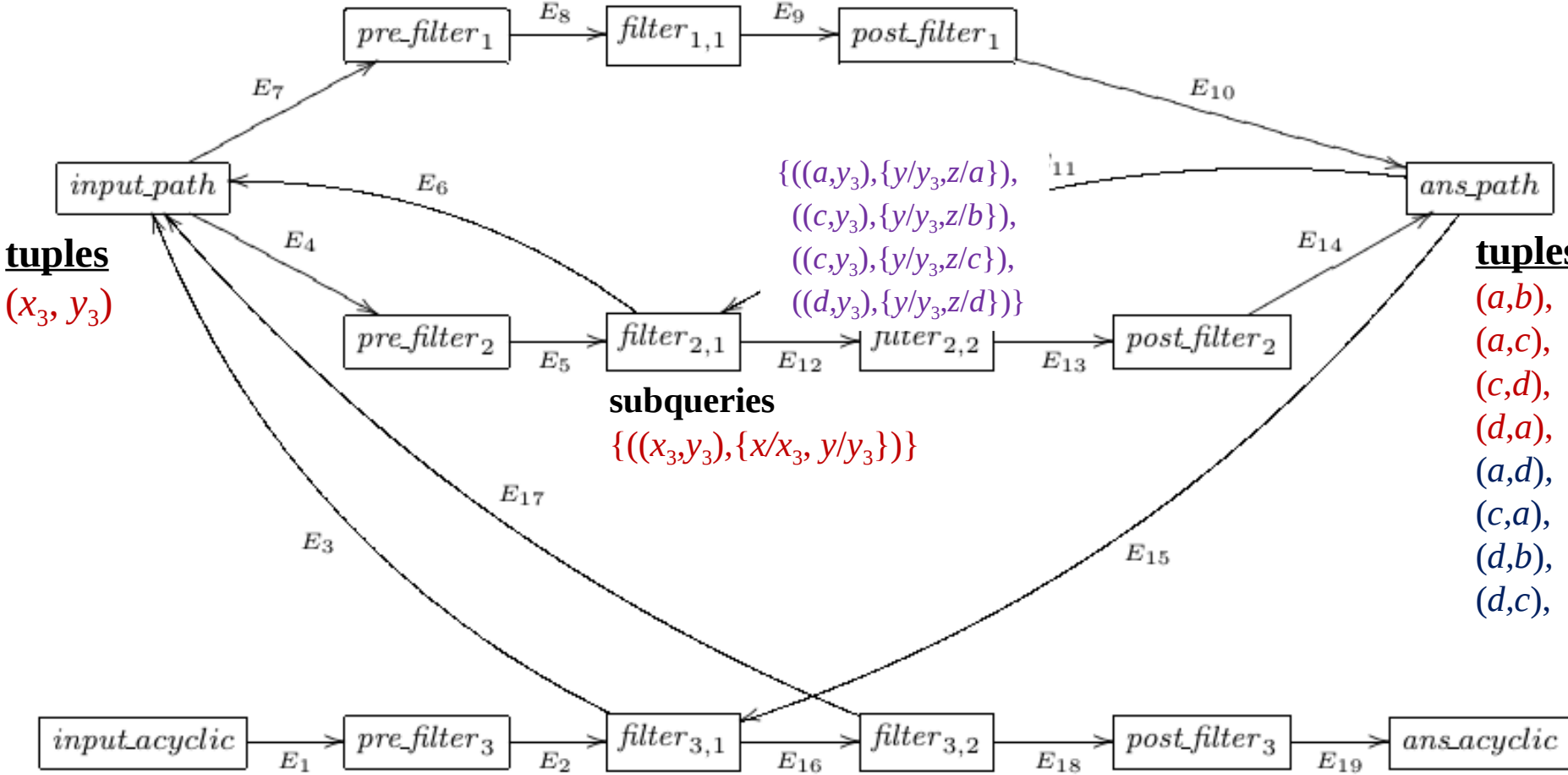
tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

$\{((a,y_3),\{y/y_3,z/a\}),$
$((c,y_3),\{y/y_3,z/b\}),$
$((c,y_3),\{y/y_3,z/c\}),$
$((d,y_3),\{y/y_3,z/d\})\}$

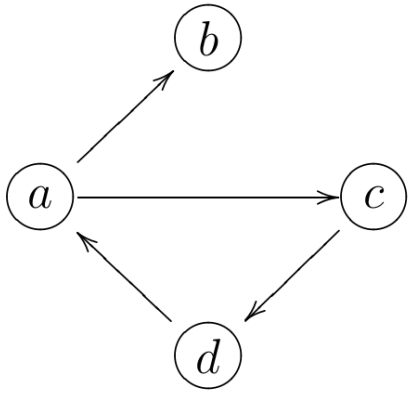
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

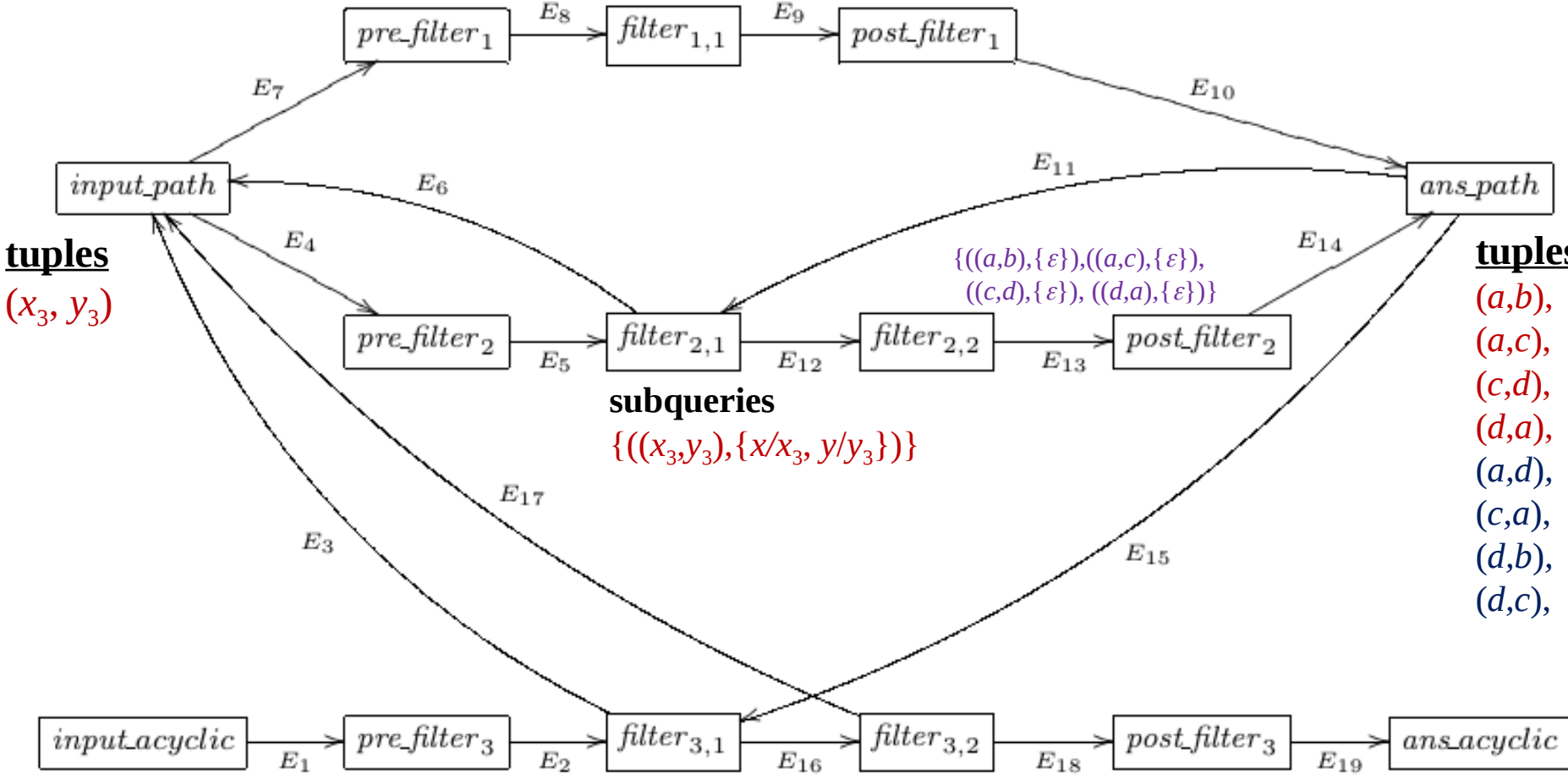
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

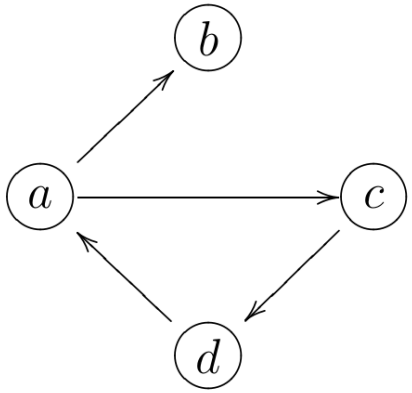
$\{((a,b),\{\varepsilon\}),((a,c),\{\varepsilon\}),$
$((c,d),\{\varepsilon\}), ((d,a),\{\varepsilon\})\}$

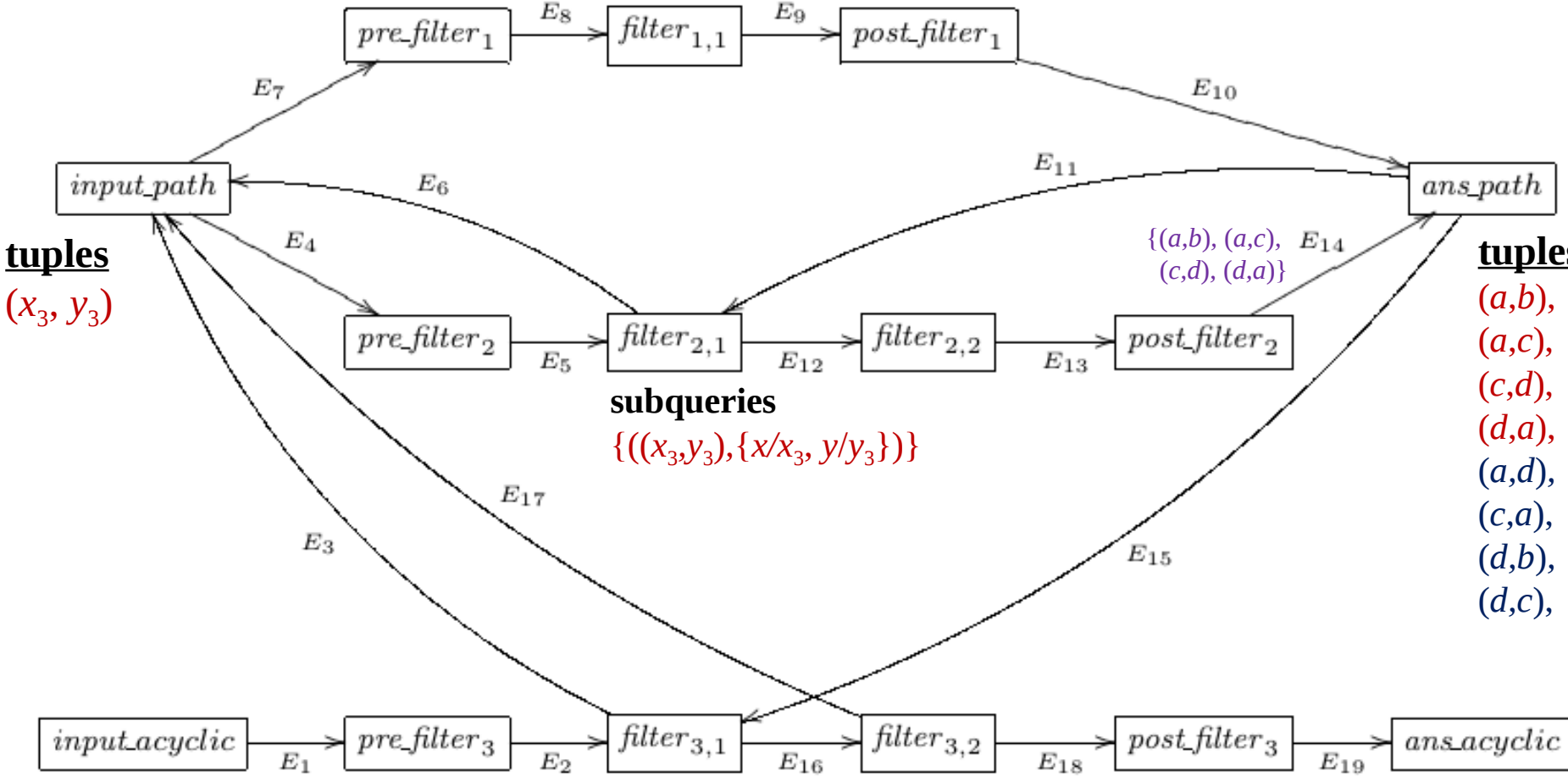
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

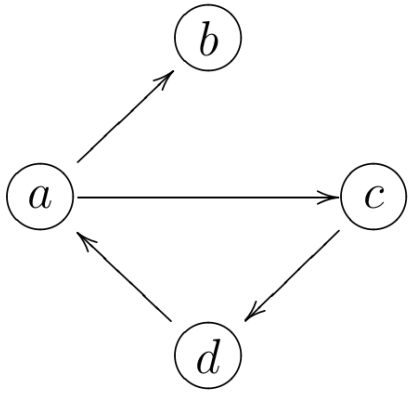
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

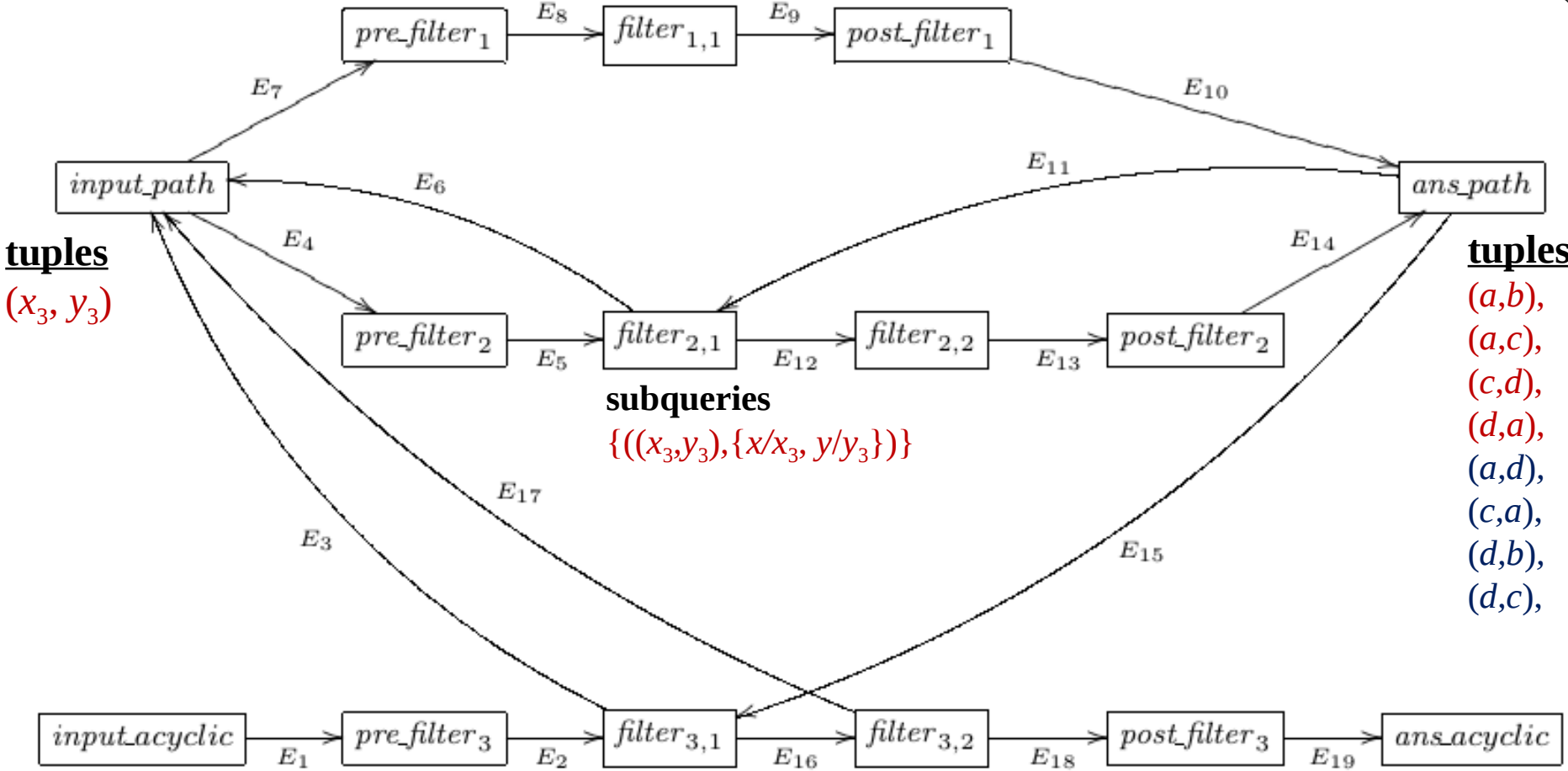
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

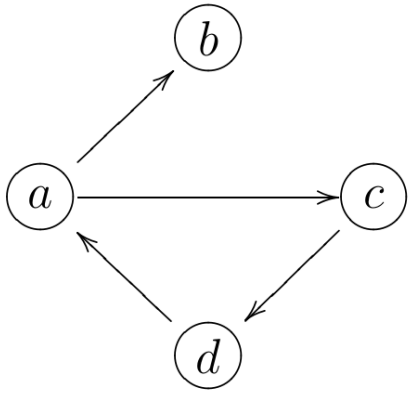
No tuples is added to *tuples*(*ans_path*)

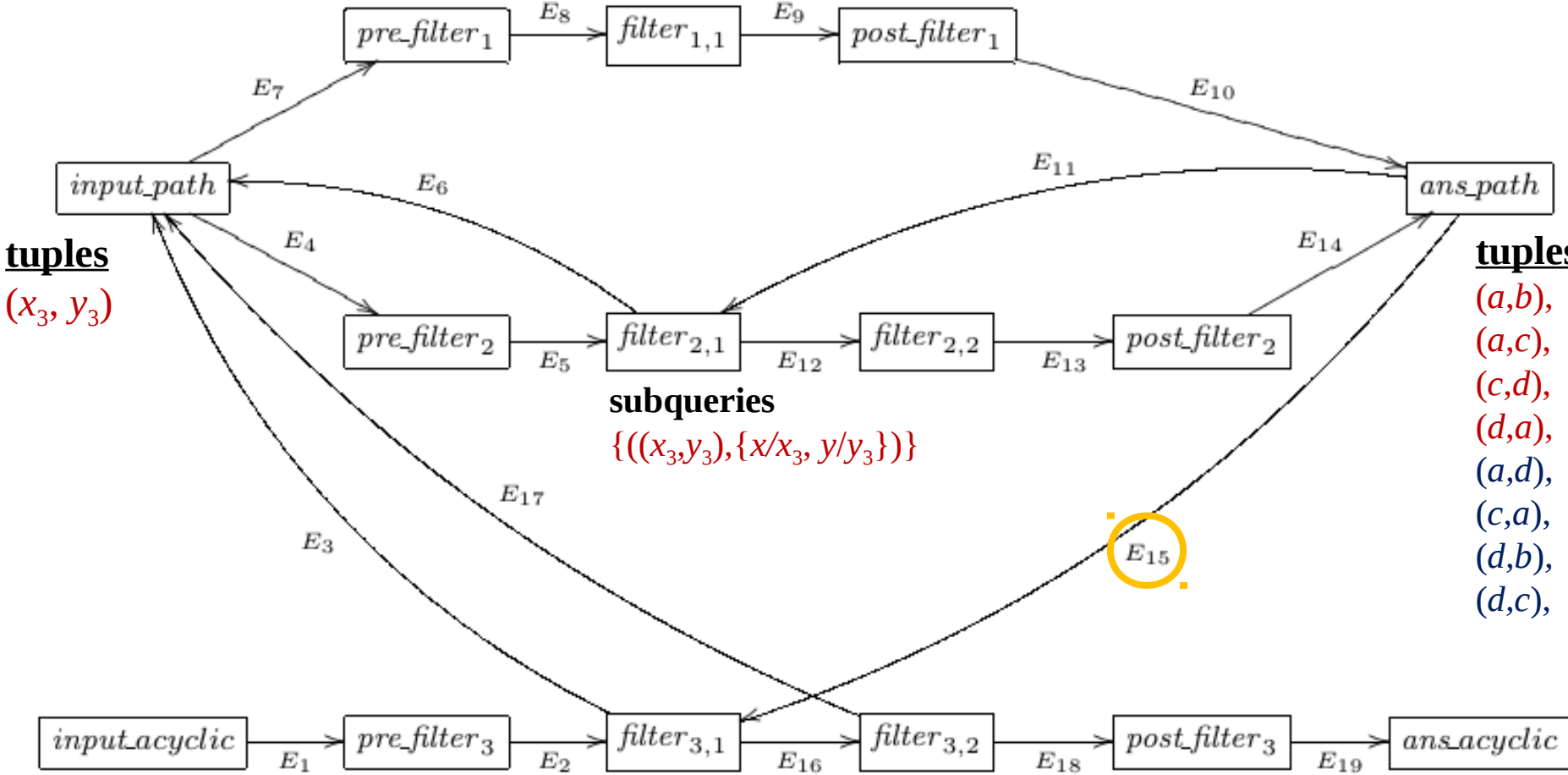
tuples
(*a*,*b*), (*a*,*a*),
(*a*,*c*), (*c*,*b*),
(*c*,*d*), (*c*,*c*),
(*d*,*a*), (*d*,*d*)
(*a*,*d*),
(*c*,*a*),
(*d*,*b*),
(*d*,*c*),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

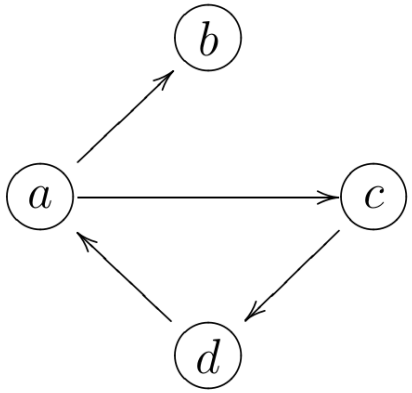
subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

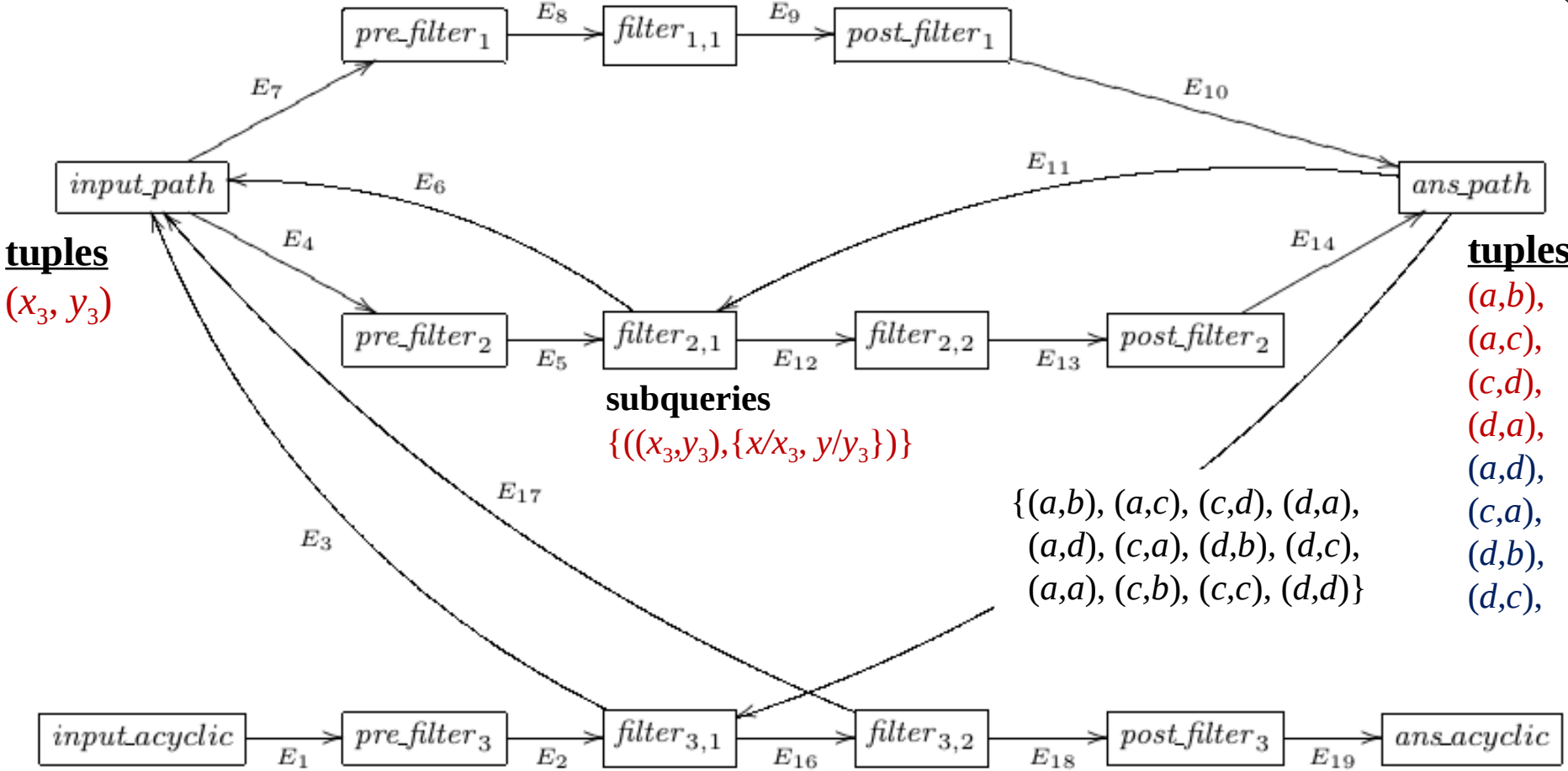
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

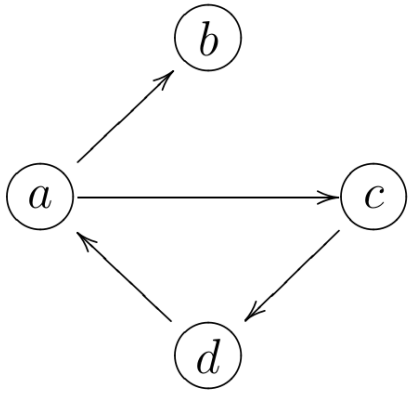
tuples
(x_1, y_1)

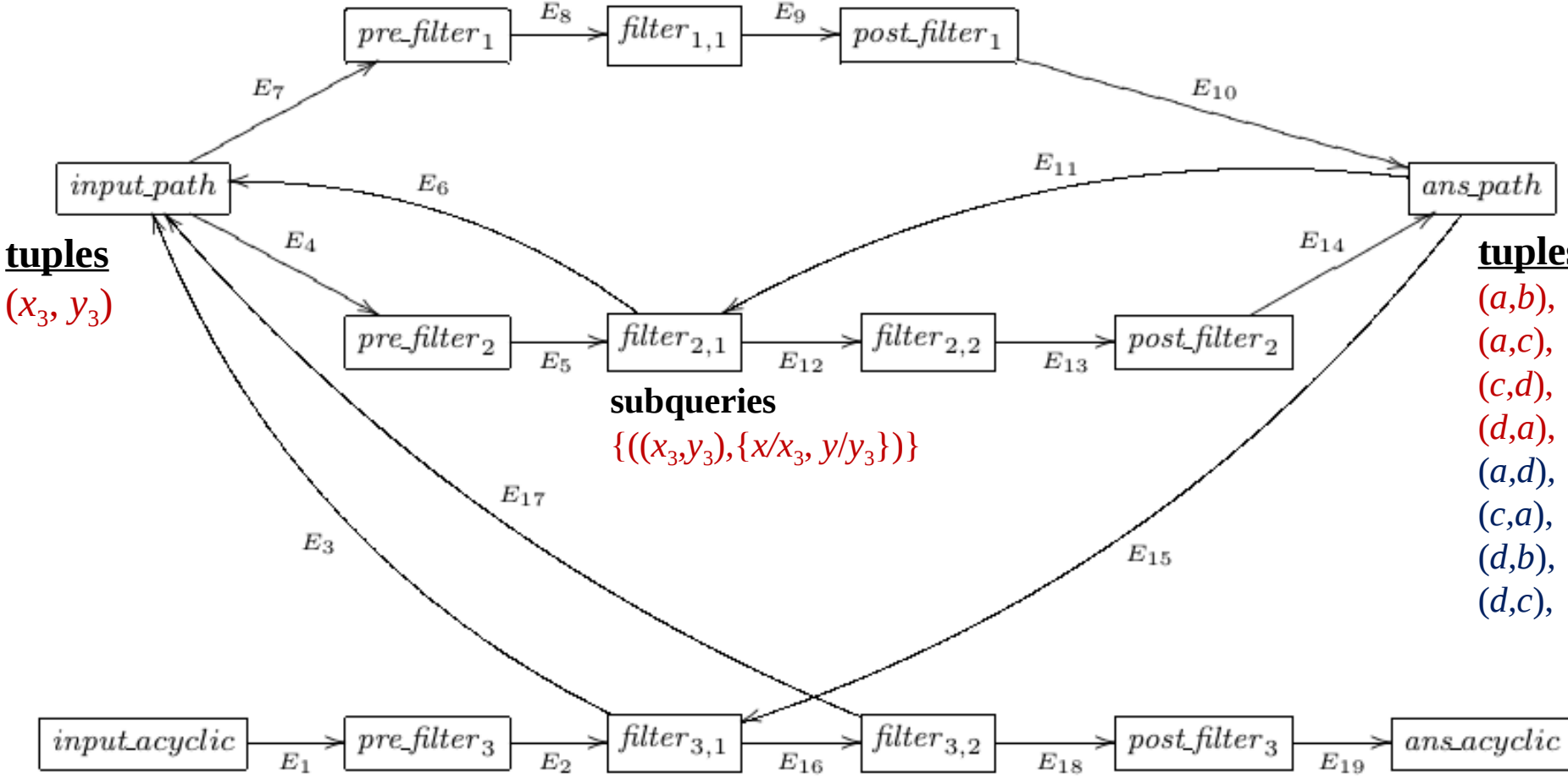
subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

$\{(a,b), (a,c), (c,d), (d,a),$
$(a,d), (c,a), (d,b), (d,c),$
$(a,a), (c,b), (c,c), (d,d)\}$

tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

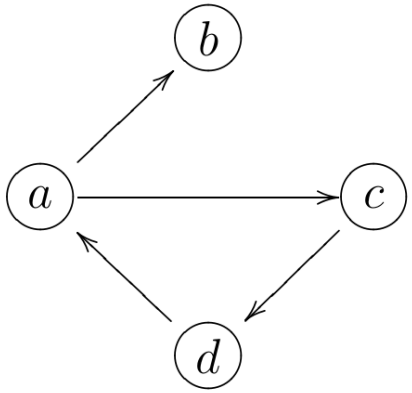
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

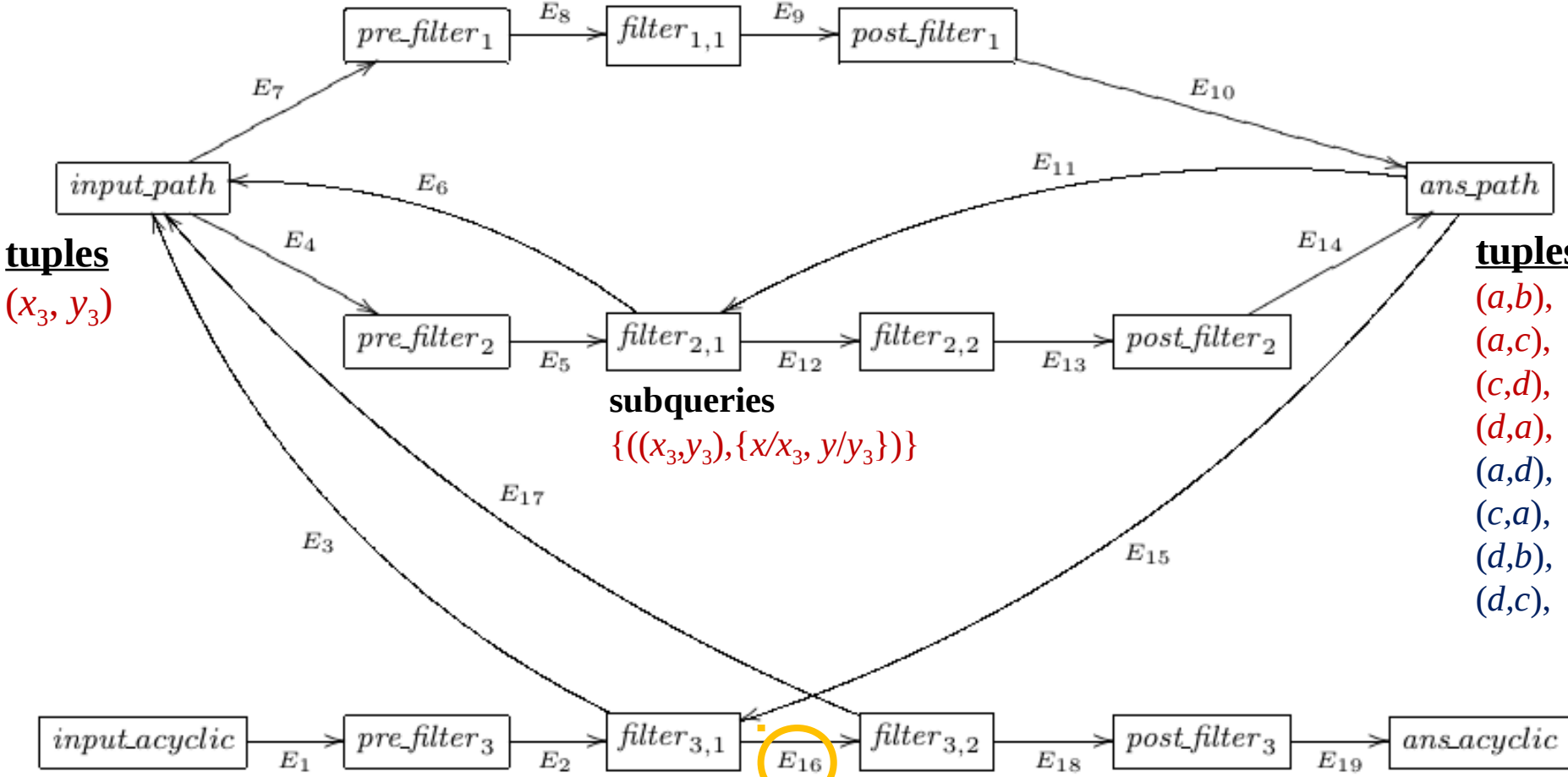
tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

unprocessed_tuples
$\{(a,b), (a,c), (c,d), (d,a), (a,d), (c,a), (d,b), (d,c), (a,a), (c,b), (c,c), (d,d)\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

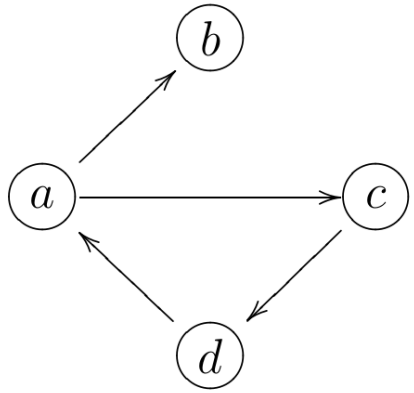
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

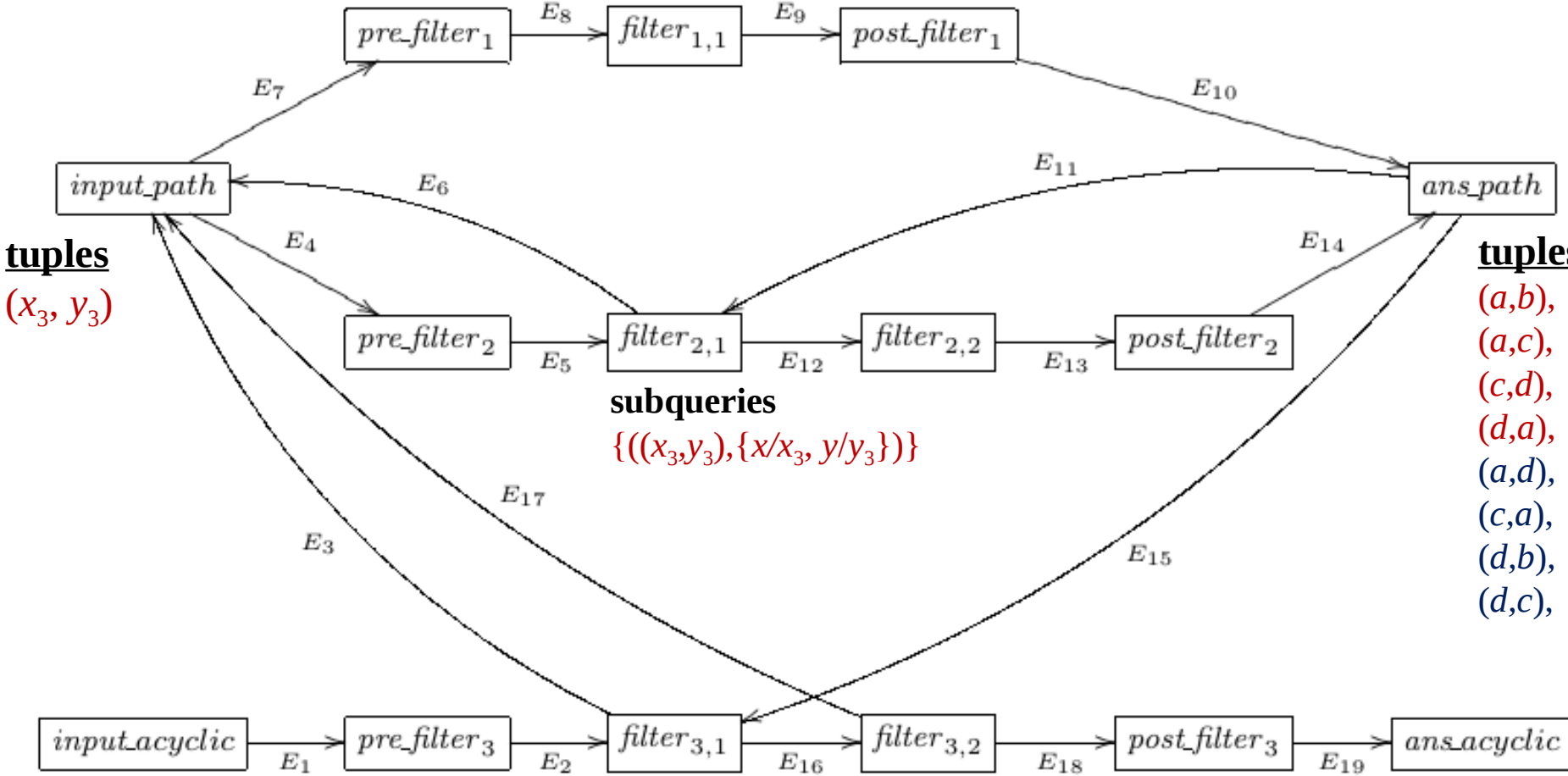
tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

unprocessed_tuples
$\{(a,b), (a,c), (c,d), (d,a), (a,d), (c,a), (d,b), (d,c), (a,a), (c,b), (c,c), (d,d)\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

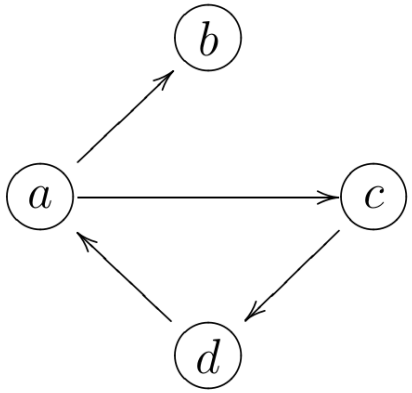
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

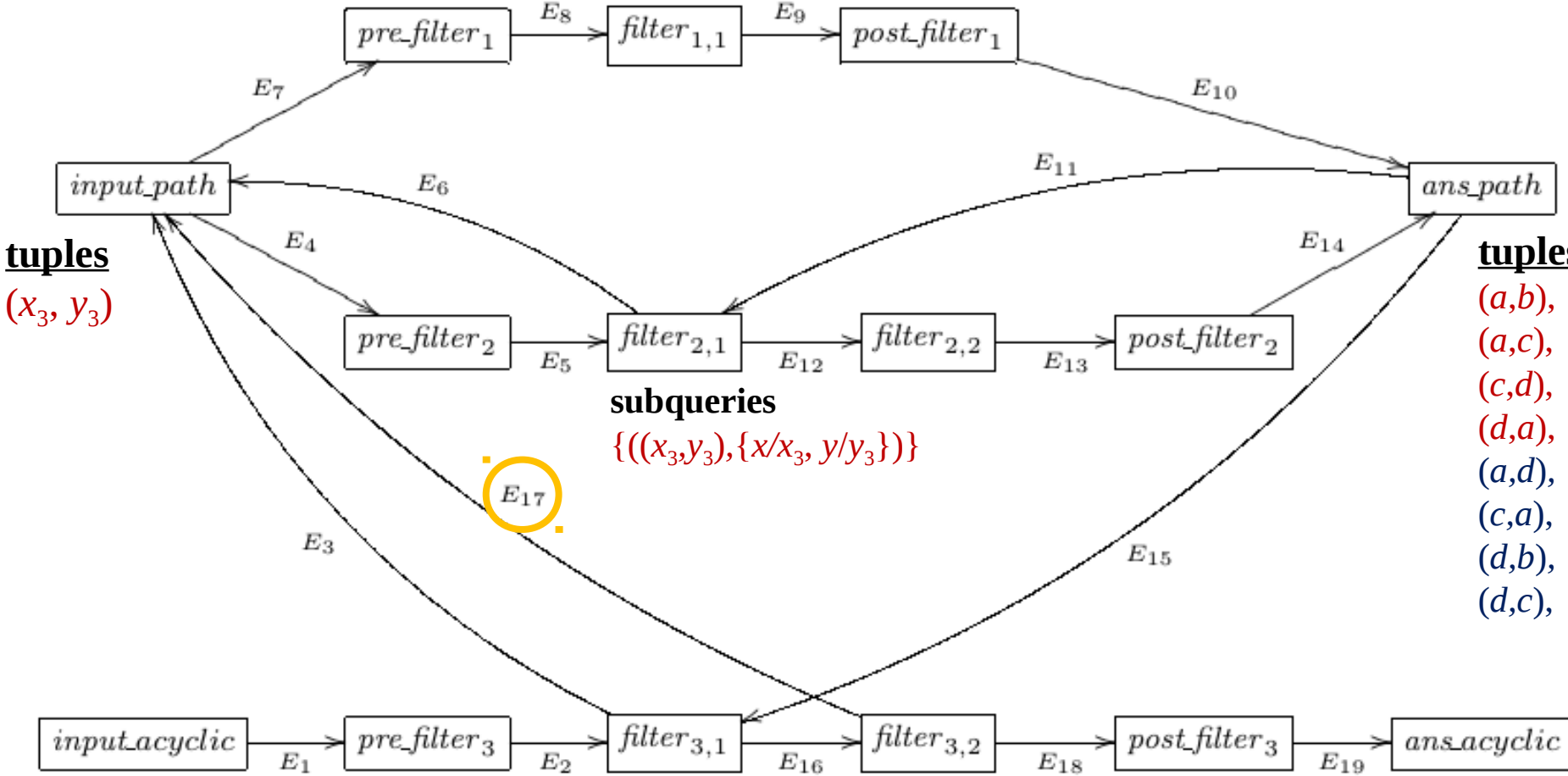
tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

subqueries
$\{((a,b),\{x/a,y/b\}), ((a,c),\{x/a,y/c\}), ((c,d),\{x/c,y/d\}), ((d,a),\{x/d,y/a\}),$
$((a,d),\{x/a,y/d\}), ((c,a),\{x/c,y/a\}), ((d,b),\{x/d,y/b\}), ((d,c),\{x/d,y/c\}),$
$((a,a),\{x/a,y/a\}), ((c,b),\{x/c,y/b\}), ((c,c),\{x/c,y/c\}), ((d,d),\{x/d,y/d\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

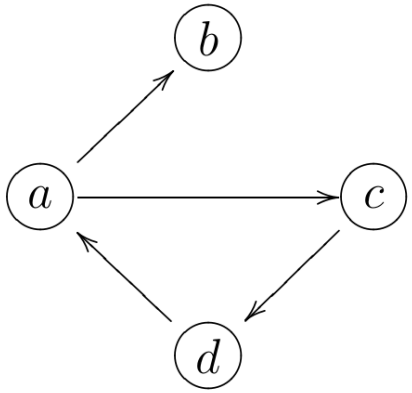
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

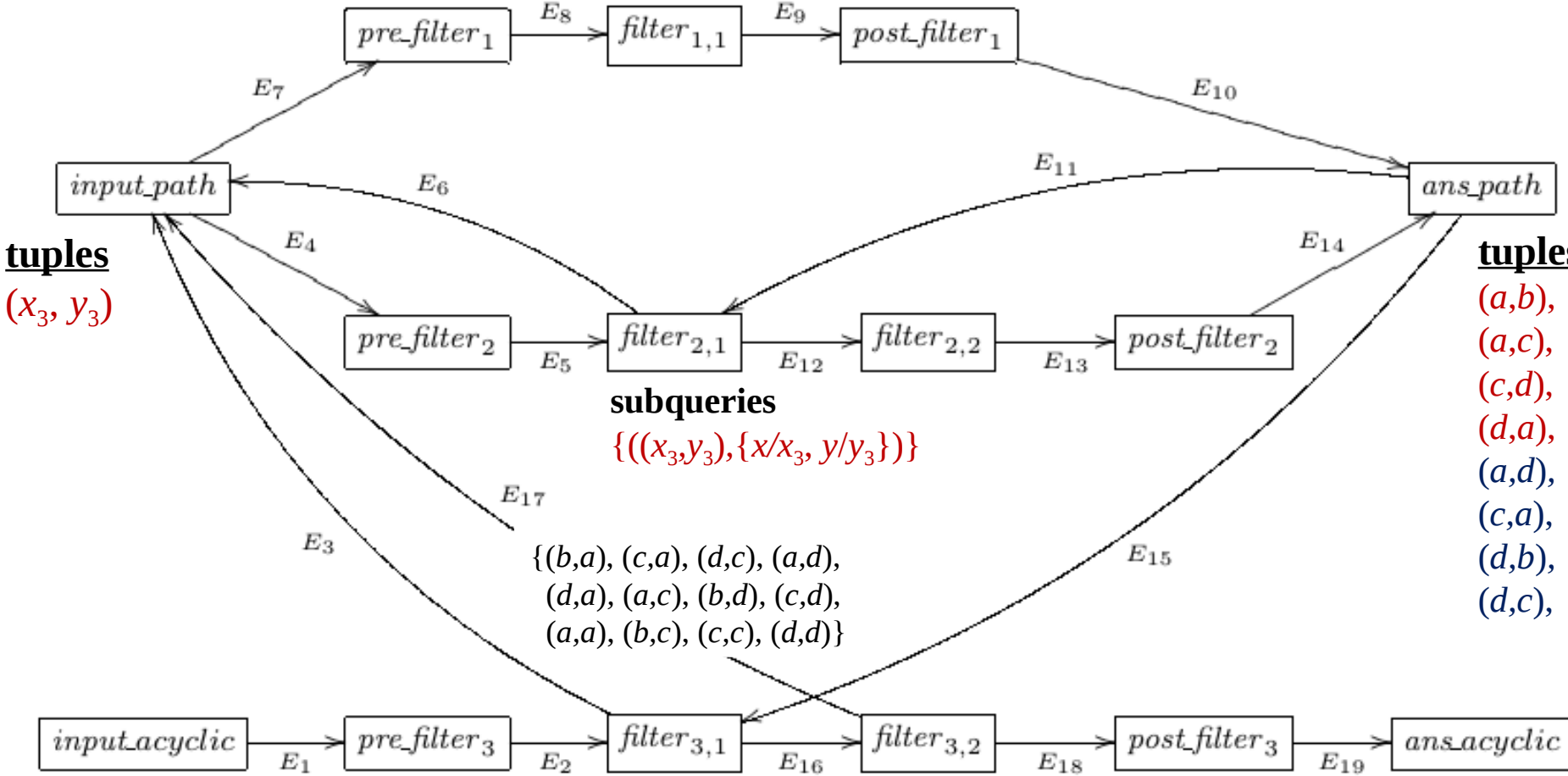
tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

subqueries
$\{((a,b),\{x/a,y/b\}), ((a,c),\{x/a,y/c\}), ((c,d),\{x/c,y/d\}), ((d,a),\{x/d,y/a\}),$
$((a,d),\{x/a,y/d\}), ((c,a),\{x/c,y/a\}), ((d,b),\{x/d,y/b\}), ((d,c),\{x/d,y/c\}),$
$((a,a),\{x/a,y/a\}), ((c,b),\{x/c,y/b\}), ((c,c),\{x/c,y/c\}), ((d,d),\{x/d,y/d\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

$\{(b,a), (c,a), (d,c), (a,d),$
$(d,a), (a,c), (b,d), (c,d),$
$(a,a), (b,c), (c,c), (d,d)\}$

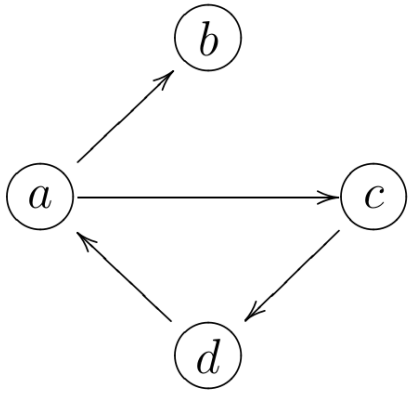
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

tuples
(x_1, y_1)

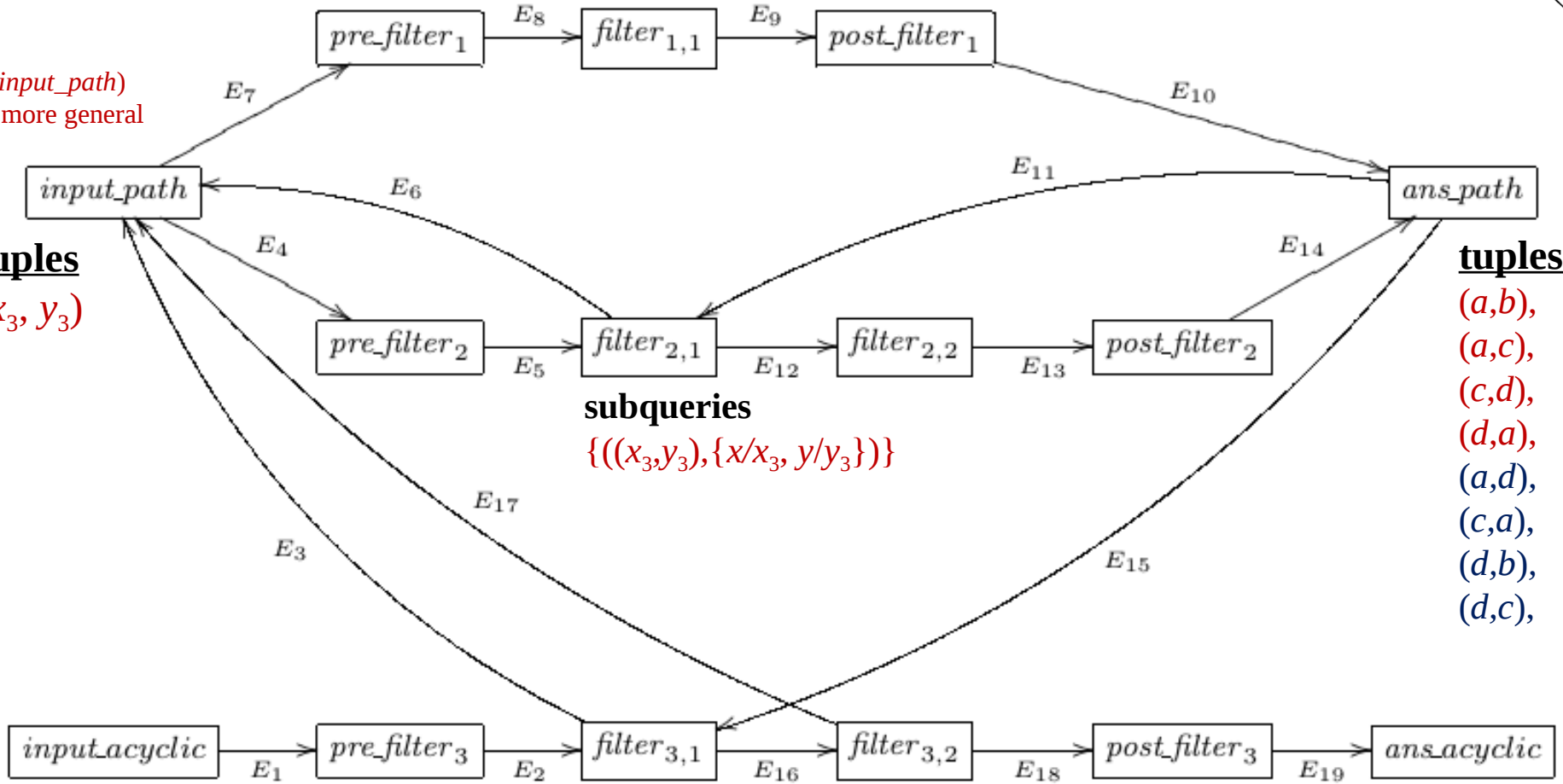
subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

subqueries
$\{((a,b),\{x/a,y/b\}), ((a,c),\{x/a,y/c\}), ((c,d),\{x/c,y/d\}), ((d,a),\{x/d,y/a\}),$
$((a,d),\{x/a,y/d\}), ((c,a),\{x/c,y/a\}), ((d,b),\{x/d,y/b\}), ((d,c),\{x/d,y/c\}),$
$((a,a),\{x/a,y/a\}), ((c,b),\{x/c,y/b\}), ((c,c),\{x/c,y/c\}), ((d,d),\{x/d,y/d\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$



no tuple is added to tuples(*input_path*) because the tuple (x_3, y_3) is more general than any other tuples



tuples
(x_3, y_3)

subqueries
{((x_3,y_3),{x/x_3, y/y_3})}

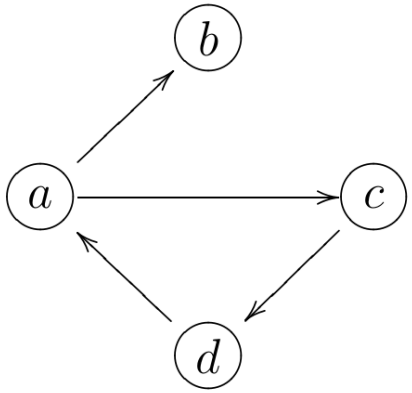
tuples
(*a*,*b*), (*a*,*a*),
(*a*,*c*), (*c*,*b*),
(*c*,*d*), (*c*,*c*),
(*d*,*a*), (*d*,*d*)
(*a*,*d*),
(*c*,*a*),
(*d*,*b*),
(*d*,*c*),

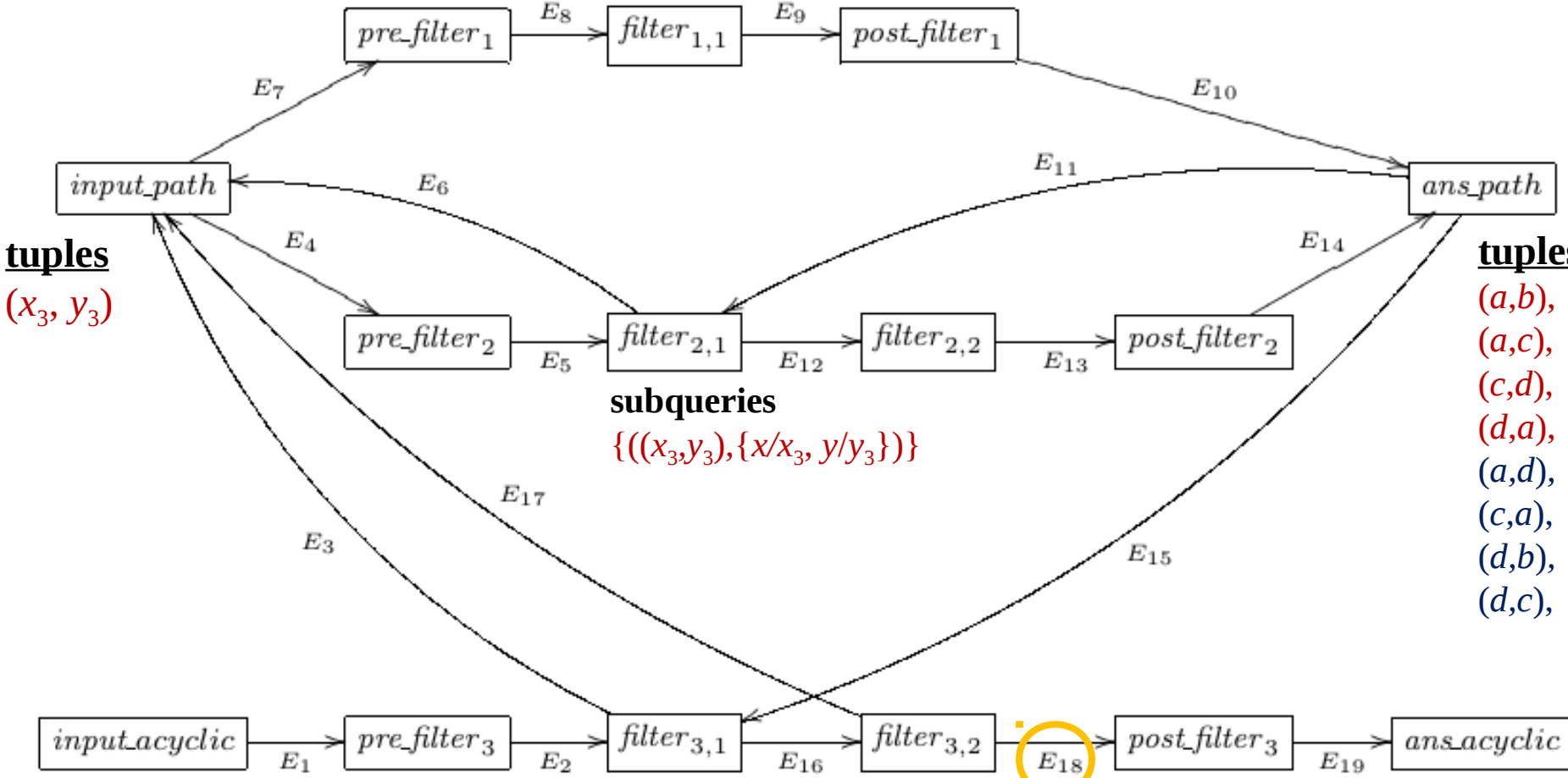
tuples
(x_1, y_1)

subqueries
{((x_1,y_1),{x/x_1, y/y_1})}

subqueries
{((*a*,*b*),{*x*/*a*,*y*/*b*}), ((*a*,*c*),{*x*/*a*,*y*/*c*}), ((*c*,*d*),{*x*/*c*,*y*/*d*}), ((*d*,*a*),{*x*/*d*,*y*/*a*}),
((*a*,*d*),{*x*/*a*,*y*/*d*}), ((*c*,*a*),{*x*/*c*,*y*/*a*}), ((*d*,*b*),{*x*/*d*,*y*/*b*}), ((*d*,*c*),{*x*/*d*,*y*/*c*}),
((*a*,*a*),{*x*/*a*,*y*/*a*}), ((*c*,*b*),{*x*/*c*,*y*/*b*}), ((*c*,*c*),{*x*/*c*,*y*/*c*}), ((*d*,*d*),{*x*/*d*,*y*/*d*})}

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

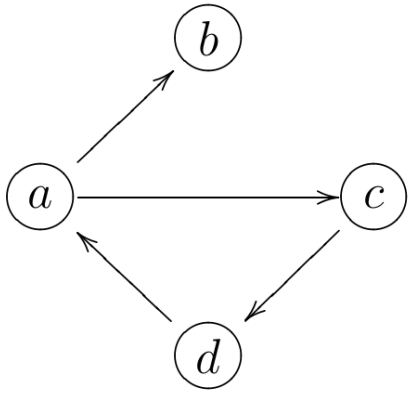
tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

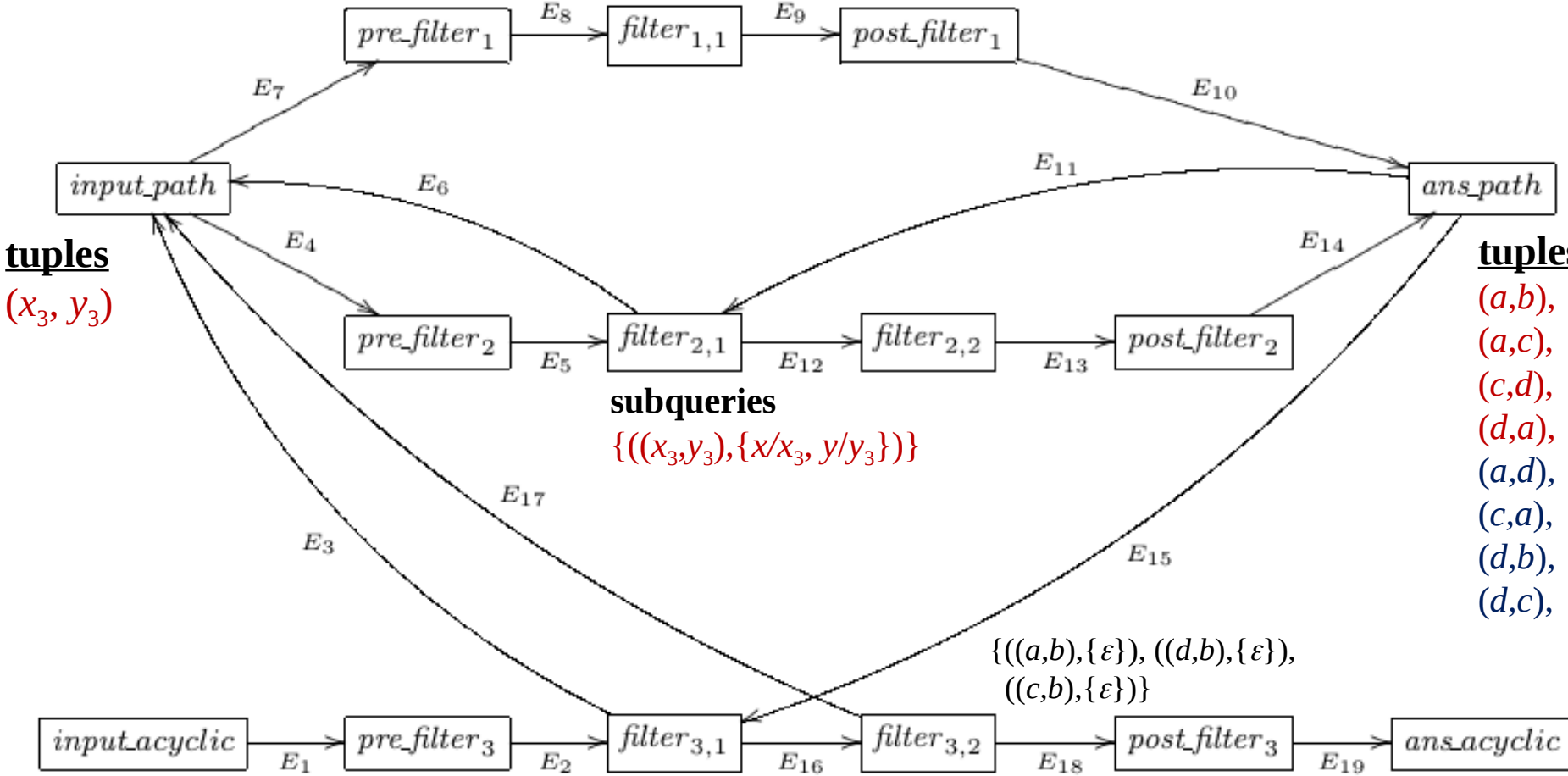
tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

subqueries
$\{((a,b),\{x/a,y/b\}), ((a,c),\{x/a,y/c\}), ((c,d),\{x/c,y/d\}), ((d,a),\{x/d,y/a\}),$
$((a,d),\{x/a,y/d\}), ((c,a),\{x/c,y/a\}), ((d,b),\{x/d,y/b\}), ((d,c),\{x/d,y/c\}),$
$((a,a),\{x/a,y/a\}), ((c,b),\{x/c,y/b\}), ((c,c),\{x/c,y/c\}), ((d,d),\{x/d,y/d\})\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

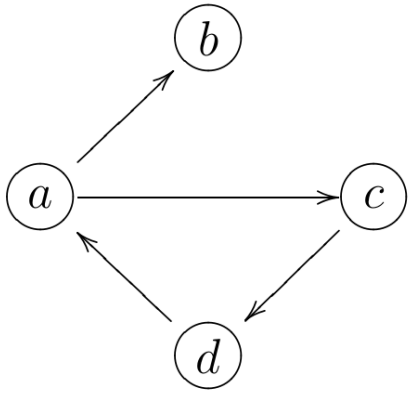
$\{((a,b),\{\varepsilon\}), ((d,b),\{\varepsilon\}),$
$((c,b),\{\varepsilon\})\}$

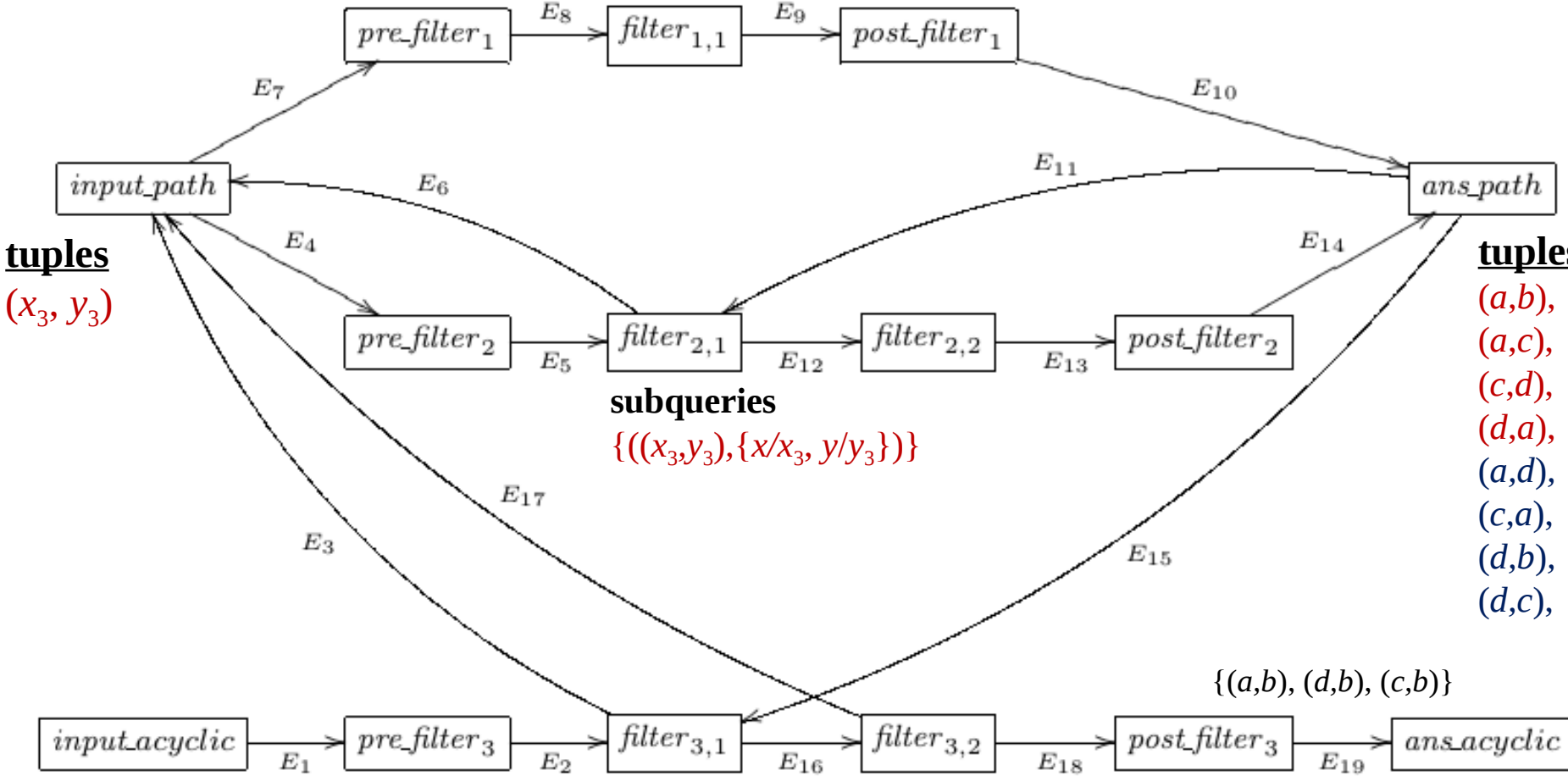
tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

subqueries
$\{((a,b),\{x/a,y/b\}), \ldots\}$

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

tuples
(*a*,*b*), (*a*,*a*),
(*a*,*c*), (*c*,*b*),
(*c*,*d*), (*c*,*c*),
(*d*,*a*), (*d*,*d*)
(*a*,*d*),
(*c*,*a*),
(*d*,*b*),
(*d*,*c*),

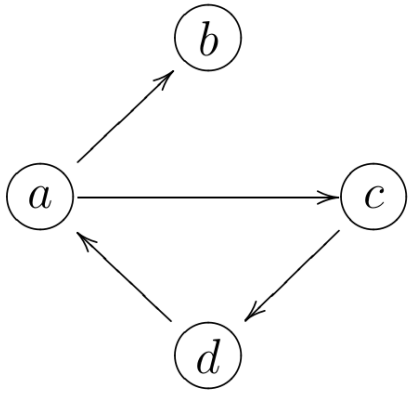
{(*a*,*b*), (*d*,*b*), (*c*,*b*)}

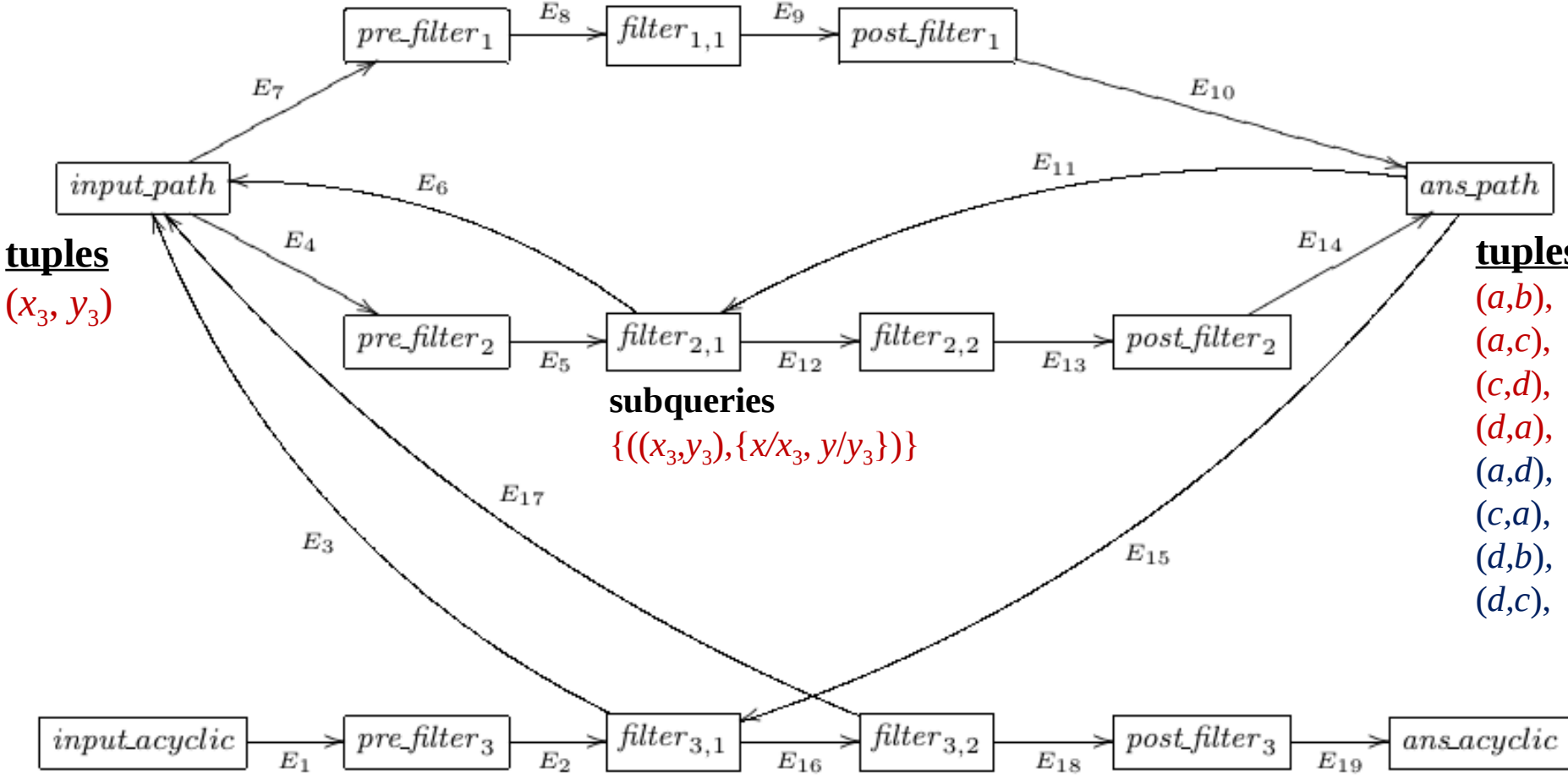
tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

subqueries
{((*a*,*b*),{*x*/*a*,*y*/*b*}), …}

$$path(x, y) \leftarrow edge(x, y)$$
$$path(x, y) \leftarrow path(x, z), edge(z, y)$$
$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x).$$





tuples
(x_3, y_3)

subqueries
$\{((x_3,y_3),\{x/x_3, y/y_3\})\}$

tuples
(a,b), (a,a),
(a,c), (c,b),
(c,d), (c,c),
(d,a), (d,d)
(a,d),
(c,a),
(d,b),
(d,c),

tuples
(x_1, y_1)

subqueries
$\{((x_1,y_1),\{x/x_1, y/y_1\})\}$

subqueries
$\{((a,b),\{x/a,y/b\}), \ldots\}$

tuples
(a,b),
(d,b),
(c,b)

At this point, no edge in the net is active.
The algorithm terminates and returns the set *tuples*(*ans_acyclic*) = {(*a*,*b*), (*d*,*b*), (*c*,*b*)}.

Computing and Informatics, Vol. 38, 2019, 57–84, doi: 10.31577/cai_2019_1_57

VARIABLE NEIGHBORHOOD SEARCH APPROACH FOR SOLVING ROMAN AND WEAK ROMAN DOMINATION PROBLEMS ON GRAPHS

Marija Ivanović

Faculty of Mathematics
University of Belgrade
Studentski trg 16/IV
11 000 Belgrade, Serbia
e-mail: marijai@math.rs

Dragan Urošević

Mathematical Institute, SANU
Kneza Mihaila 36
11 000 Belgrade, Serbia
e-mail: draganu@mi.sanu.ac.rs

Abstract. In this paper Roman and weak Roman domination problems on graphs are considered. Given that both problems are NP hard, a new heuristic approach, based on a Variable Neighborhood Search (VNS), is presented. The presented algorithm is tested on instances known from the literature, with up to 600 vertices. The VNS approach is justified since it was able to achieve an optimal solution value on the majority of instances where the optimal solution value is known. Also, for the majority of instances where optimization solvers found a solution value but were unable to prove it to be optimal, the VNS algorithm achieves an even better solution value.

Keywords: Roman domination in graphs, weak Roman domination in graphs, combinatorial optimization, metaheuristic, variable neighborhood search

Mathematics Subject Classification 2010: 05C69, 05C85, 90C10

1 INTRODUCTION

The Roman domination problem (RD problem) was introduced by ReVelle and Rosing [1] and Cockayne et al. [2] and can be interpreted as follows.

Assuming that any province of the Roman Empire is considered to be safe if there is at least one legion (of maximum 2) stationed within it, the RD problem requires that every unsafe province must be adjacent to a province with at least two legions stationed within it and the total number of stationed legions within all provinces of the Roman Empire is minimal.

In a graph terminology, let $G = (V, E)$ be a simple undirected graph with a vertex set V such that each vertex $u \in V$ represents a province of the Roman Empire and each edge, $e \in E$, represents an existing connection between two provinces. Let f be a function $f : V \to \{0, 1, 2\}$ and let the weight of the vertex u, denoted by $f(u)$, represent the number of legions stationed at province u. Further, let the weight of the function f be calculated by a formula $\sum_{v \in V} f(v)$. Function f is called a Roman dominating function (RD function) if every vertex u such that $f(u) = 0$ is adjacent to a vertex v such that $f(v) = 2$. The Roman domination problem is to find an RD function f of a graph G with the smallest weight. The smallest weight of the RD function f, denoted by $\gamma_R(G)$, is known as the Roman domination number.

We illustrate the Roman domination problem in the example below.

Example 1. Let us assume that the Roman Empire can be described by a graph $G = (V, E)$ as it is presented below, in Figure 1.

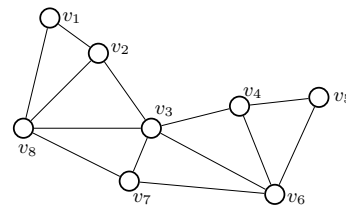


Figure 1: Graph $G = (V, E)$

The optimal number of legions necessary to defend the given graph is 4, provinces represented by vertices v_1 and v_5 are with one stationed legion, province represented by vertex v_3 is with two stationed legions and all other provinces are without stationed legions. With the given schedule, vertices v_1, v_3 and v_5 are defended because they have at least one legion stationed within it, while v_2, v_4, v_6, v_7 and v_8 are defended since they are in the neighborhood of the vertex v_3, which is with two stationed legions. The optimal solution to the proposed problem is illustrated in Figure 2, where vertices are marked by black squares if they are representing provinces with two stationed legions, marked by red circles if they are representing provinces with one stationed legion, and marked by white circles if they are representing provinces without stationed legions.

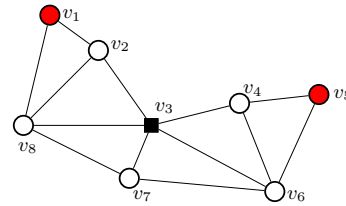


Figure 2: Illustrated solution of the RD problem on a graph G defined in the Example 1

In order to reduce the number of legions necessary to defend the Roman Empire against a single attack, Henning and Hedetniemi [3] introduced the weak Roman domination problem (WRD problem) as a variant of the RD problem. First, they assumed that every province of the Roman Empire is safe if there is at least one legion stationed within it and every unsafe province is defended if it is adjacent to a safe province. Then they required that for every unsafe province there exists at least one adjacent safe province whose legion could move and protect it in case it is attacked, such that this particular legion movement does not affect the Empire's safety, i.e., all provinces are considered to be defended before and after the movement.

Similarly as for the RD problem, for a graph $G = (V, E)$ and a function $f : V \to \{0, 1, 2\}$, every vertex with positive weight is considered to be defended, and a vertex u with property $f(u) = 0$ is considered to be defended if it is adjacent to a vertex $v \in V$ with positive weight. A function f is called a weak Roman dominating function (WRD function) on a graph G if every vertex u with property $f(u) = 0$ is adjacent to a vertex v with property $f(v) > 0$ and, with respect to the function f', $f' : V \to \{0, 1, 2\}$ defined by $f'(u) = 1$, $f'(v) = f(v) - 1$ and $f'(w) = f(w)$, $w \in V \setminus \{u, v\}$, all vertices are defended. The problem of finding the WRD function f with the minimal weight for a given graph G is referred to as the weak Roman domination problem (WRD problem). The minimum weight of the WRD function f, denoted by $\gamma_r(G)$, represents the weak Roman domination number.

We illustrate the weak Roman domination problem in the example below.

Example 2. Let us assume that the Roman Empire can be described by the graph $G = (V, E)$ presented on Figure 1. The optimal solution value for the WRD problem on the given graph is 3. Legions are stationed such that provinces represented by vertices v_1, v_5 and v_7 are with one stationed legion while all other provinces are without stationed legions, see Figure 3 (vertices are marked by red circles if they are representing provinces with one stationed legion and marked by white circles if they are representing provinces without stationed legions).

With the given strategy, in case of an attack, provinces represented by vertices v_2 and v_8 are defended by the legion stationed at the province represented by the vertex v_1. In case of attack, movements of legion stationed at province v_1 to province v_2 or to v_8 does not affect Empire's safety. Similarly, provinces v_4 and v_6 are defended

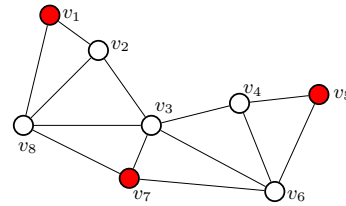


Figure 3: Illustrated solution of the WRD problem for a graph G defined in the Example 2

by the legion from province v_5, province v_3 is defended by the legion from province v_7, etc.

Ivanović [6] showed that neither the CPLEX nor the Gurobi optimization solvers were able to solve the WRD problem on a huge number of instances with more than 100 vertices. Since there is only one algorithm for solving the WRD problem (see [7]), which is written only for block graphs, we present a Variable Neighborhood Search solution for solving the WRD problem on any types of graphs.

We also show that the same algorithm can be applied to the RD problem, although Burger et al. [8] showed that there are significant differences in solving these two problems (their assumption was based on the fact that the RD problem involves static configuration of legions on the vertices of G, while the WRD problem involves moving a legion between the adjacent vertices).

This paper is organized as follows. Previous work is given in Section 2. The Variable Neighborhood Search algorithm is proposed in Section 3. Computational results are summarized in Section 4.

2 PREVIOUS WORK

The Roman domination problem was introduced by Stewart [9] and ReVelle and Rossing [1]. Inspired by Stewart's paper, Cockayne et al. [2] gave some properties of the Roman domination sets. Later Henning et al. [3] introduced the WRD problem as special variant of the RD problem and observed that every RD function in a graph G is also a WRD function in G. In the same paper they proved relation $\gamma(G) \leq \gamma_r(G) \leq \gamma_R(G) \leq 2\gamma(G)$, where $\gamma(G)$ represents cardinality of the minimum dominating set on the graph G (dominating set is a set of vertices such that each of the other vertices has a neighbor in the dominating set). Relations between several different domination numbers were summarized by Chellali et al. [10].

Upper and lower bounds for γ_R for special types of graphs were determined, for instance, in [2, 11, 13, 14, 15, 16, 17]. Exact values for γ_R for paths, cycles, complete, complete n-partite and Petersen $P(n, 2)$ graphs were given in [2, 11, 15, 16, 18, 19, 20, 21, 22], while cardinal and Cartesian products of paths and cycles and lexicographic product of some graphs were given in [15, 16, 19]. Exact values of

the $\gamma_r(G)$ for paths, cycles, complete, complete n-partite, $2 \times n$ grid and web graphs and values of $\gamma_r(G)$ of corona and products of some special types of graphs were given in [3, 12, 23, 24].

The complexity of computing γ_R when restricted to interval graphs was mentioned as an open question in [2]. In the same paper it was shown that the problem of computing γ_R on trees can be solved in linear time and that it remains NP-complete even when restricted to split graphs, bipartite graphs, and planar graphs. Linear-time algorithm for computing γ_R on bounded tree-width graphs was proposed in [25]. In [20] it was shown that γ_R can be computed in linear time on interval graphs and co-graphs. In the same papers, the authors give a polynomial time algorithm for computing γ_R on AT-graphs and graphs with d-octopus. Linear-time approximation algorithm and a polynomial time approximation scheme for the RD problem on unit disk graphs was given in [22]. If we assume that the size of G is a given constant, Pavlič and Žerovnik provided algorithm for computing γ_R for polygraphs, including rota-graphs and fascia-graphs, that run in constant time in [19]. Some variants of the algorithm for solving the RD problem on a grid graph together with theoretical properties of γ_R of grid graphs were given in [13]. In [13] Currò also showed that the same algorithm can be applied to some other types of graph.

A binary programing formulations for the RD problem, which can be used for computing γ_R on arbitrary graphs by using standard optimization solvers, were provided by ReVelle and Rossing [1] and Burger et al. [4]. Burger et al. [4] also gave a binary programming formulations for the WRD problem. Recently Ivanović [6] gave another formulation for the WRD problem. Ivanović compared formulations for the WRD problem in [6], showing that neither CPLEX nor Gurobi optimization solvers were able to solve the WRD problem, regardless of the used formulation, on many instances with more than 100 vertices.

Peng [7] gave a linear time algorithm for computing γ_r on block graphs. Providing two faster algorithms, Chapelle et al. [26] broke trivial enumeration barrier of $O^*(3^n)$ for calculating $\gamma_r(G)$ (the notation $O^*(f(n))$ suppresses polynomial factors). With the first algorithm they proved that the WRD problem can be solved in $O^*(2^n)$ time needing exponential space. The second algorithm uses polynomial space and time, $O^*(2.2279^n)$.

For some special classes of graphs (interval graphs, intersection graphs, co-graphs and distance-hereditary graphs) the RD problem can be solved in linear time [15], but in the general case, the RD problem is NP-complete, [11]. Proof that the WRD problem is NP-complete, even when restricted to bipartite and chordal graphs, is given in [3].

Now, since both the Roman and the weak Roman domination problems are NP-complete problems, creating a heuristic that could be successful in finding an optimal solution value, providing legions schedule as well, represents a challenge.

Therefore, in [13] a genetic algorithm for solving the RD problem was proposed by Currò, and that was the only heuristic written for any type of Roman domination problem known to the authors. In the mentioned paper, the author proposes a set

of instances on random generated graphs which will be used in experimental results of this paper.

In the next section we propose the Variable Neighborhood Search algorithm for solving both the Roman and the weak Roman domination problems on graphs. The VNS heuristic is chosen because it was previously proven to be successful for some problems on graphs, for example [27, 28].

3 VARIABLE NEIGHBORHOOD SEARCH APPROACH FOR SOLVING ROMAN AND WEAK ROMAN DOMINATION PROBLEMS

The Variable Neighborhood Search (VNS) is a heuristic method, which starts from some point from the search space, explores its neighborhoods, then changes the starting point through some search procedures such that it moves to another point of the search space, explores its neighborhoods, and repeats the whole procedure in order to find a better solution. The VNS heuristic was proposed by Mladenović [29] and later studied by Mladenović and Hansen [30] and Hansen and Mladenović in [31].

With respect to the problems' definitions, let us assume that all Roman provinces are represented by a set of vertices V, $n = |V|$, and all existing roads by the set of edges $E = \{e = (i, j), \quad i, j \in V,\ i \text{ and } j \text{ are connected}\}$, $m = |E|$, of some simple undirected graph $G = (V, E)$. Given that graph G is undirected, we will say that $e = (i, j) \in E$ implies $(j, i) \in E$. Moreover, for every vertex $i \in V$ let the set of all vertices adjacent to the vertex i be marked by N_i. Furthermore, let us assume that each province is represented by a number $i = 1, \ldots, n$, and the number of legions stationed within a province i is represented by value x_i. Vector $X = (x_1, \ldots, x_n)$ of values x_i, $i = 1, \ldots, n$, is a feasible solution to the RD problem (WRD problem) if f, $f : V \to \{0, 1, 2\}$ defined by

$$f(i) = x_i, \quad i \in V \tag{1}$$

is a Roman domination function (weak Roman domination function).

Given that a feasible solution to the WRD problem does not have to be a feasible solution to the RD problem, we define a function *feasibleSolution*$(X, problem)$ which checks if X is a feasible solution for the *problem* $\in$ {RD, WRD}.

In order to check if vector X is a feasible solution to the RD problem, for every element x_i $(i = 1, \ldots, n)$ *feasibleSolution(X, RD)* checks if x_i is a positive value, or $x_i = 0$ and there is at least one vertex v_j connected to v_i such that $x_j = 2$.

In order to check if vector X is a feasible solution to the WRD problem, for every element x_i $(i = 1, \ldots, n)$ *feasibleSolution(X, WRD)* checks if it is a positive value, or $x_i = 0$ and at least one of the following two conditions holds:

1. there exists at least one element x_j $(j = 1, \ldots, n,\ j \neq i)$ with properties $x_j = 2$ and $j \in N_i$, i.e.

- after a single legion movement from a province j to a province s ($s \neq i, j$) there still is one legion stationed at a province j which defends provinces i and j;
- after a single legion movement from a province j to a province i, both provinces i and j are defended by stationed legions.

2. there exists at least one element x_j, $j \in N_i$, such that $x_j = 1$ and swapping the values of x_i and x_j does not affect the feasibility of the vector X. More precisely, after the swap, for every element x_s, $s \in N_j$, with property $x_s = 0$, there exists at least one x_k, $k \in N_s, k \neq j$, with property $x_k > 0$, i.e.

   - in order to move a single legion from a province j to a province i, all provinces s, which are neighbors with j and which are without any stationed legion, must have another neighbor k ($k \neq j$) with at least one stationed legion.

We will say that the function *feasibleSolution(X, problem)* is satisfied if there are no undefended provinces with respect to the *problem*.

Also, we create function *penalty(X, problem)*, which calculates the number of undefended provinces with respect to the *problem*.

Further, we will say that two solutions, X and X', have difference of the first order if one legion was moved from one province to another (value of one element, with value lower than 2, of the vector X, is increased by one, while value of the other element, with positive value, of the vector X, is decreased by one) or disbanded (value of one element, with positive value, of the vector X, is decreased by one). Respectively, two solutions have difference of the k^{th} order if at most k legions were moved, including possible disbanding.

Now, let us define a set $\mathcal{N}_k(X)$, $k = k_{min}, \ldots, k_{max}$ as the set of all vectors X' that have difference of the k^{th} order from the solution X and call that set k^{th} *Neighborhood to the solution* X.

The VNS-based heuristic can be defined in such a way that it starts from the *initial* feasible solution X, *shakes* it by creating another solution $X' \in \mathcal{N}_k(X)$ (by the expression *shake* we mean *movement of a certain number of legions*) and then applies *local search method* in order to create a better feasible solution X''. If the feasible solution X'', obtained by the local search procedure, is not better than the current incumbent X ($F(X'') \geq F^*$), the VNS algorithm repeats the procedure of shaking, but in neighborhood $\mathcal{N}_{k+k_{step}}(X)$ (i.e., k increments by k_{step}) and local search within it and so on until k reaches its maximum k_{max}. Otherwise, if $F(X'') < F^*$, X^* becomes X'', F^* becomes $F(X'')$ and k becomes k_{min}. Changing neighborhoods enables one to get out from the local minima. The VNS algorithm is presented as Algorithm 1. Functions *InitialSolution()*, *Shake()*, *LocalSearch()* and *StoppingCondition()* are described below.

Function *InitialSolution*() (pseudo code is presented as Algorithm 2) is defined so that it produces an initial feasible solution X^* by applying random changes to elements of the zero vector X. That is, *InitialSolution*() assigns randomly generated number from the set $\{1, 2\}$ to a randomly chosen element of the vector X until X

Algorithm 1 Variable Neighborhood Search metaheuristic

```
1: X* ← InitialSolution();
2: F* ← F(X*);
3: repeat
4:     k ← k_min;
5:     repeat
6:         X ← X*;
7:         X' ← Shake(X, k);
8:         X'' ← LocalSearch(X');
9:         if F(X'') < F* then
10:            F* ← F(X'');
11:            X* ← X'';
12:            k ← k_min;
13:        else
14:            k ← k + k_step;
15:    until k > k_max
16: until StoppingCondition()
```

Algorithm 2 *InitialSolution*()

```
1: X ← {0, ..., 0};
2: repeat
3:     i ← random number ∈ {1, ..., n};
4:     x_i ← random number ∈ {1, 2};
5: until (feasibleSolution(X, problem))
6: for i = 1, ..., n do
7:     if x_i > 0 then
8:         x_i ← x_i − 1;
9:         if not(feasibleSolution(X, problem)) then
10:            x_i ← x_i + 1;
```

becomes a feasible solution. Then, given that the function *InitialSolution*() finds a feasible solution, and our goal is to find a feasible solution such that the objective function value $F(X)$ ($F(X) = \sum_{i=1}^{n} x_i$) is minimal, the found solution will be, for now, saved as the best one ($X^* \leftarrow X$, $F^* \leftarrow F(X^*)$).

Further, in order to lower the value F^*, i.e., to improve the incumbent, among the elements of the vector X with positive value, *InitialSolution*() searches for an element whose value could be decreased by one such that the resulting vector remains a feasible solution. If such an element is found, *InitialSolution*() will decrease its value by one, and then continue to search for an element of the incumbent with the same property. Whenever the procedure of decreasing a value of one element produces a feasible vector, the resulting vector will be stored as the best one and objective function value $F(X)$ will be stored as F^*. This procedure repeats until there are no elements whose decreased value will result with feasible X.

Algorithm 3 *Shake*()

```
1: X ← X*
2: DecreasingProcedure(X);
3: for j = 1, ..., k do
4:     a ← random number ∈ {1, ..., n} such that x_a ≠ 0;
5:     b ← random number ∈ {1, ..., n} such that x_b ≠ 2;
6:     x_a ← x_a − 1;
7:     x_b ← x_b + 1;
8: if feasibleSolution(X, problem) then
9:     X* ← X;
10:    DecreasingProcedure(X);
```

Now, if it is possible to find a feasible solution with the same or smaller objective function value than F^*, the resulting solution will be better than the current incumbent. Hence, we define the following two functions, *Shake*() and *LocalSearch*(). These two functions are defined to search for a better feasible solution than the one with which they start the searching process.

Therefore, $Shake(X^*, k)$ function (presented as Algorithm 3) starts with a feasible solution X^*, stores it as X ($X \leftarrow X^*$) and then randomly chooses an element of the solution X with positive value and decreases its value by one. If the resulting vector is again a feasible solution, it stores it as the new best solution and repeats the process until an infeasible solution is found. We call this process *DecreasingProcedure*(). Then, among the elements of the current solution X with value lower than 2, shake function randomly choses one element, and among the elements with positive value of the incumbent X, it randomly chooses another element and increases a value of the first chosen element by one and decreases the value of the second chosen element also by one (i.e., it moves one legion) and repeats this process k times. If the resulting vector X' is a feasible one, given that $F(X') < F^*$ the new best feasible is found. Therefore, X' will be stored as the new best feasible ($X^* \leftarrow X'$). Also, if $X^{'}$ is feasible, we will apply *DecreasingProcedure*() to the vector $X^{'}$ and resulting vector denote as X' (note that in this case it follows that $F(X') \leq F^* - 1$).

Now, the $LocalSearch(X')$ function (presented as Algorithms 4 and 5) starts with an infeasible incumbent X', calculates its $penalty(X', problem)$ value and stores it as nd_{min}. Then it searches a neighborhood $\mathcal{N}_1(X')$ of the incumbent X' in order to find a feasible solution. If a solution with lower penalty value is found it will be stored as incumbent and search for a better solution continues. If a solution with penalty value equal to zero is found, it means that a feasible solution is found. If there is no solution with penalty value lower or equal to nd_{min} within the neighborhood $\mathcal{N}_1(X')$ of the incumbent, local search procedure will continue its search in the neighborhood $\mathcal{N}_2(X')$ of the incumbent. In both cases, whenever a feasible solution is found, it will be stored as the new best feasible solution. Also, local search procedure will continue to search for a feasible solution within the neighborhoods of the incumbent

Algorithm 4 *LocalSearch*()

```
1: nd_min ← penalty(X′, problem);
2: while some improvement is made do
3:     for i = 1, ..., n such that x_i′ > 0 do
4:         x_i′ ← x_i′ − 1;
5:         if feasibleSolution(X′, problem) then
6:             X* ← X′;
7:             DecreasingProcedure(X′);
8:             nd_min ← penalty(X′, problem);
9:             go to line 3;
10:        else
11:            for j = 1, ..., n, j ≠ i such that x_j′ < 2 do
12:                x_j′ ← x_j′ + 1;
13:                nd ← penalty(X′, problem);
14:                if nd = 0 then
15:                    execute lines 6-9;
16:                else
17:                    if nd < nd_min then
18:                        X′_better ← X′;
19:                        nd_min ← nd;
20:                    if nd = nd_min then
21:                        X′_same ← X′ with some probability;
22:                x_j′ ← x_j′ − 1;
23:        x_i′ ← x_i′ + 1;
24:    if X′_better is found then
25:        X′ ← X′_better;
26:    else
27:        if X′_same is found then
28:            with some probability X′ ← X′_same;
29:        else
30:            run LS2();
31: X″ ← X*;
```

(i.e., a decreasing procedure will be applied to the feasible incumbent) until there is no better feasible solution.

In other words, local search procedure consists of three steps. In the first step, local search procedure searches for an element (of the incumbent X') with positive value, decreases its value by one and checks if the resulting vector is a feasible one. If the resulting vector is a feasible solution, it will be stored as X^*. If the resulting vector is infeasible, the procedure goes to the second step of the local search. In the second step, the local search procedure searches for an element $x_j^{'}$ of the incumbent of the local search procedure with property $x_j^{'} < 2$, such that increasing its value by one creates a feasible solution. If the required element is

found, its value will be increased by one and the resulting feasible solution stored as X^*. If a feasible solution is found (both in the first and in the second step), $DecreasingProcedure()$ will be applied to that feasible incumbent, nd_{min} will be set to be equal to $penalty(X', problem)$ and the local search procedure will restart from the beginning of the first step (lines 6-9 and 15 of Algorithm 4). If the required element of the second step was not found, solution with the smallest penalty value $penalty(X', problem)$ will be stored as X'_{better} and the solution with the penalty value equal to the incumbent will be stored as X'_{same}. Then, when the second step is finished, in case that a better solution than the incumbent is found, it will be set as the incumbent solution and the second step will restart from the beginning. Similarly, if at least a solution of the same quality is found, it will be set as the incumbent solution with some probability and the second step will restart from the beginning. Otherwise, if there is no better solution nor a solution of the same quality, the third step of the local search procedure will start.

In the third step of the local search procedure, we explore a neighborhood $\mathcal{N}_2(X')$ of the incumbent in order to find a feasible solution. We denoted the third step of the local search procedure as $LS2()$ only because we want to make algorithm of $LocalSearch()$ function easier for reading.

In the third step (which is presented as Algorithm 5), the local search procedure searches for an element x'_i with value $x'_i = 2$ and for an element x'_j with value $x'_j < 2$ $(i, j = 1, \ldots, n)$. Then, it decreases the value of x'_i by two and increases a value of x'_j by one and then checks if a feasible solution is found, or if there exists an element $x'_s < 2$ such that increasing its value by one results with a feasible solution or with a better infeasible solution. Similarly as in the first two steps, $LS2()$ function computes *penalty()* value before and after each change and stores an incumbent solution X' with smaller penalty value than nd_{min} as X'_{better} and the incumbent with the same penalty value as X'_{same}. Again, whenever a better incumbent is found, nd_{min} will be set to be equal to *penalty*(X'_{better}, *problem)* and the incumbent solution of the same quality will be stored with some probability. Then, if a process of decreasing a value of an element x'_i by two and increasing a value of each pair of elements x'_j and x'_s by one does not create a feasible solution, values of elements x_i, x_j and x_s will be restored and the third step will continue its search with the next element whose value is equal to 2. In case that all element combinations are checked and better solution is found, it will be set as the incumbent and $LS2()$ will restart its search within the new incumbent. Similarly, in case that all elements combinations are checked and only a solution of the same quality is found, it will be set as the incumbent with some probability and $LS2()$ will restart.

During all the steps of the local search procedure we are also checking if moves from one solution to the solution of the same quality will not make a loop, i.e., we will not store the incumbent of the same quality if it will take us to some previous incumbent. Given that the size of a loop may vary, we do not allow moves from one incumbent to the incumbent of the same quality for more then k_{max} successive times. This means that the second and the third step will restart with the solution

Algorithm 5 $LS2()$

1: $nd_{min} \leftarrow penalty(X', problem)$
2: **while** some improvement is made **do**
3: **for** $i = 1, \ldots, n$ such that $x_i^{'} = 2$ **do**
4: $x_i^{'} \leftarrow x_i^{'} - 2$
5: **for** $j = 1, \ldots, n$ such that $x_j^{'} < 2$ **do**
6: $x_j^{'} \leftarrow x_j^{'} + 1$
7: **if** $feasibleSolution(X', problem)$ **then**
8: $X^* \leftarrow X'$
9: $DecreasingProcedure(X')$
10: $nd_{min} \leftarrow penalty(X', problem)$
11: go to line 2
12: **else**
13: **for** $s = 1, \ldots, n$, such that $x_s^{'} < 2$ **do**
14: $x_s^{'} \leftarrow x_s^{'} + 1$
15: **if** $feasibleSolution(X', problem)$ **then**
16: apply lines $8 - 11$
17: **else**
18: $nd \leftarrow penalty(X')$
19: **if** $nd < nd_{min}$ **then**
20: $X'_{better} \leftarrow X'$
21: $nd_{min} \leftarrow nd$
22: **if** $nd = nd_{min}$ **then**
23: $X'_{same} \leftarrow X'$ with some probability
24: $x_s^{'} \leftarrow x_s^{'} - 1$
25: $x_j^{'} \leftarrow x_j^{'} - 1$
26: $x_i^{'} \leftarrow x_i^{'} + 2$
27: **if** $X_{better}^{'}$ is found **then**
28: $X^{'} \leftarrow X_{better}^{'}$
29: **else**
30: **if** $X_{same}^{'}$ is found **then**
31: with some probability $X^{'} \leftarrow X_{better}^{'}$
32: **else**
33: finish $LS2()$

of the same quality for no more than k_{max} successive times. If some improvements are made within $LS2()$, the local search procedure restarts from the beginning of the first step with the new incumbent. Finally, when all three steps are finished and no improvement is made, *LocalSearch()* function will finish its search and the feasible solution X^* will be returned as X''. Now, if a better feasible solution is obtained $(F(X'') < F^*)$, its objective function value will be stored $(F^* \leftarrow F(X''))$ and k will be set to k_{min}, otherwise k will be increased by k_{step}. The VNS algorithm continues until k reaches its maximum or some other stopping condition occurs.

Input parameters for the VNS heuristic are the *problem*, the minimal (k_{min}) and the maximal (k_{max}) numbers of neighborhoods that should be searched, the increment of the parameter k (k_{step}) and the maximum CPU time allowed (t_{max}). In our implementation *StoppingCondition()* finishes the VNS algorithm if either k_{max} or maximal CPU time allowed is reached.

The parameters used for the proposed VNS algorithm are $k_{min} = 1$, $k_{max} = 30$, $k_{step} = 1$ and $t_{max} = 7\,200$ s and probability is set to $p = 0.5$.

The VNS algorithm cannot guarantee finding global optima because of its non-deterministic nature. Therefore, in order to find solution of sufficiently high quality it is necessary to run the VNS heuristic algorithm on the same instance more than once. Hence, in out experiments each instance was run 20 times.

4 COMPUTATIONAL RESULTS

Experimental results obtained by the proposed VNS algorithm for solving the RD and the WRD problems are presented in this section. The VNS algorithm was implemented in C++. All computational experiments have been performed on Intel® Core™ i7-4700MQ CPU@2.40 GHz with 8 GB RAM, under Windows 10 operating system.

CPLEX optimizations solver was run on all five formulations of the RD problem presented in [5] on grid, planar, net and randomly generated sets of graphs. The set of randomly generated graphs is the same as the one generated and proposed by Currò in [13] (names of instances consist of the number of vertices and of the probability that edge is incident to vertices expressed in percentage) while grid, net and planar sets are well known sets of graphs and also provided by Currò. Since there are several different ILP formulations of the Roman and the weak Roman domination problems (see [5] and [6]), and that performance of CPLEX differs in accordance with used ILP formulation, for the RD problem we present only instances for which optimal solution value is found, while for the WRD problem the results are presented on all instances with some known solution. In case that CPLEX was successful in finding an optimal solution value by using more than one formulation, the smallest running time is presented.

The results are summarized in Tables 1–8.

Tables 1–4 contain instances where CPLEX optimization solver was able to find and prove optimality of the found solution value for the RD problem (CPLEX was run for all five formulations of the RD problem presented in [5]). Tables 5–8 contains instances where CPLEX and Gurobi optimization solvers were successful in finding some solution value by using at least one ILP formulation presented in [6] within the given time. In all tables, whenever the optimal solution value is found by more than one formulation, the smallest running time is shown. Also, whenever optimization solver was unable to prove optimality of the found solution either because of time limit or "out of memory" status, in the column t_{sol} we put sign "–".

Instances are sorted by the number of vertices and the number of edges, in that order. Tables are organized as follows: The name of the instance is given in the first column. The next two columns ($|V|$, $|E|$) represent the number of vertices and the number of edges. In tables that correspond to the RD problem for all instances we have optimal solution values. Therefore, in the next two columns, *opt* and t_{cpl}, optimal solution value and minimal running time are given. In tables that correspond to the WRD problem we have three columns, the optimal solution value, the best solution value and the smallest running time, which is given regardless the optimization solver and ILP formulation. It should be noted that for the WRD problem optimal solution values and minimal running times of standard optimization solvers are taken from [6]. Also note that, in case that optimization solver could not provide an optimal solution value, a symbol "-" stands in the column t_{sol}.

For both problems, the VNS algorithm was run 20 times for each problem instance and informations of the best solution values obtained in these 20 runs are given in the final four columns (*sol*, t, *err*, σ) of all the tables. The best solution value obtained by the VNS algorithm is given in the column *sol* and whenever the VNS solution value was equal to the optimal solution value (from *opt* column), it was marked as "*opt*". The best time in 20 runs, necessary for the VNS algorithm to reach the corresponding solution in the first occurrence is given in the column t. The final two columns *err* and σ contains informations on the average solution quality: *err* stands for average relative error of found solutions from the best found solution, which is calculated as $err = \frac{1}{20}\sum_{i=1}^{20} err_i$, where $err_i = |\mathrm{VNS}_i - sol|/|\mathrm{VNS}_i|$, and VNS_i is the VNS solution obtained in the i^{th} run. Parameter σ is the standard deviation of the *err* obtained by the formula $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(err_i - err)^2}$.

The VNS algorithm for the RD problem is tested on 231 different instances and achieves the optimal solution on 218 of them. All solutions are found within the time limit (running time for 99 instances is lower than 1 second and only for 29 larger than 100 seconds). For majority of instances (on 214 instances), percentage average relative error from the found solution is lower than 2.5 %. Also, for the majority of instances (for 121 instances) the VNS heuristic running time is lower than the best CPLEX running time. Detailed informations of these testings are given in Tables 1–4.

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	*opt*	t_{cpl}	*sol*	t	*err*	σ
grid04x10	40	66	20	0.081	opt	0.01	0	0
grid05x08	40	67	21	0.081	opt	0.005	0	0
grid08x05	40	67	21	0.062	opt	< 0.01	0	0
grid10x04	40	66	20	0.077	opt	0.013	0	0
grid03x14	42	67	22	0.042	opt	< 0.01	0	0
grid06x07	42	71	22	0.119	opt	< 0.01	0	0
grid07x06	42	71	22	0.115	opt	< 0.01	0	0
grid14x03	42	67	22	0.062	opt	0.031	0	0

Table 1 continues ...

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	*opt*	t_{cpl}	*sol*	t	*err*	σ
grid04x11	44	73	22	0.057	opt	0.013	0	0
grid11x04	44	73	22	0.046	opt	< 0.01	0	0
grid03x15	45	72	24	0.046	opt	0.012	0	0
grid05x09	45	76	23	0.168	opt	< 0.01	0	0
grid09x05	45	76	23	0.148	opt	0.013	0	0
grid15x03	45	72	24	0.061	opt	0.022	0	0
grid04x12	48	80	24	0.058	opt	0.034	0	0
grid06x08	48	82	24	0.05	opt	0.033	0.0040	0.0120
grid08x06	48	82	24	0.098	opt	0.012	0.0040	0.0120
grid12x04	48	80	24	0.076	opt	0.019	0	0
grid07x07	49	84	24	0.098	opt	0.029	0.0060	0.0143
grid05x10	50	85	26	0.147	opt	< 0.01	0	0
grid10x05	50	85	26	0.166	opt	< 0.01	0	0
grid04x13	52	87	26	0.162	opt	0.104	0	0
grid13x04	52	87	26	0.099	opt	0.017	0.0019	0.0081
grid06x09	54	93	27	0.111	opt	0.436	0.0143	0.0175
grid09x06	54	93	27	0.179	opt	< 0.01	0.0071	0.0143
grid05x11	55	94	28	0.153	opt	0.013	0	0
grid11x05	55	94	28	0.184	opt	< 0.01	0	0
grid04x14	56	94	28	0.059	opt	0.035	0.0017	0.0075
grid07x08	56	97	28	0.131	opt	< 0.01	0	0
grid08x07	56	97	28	0.153	opt	0.033	0	0
grid14x04	56	94	28	0.06	opt	0.438	0.0356	0.0109
grid04x15	60	101	30	0.092	opt	< 0.01	0.0016	0.0070
grid05x12	60	103	30	0.13	opt	0.036	0.0194	0.0158
grid06x10	60	104	30	0.092	opt	0.041	0.0048	0.0115
grid10x06	60	104	30	0.152	opt	0.163	0.0097	0.0148
grid12x05	60	103	30	0.177	opt	0.078	0.0129	0.0158
grid15x04	60	101	30	0.075	opt	0.04	0	0
grid07x09	63	110	31	0.066	opt	0.135	0.0094	0.0143
grid09x07	63	110	31	0.162	opt	0.082	0	0
grid08x08	64	112	32	0.118	opt	0.031	0.0015	0.0066
grid05x13	65	112	33	0.173	opt	0.171	0.0029	0.0088
grid13x05	65	112	33	0.204	opt	0.054	0.0044	0.0105
grid06x11	66	115	33	0.137	opt	0.045	0.0059	0.0118
grid11x06	66	115	33	0.169	opt	0.246	0.0029	0.0088
grid05x14	70	121	35	0.207	opt	0.264	0.0083	0.0127
grid07x10	70	123	34	0.146	opt	0.164	0.0171	0.0140
grid10x07	70	123	34	0.119	opt	0.699	0.0171	0.0165
grid14x05	70	121	35	0.191	opt	0.19	0.0083	0.0127
grid06x12	72	126	36	0.169	opt	0.198	0.0054	0.0108
grid08x09	72	127	35	0.153	opt	0.017	0.0069	0.0120
grid09x08	72	127	35	0.125	opt	0.037	0.0110	0.0181

Table 1 continues ...

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	*opt*	t_{cpl}	*sol*	t	*err*	σ
grid12x06	72	126	36	0.186	opt	0.161	0.0014	0.0059
grid05x15	75	130	38	0.214	opt	0.352	0.0026	0.0077
grid15x05	75	130	38	0.247	opt	0.101	0	0
grid07x11	77	136	38	0.169	opt	0.094	0.0013	0.0056
grid11x07	77	136	38	0.186	opt	0.102	0.0026	0.0077
grid06x13	78	137	38	0.148	opt	1.553	0.0242	0.0148
grid13x06	78	137	38	0.209	opt	38	0.0256	0.0079
grid08x10	80	142	39	0.128	opt	0.132	0.0113	0.0124
grid10x08	80	142	39	0.142	opt	0.039	0.0075	0.0115
grid09x09	81	144	38	0.073	opt	2.937	0.0226	0.0236
grid06x14	84	148	41	0.134	opt	22.542	0.0306	0.0128
grid07x12	84	149	41	0.168	opt	1.727	0.0083	0.0114
grid12x07	84	149	41	0.192	opt	1.062	0.0024	0.0071
grid14x06	84	148	41	0.231	opt	7.766	0.0225	0.0116
grid08x11	88	157	42	0.192	opt	11.391	0.0275	0.0151
grid11x08	88	157	42	0.141	opt	0.778	0.0206	0.0187
grid06x15	90	159	44	0.247	opt	5.733	0.0188	0.0125
grid09x10	90	161	43	0.223	opt	1.224	0.0279	0.0184
grid10x09	90	161	43	0.237	opt	0.672	0.0102	0.0133
grid15x06	90	159	44	0.264	opt	3.141	0.0133	0.0109
grid07x13	91	162	44	0.178	opt	0.801	0.0177	0.0112
grid13x07	91	162	44	0.178	opt	0.882	0.0177	0.0131
grid08x12	96	172	46	0.21	opt	1.527	0.0178	0.0176
grid12x08	96	172	46	0.191	opt	5.175	0.0159	0.0113
grid07x14	98	175	47	0.247	opt	1.621	0.0247	0.0149
grid14x07	98	175	47	0.214	opt	2.929	0.0196	0.0137
grid09x11	99	178	47	0.194	opt	3.737	0.0124	0.0136
grid11x09	99	178	47	0.287	opt	4.522	0.0245	0.0187
grid10x10	100	180	48	0.22	opt	0.199	0.0051	0.0088
grid08x13	104	187	50	0.262	opt	0.274	0.0097	0.0130
grid13x08	104	187	50	0.401	opt	9.993	0.0146	0.0135
grid07x15	105	188	50	0.348	opt	20.739	0.0252	0.0121
grid15x07	105	188	50	0.278	opt	22.274	0.0243	0.0102
grid09x12	108	195	51	0.268	opt	9.665	0.0244	0.0204
grid12x09	108	195	51	0.29	opt	22.53	0.0208	0.0183
grid10x11	110	199	52	0.306	opt	2.545	0.0185	0.0192
grid11x10	110	199	52	0.256	opt	12.061	0.0222	0.0188
grid08x14	112	202	53	0.289	opt	6.864	0.0201	0.0138
grid14x08	112	202	53	0.284	opt	1.213	0.0228	0.0159
grid09x13	117	212	55	0.232	opt	10.045	0.0260	0.0224
grid13x09	117	212	55	0.439	opt	46.869	0.0262	0.0184
grid08x15	120	217	57	0.404	opt	5.05	0.0154	0.0119
grid10x12	120	218	56	0.236	opt	29.077	0.0381	0.0228

Table 1 continues ...

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	*opt*	t_{cpl}	*sol*	t	*err*	σ
grid12x10	120	218	56	0.326	opt	27.666	0.0343	0.0150
grid15x08	120	217	57	0.414	opt	18.631	0.0161	0.0155
grid11x11	121	220	57	0.443	opt	2.414	0.0219	0.0198
grid09x14	126	229	58	0.25	opt	0.518	0.0397	0.0277
grid14x09	126	229	58	0.334	opt	46.688	0.0458	0.0170
grid10x13	130	237	61	0.519	opt	12.797	0.0174	0.0178
grid13x10	130	237	61	0.529	opt	1.85	0.0188	0.0226
grid11x12	132	241	62	0.453	opt	26.008	0.0248	0.0183
grid12x11	132	241	62	0.464	opt	31.964	0.0204	0.0131
grid09x15	135	246	63	0.535	opt	36.088	0.0252	0.0185
grid15x09	135	246	63	0.733	opt	23.271	0.0312	0.0168
grid10x14	140	256	65	0.478	opt	78.302	0.0339	0.0165
grid14x10	140	256	65	0.432	opt	10.337	0.0359	0.0210
grid11x13	143	262	66	0.463	opt	70.571	0.0303	0.0223
grid13x11	143	262	66	0.503	opt	21.158	0.0372	0.0250
grid12x12	144	264	67	0.516	opt	36.922	0.0349	0.0185
grid10x15	150	275	70	0.715	opt	126.053	0.0266	0.0221
grid15x10	150	275	70	0.951	opt	24.143	0.0301	0.0189
grid11x14	154	283	71	0.483	opt	59.802	0.0438	0.0246
grid14x11	154	283	71	0.67	opt	62.236	0.0382	0.0203
grid12x13	156	287	72	0.715	**73**	115.106	0.0232	0.0159
grid13x12	156	287	72	0.783	opt	62.928	0.0384	0.0168
grid11x15	165	304	76	0.77	opt	117.803	0.0484	0.0198
grid15x11	165	304	76	0.918	opt	52.315	0.0406	0.0193
grid12x14	168	310	77	0.614	opt	181.88	0.0389	0.0205
grid14x12	168	310	77	0.721	opt	155.635	0.0424	0.0222
grid13x13	169	312	78	0.77	opt	68.571	0.0325	0.0191
grid12x15	180	333	82	0.94	**83**	164.384	0.0362	0.0191
grid15x12	180	333	82	1.3	**83**	130.71	0.0439	0.0207
grid13x14	182	337	83	0.777	opt	75.472	0.0486	0.0249
grid14x13	182	337	83	0.776	opt	201.98	0.0441	0.0270
grid13x15	195	362	89	1.73	opt	407.358	0.0483	0.0277
grid15x13	195	362	89	1.309	opt	139.451	0.0460	0.0239
grid14x14	196	364	88	0.739	opt	353.878	0.0516	0.0254
grid14x15	210	391	95	1.198	opt	282.147	0.0508	0.0250
grid15x14	210	391	95	1.159	opt	92.424	0.0543	0.0202
grid15x15	225	420	102	1.357	opt	697.859	0.0536	0.0240
grid20x20	400	760	176	37.579	**185**	676.713	0.0390	0.0135
grid30x20	600	1 150	260	1 279.438	**286**	5 114.624	0.0330	0.0160

Table 1. Experimental results for the RD problem on grid graph instances

From Table 1 it can be concluded that the VNS algorithm reaches the solution value equal to the optimal solution value on almost all instances (unsuccessful on 5 among 133 instances of grid type). On instances "grid12x13", "grid12x15", "grid15x12", "grid20x20" and "grid30x20", where an optimal solution was not reached, percentage average relative error from the found solution is lower than 2.1 %. Further, on 123 of 133 instances, percentage average relative error from the found solution is lower or equal to 2.5 % and on 5 instances between 2.5 % and 3 %. So, from Table 1 we can conclude that for the RD problem on grid graph instances the VNS algorithm provides solutions of good quality and within the time limit.

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	opt	t_{cpl}	sol	t	err	σ
plan10	10	27	3	0.048	opt	< 0.01	0	0
plan20	20	105	5	0.062	opt	< 0.01	0	0
plan30	30	182	5	0.046	opt	< 0.01	0	0
plan50	50	465	6	0.082	opt	< 0.01	0	0
plan100	100	1 540	10	0.0383	opt	0.054	0	0
plan150	150	2 867	12	1.303	opt	1.166	0	0
plan200	200	4 475	16	145.262	opt	2.466	0	0

Table 2: Experimental results for the RD problem on planar graph instances

From Table 2 it can be concluded that the VNS algorithm reaches the solution value equal to the optimal solution value on all instances with σ equal to zero. The VNS algorithm was also tested on instances "plan250" and "plan300" but, because CPLEX was unable to provide optimal solution values on these instances, we will not present the VNS algorithm results for these instances either. Also, we can conclude that instances of planar type are easier for solving for the VNS algorithm than for CPLEX, given the fact that the VNS algorithm provides results much more rapidly.

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	opt	t_{cpl}	sol	t	err	σ
Net-10-10	100	342	28	0.043	opt	0.129	0	0
Net-10-20	200	712	56	0.088	opt	18.013	0.0018	0.0053
Net-20-20	400	1 482	98	0.134	opt	944.94	0.0228	0.0316
Net-30-20	600	2 252	140	0.162	**145**	6916.4	0.0580	0.0274

Table 3: Experimental results for the RD problem on net graph instances

From Table 3 it can be concluded that the VNS algorithm reaches the solution value equal to the optimal solution value on 3 of 4 instances. On instance "Net-30-20", where an optimal solution value was not reached, percentage average relative error is equal to 2.74 %. Instances of the net type can be considered as easy for

solving for CPLEX given the fact that CPLEX is able to provide results for less than 1 second.

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	opt	t_{cpl}	sol	t	err	σ
Random-50-1	50	49	32	0.062	opt	0.031	0	0
Random-50-2	50	49	33	0.062	opt	0.069	0	0
Random-50-3	50	58	28	0.084	opt	0.029	0	0
Random-50-4	50	54	30	0.08	opt	0.006	0	0
Random-50-5	50	67	28	0.1	opt	0.005	0	0
Random-50-6	50	86	25	0.184	opt	0.041	0	0
Random-50-7	50	84	26	0.1	opt	< 0.01	0	0
Random-50-8	50	95	23	0.121	opt	< 0.01	0	0
Random-50-9	50	108	23	0.152	opt	0.011	0	0
Random-50-10	50	112	22	0.162	opt	0.021	0	0
Random-50-20	50	248	12	0.337	opt	< 0.01	0	0
Random-50-30	50	373	9	0.178	opt	< 0.01	0	0
Random-50-40	50	475	8	0.432	opt	< 0.01	0	0
Random-50-50	50	597	6	0.285	opt	< 0.01	0	0
Random-50-60	50	739	4	0.115	opt	< 0.01	0	0
Random-50-70	50	860	4	0.121	opt	< 0.01	0	0
Random-50-80	50	980	4	0.131	opt	< 0.01	0	0
Random-50-90	50	1 103	3	0.131	opt	< 0.01	0	0
Random-100-1	100	100	61	0.062	opt	4.662	0.0056	0.0092
Random-100-2	100	109	59	0.1	opt	2.744	0.0058	0.0095
Random-100-3	100	181	48	0.168	opt	3.767	0.0142	0.0113
Random-100-4	100	206	45	0.438	opt	0.895	0.0184	0.0103
Random-100-5	100	231	39	0.469	opt	3.425	0.0243	0.0251
Random-100-6	100	321	34	0.532	opt	3.572	0.0157	0.0142
Random-100-7	100	317	32	0.585	opt	3.291	0.0152	0.0152
Random-100-8	100	398	29	0.774	opt	0.669	0.0017	0.0073
Random-100-9	100	430	27	0.728	opt	0.389	0	0
Random-100-10	100	498	24	1.263	opt	3.95	0.0160	0.0196
Random-100-20	100	981	14	0.971	opt	0.086	0	0
Random-100-30	100	1 477	11	2.916	opt	0.137	0.0083	0.0250
Random-100-40	100	1 945	8	0.761	opt	0.052	0	0
Random-100-50	100	2 483	7	0.808	opt	0.049	0.0188	0.0446
Random-100-60	100	2 985	6	0.345	opt	< 0.01	0	0
Random-100-70	100	3 435	5	0.285	opt	0.044	0	0
Random-100-80	100	3 935	4	0.238	opt	< 0.01	0	0
Random-100-90	100	4 446	4	0.263	opt	< 0.01	0	0
Random-150-1	150	157	94	0.115	opt	22.389	0.0011	0.0032
Random-150-2	150	243	78	0.332	opt	234.872	0.0290	0.0151
Random-150-3	150	322	65	0.834	opt	67.784	0.0171	0.0162
Random-150-4	150	437	53	1.046	opt	30.304	0.0264	0.0155
Random-150-5	150	557	46	3.115	opt	2.293	0.0169	0.0142
Random-150-6	150	705	38	10.362	opt	19.279	0.0165	0.0165

Table 4 continues ...

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	opt	t_{cpl}	sol	t	err	σ
Random-150-7	150	778	34	5.622	opt	0.462	0.0057	0.0114
Random-150-8	150	906	31	18.691	opt	0.865	0	0
Random-150-9	150	965	30	10.489	opt	3.727	0.0064	0.0161
Random-150-10	150	1 152	27	45.44	opt	3.128	0.0054	0.0128
Random-150-20	150	2 228	16	31.857	opt	1.561	0	0
Random-150-30	150	3 318	12	21.507	opt	0.383	0	0
Random-150-40	150	4 476	9	13.628	opt	0.409	0.0700	0.0458
Random-150-50	150	5 550	8	17.671	opt	0.014	0	0
Random-150-60	150	6 734	6	1.742	opt	0.012	0	0
Random-150-70	150	7 807	6	8.667	opt	0.015	0	0
Random-150-80	150	8 924	4	0.366	opt	0.019	0	0
Random-150-90	150	10 043	4	0.839	opt	0.017	0	0
Random-200-1	200	229	116	0.132	**117**	173.552	0.0167	0.0119
Random-200-2	200	390	92	0.933	**93**	647.247	0.0294	0.0184
Random-200-3	200	581	69	2.69	opt	507.393	0.0403	0.0256
Random-200-4	200	737	60	13.301	opt	568.08	0.0433	0.0214
Random-200-5	200	1 010	47	60.589	opt	41.339	0.0354	0.0217
Random-200-6	200	1 180	42	245.778	opt	84.363	0.0518	0.0332
Random-200-7	200	1 453	36	130.93	opt	11.272	0.0093	0.0173
Random-200-30	200	5 876	12	153.586	opt	9.478	0.0110	0.0346
Random-200-40	200	7 907	10	89.663	opt	0.302	0	0
Random-200-50	200	9 895	8	30.844	opt	0.248	0	0
Random-200-60	200	11 971	6	7.707	opt	0.496	0	0
Random-200-70	200	14 059	6	19.27	opt	0.025	0	0
Random-200-80	200	15 918	4	0.831	opt	0.038	0	0
Random-200-90	200	17 821	4	0.801	opt	0.03	0	0
Random-250-1	250	345	136	0.21	**137**	1 111.594	0.0220	0.0130
Random-250-2	250	633	97	7.95	**99**	380.006	0.0304	0.0211
Random-250-3	250	956	73	257.891	opt	132.791	0.0305	0.0252
Random-250-4	250	1 194	62	1 406.04	opt	148.167	0.0224	0.0218
Random-250-30	250	9 347	13	1 408.412	**14**	1.005	0	0
Random-250-40	250	12 500	10	359.601	opt	0.743	0	0
Random-250-50	250	15 605	8	61.927	opt	0.621	0	0
Random-250-60	250	18 660	8	206.548	opt	0.037	0	0
Random-250-70	250	21 741	6	40.379	opt	0.037	0	0
Random-250-80	250	24 836	4	3.071	opt	0.465	0	0
Random-250-90	250	27 974	4	1.404	opt	0.052	0	0
Random-300-1	300	481	145	0.299	**149**	2 797.158	0.0221	0.0135
Random-300-2	300	876	103	116.818	**105**	1 057.238	0.0394	0.0192
Random-300-40	300	17 934	10	483.378	opt	3.232	0.0174	0.0437
Random-300-50	300	22 520	8	334.329	opt	31.909	0	0
Random-300-60	300	26 952	8	622.751	opt	0.069	0	0
Random-300-70	300	31 390	6	66.546	opt	0.286	0	0

Table 4 continues ...

Instance			CPLEX		VNS			
Name	$\|V\|$	$\|E\|$	opt	t_{cpl}	sol	t	*err*	σ
Random-300-80	300	35 871	5	34.579	opt	1.725	0.0667	0.0816
Random-300-90	300	40 412	4	2.191	opt	0.092	0	0

Table 4. Experimental results for the RD problem on random graph instances

Table 4 contains the results of the experimental testing on random generated graphs. As it can be seen, the VNS algorithm reaches the solution value equal to the optimal solution value on many instances (unsuccessful on 7 among 87 instances). On instances where an optimal solution was not reached, standard deviation σ is lower than 2.5 %. Instances "Random-200-8"–"Random-200-20", "Random-250-5"–"Random-250-20" and "Random-300-3"–"Random-300-30" are omitted from Table 4 because CPLEX was unable to find an optimal solution value on these instances. Nevertheless, the VNS algorithm finds some solution value for these instances, but because we do not have an optimal solution value on these instances, we will not present the VNS algorithm results either.

Before we present experimental results for the WRD problem on the same set of instances, let us summarize the results presented in Tables 1-4. The VNS algorithm for the RD problem finds solutions of good quality relatively fast, especially on instances of planar type. On instances of grid and net type, using CPLEX optimization solver is better, but on instances of planar and random type, using the VNS algorithm is preferable.

Experimental results of the VNS algorithm for the WRD problem are performed on instances where some solution values are known from the literature. Given that CPLEX was not able to solve the WRD problem on many instances within the time limit because of the "out of memory" status or because of the time limit, we tested the VNS algorithm both on instances where the optimal solution value is known and on instances where the found solution is not proved to be the optimal solution. Testings were made on 84 instances of different type. CPLEX optimization solver was able to find the optimal solution on 64 of them. The VNS algorithm was not able to find solutions equal to the optimal ones only on two instances. On instances where the optimal solution value is unknown, the VNS solutions are equal or better than the solutions found by CPLEX. Also, for almost all instances, the VNS algorithm runtime is lower than CPLEX runtime. Detailed information considering these testings is provided in Tables 5–8.

From Table 5 it can be concluded that the VNS reaches the solution value equal to the optimal solution value on almost all instances (unsuccessful only on "grid06x13"). On instances where the optimal solution value is unknown, σ is lower than 2.2 %. Running times on instances where the optimal solution value is known shows that the VNS rapidly reaches these solutions in lower than 150 seconds. Even more, on many instances (38 of 42), running times are smaller than 30 seconds and only on "grid07x14" and "grid08x12" greater than 100 seconds. On instances where

Instance				Solver		VNS			
Name	$\|V\|$	$\|E\|$	opt	val	t	sol	t	*err*	σ
grid04x10	40	66	15	15	4.109	opt	0.015	0	0
grid05x08	40	67	14	14	4.64	opt	0.047	0.0333	0.0333
grid03x14	42	67	16	16	4.829	opt	< 0.01	0	0
grid06x07	42	71	15	15	5.801	opt	0.08	0.0063	0.0188
grid04x11	44	73	16	16	5.5	opt	0.031	0.0088	0.0210
grid03x15	45	72	17	17	7.789	opt	0.012	0	0
grid05x09	45	76	16	16	7.908	opt	0.139	0.0235	0.0288
grid04x12	48	80	17	17	12.84	opt	0.069	0.0361	0.0265
grid06x08	48	82	18	18	25.499	opt	< 0.01	0	0
grid07x07	49	84	18	18	9.845	opt	0.021	0	0
grid05x10	50	85	18	18	10.61	opt	0.055	0.0053	0.0158
grid04x13	52	87	19	19	11.813	opt	0.035	0.0050	0.0150
grid06x09	54	93	19	19	25.539	opt	0.331	0.0450	0.0150
grid05x11	55	94	19	19	11.424	opt	0.388	0.0300	0.0245
grid04x14	56	94	20	20	35.326	opt	0.082	0.0214	0.0237
grid07x08	56	97	20	20	21.882	opt	0.076	0.0286	0.0233
grid04x15	60	101	22	22	40.256	opt	0.163	0	0
grid05x12	60	103	21	21	14.88	opt	4.036	0.0271	0.0260
grid06x10	60	104	21	21	35.713	opt	0.746	0.0273	0.0223
grid07x09	63	110	22	22	70.259	opt	0.318	0.0370	0.0155
grid08x08	64	112	23	23	171.925	opt	0.037	0.0063	0.0149
grid05x13	65	112	23	23	67.007	opt	0.928	0.0208	0.0208
grid06x11	66	115	24	24	381.771	opt	0.757	0.0040	0.0120
grid05x14	70	121	24	24	73.489	opt	27.03	0.0491	0.0202
grid07x10	70	123	25	25	618.089	opt	0.67	0.0077	0.0154
grid06x12	72	126	26	26	1 166.405	opt	0.544	0.0074	0.0148
grid08x09	72	127	25	25	435.146	opt	15.935	0.0383	0.0117
grid05x15	75	130	26	26	288.06	opt	8.133	0.0313	0.0174
grid07x11	77	136	27	27	988.596	opt	0.582	0.0268	0.0155
grid06x13	78	137	27	27	1 005.126	**28**	0.407	0.0086	0.0149
grid08x10	80	142	28	28	2 162.812	opt	10.011	0.0375	0.0178
grid09x09	81	144	28	28	737.579	opt	12.521	0.0437	0.0251
grid06x14	84	148	30	30	–	30	2.319	0.0097	0.0148
grid07x12	84	149	29	29	4 637.38	opt	47.642	0.0441	0.0181
grid08x11	88	157	31	31	–	31	3.412	0.0278	0.0190
grid06x15	90	159	32	32	–	32	1.196	0.0179	0.0218
grid09x10	90	161	31	31	–	31	40.651	0.0443	0.0197
grid07x13	91	162	32	32	–	32	16.778	0.0272	0.0130
grid08x12	96	172	33	33	–	33	107.765	0.0403	0.0184
grid07x14	98	175	34	34	1 720.86	opt	143.804	0.0433	0.0181
grid09x11	99	178	35	35	–	35	2.261	0.0181	0.0132
grid10x10	100	180	35	35	–	35	6.63	0.0302	0.0168

Table 5: Experimental results for the WRD problem on grid graph instances

optimization solvers were unable to prove optimality of the found solutions, the VNS heuristic reaches the same solution values for less than 108 seconds. So, we can conclude that the VNS heuristic solves the WRD problem on grid graph instance significantly faster than the optimization solver CPLEX and found solutions are of good quality.

Instance				Solver		VNS			
Name	$\|V\|$	$\|E\|$	opt	val	t	sol	t	*err*	σ
plan10	10	27	3	3	0.156	opt	< 0.01	0	0
plan20	20	105	3	3	1.36	opt	< 0.01	0	0
plan30	30	182	5	5	7.49	opt	< 0.01	0	0
plan50	50	465	6	6	98.49	opt	0.01	0	0
plan100	100	1 540	9	9	–	8	4.916	0	0
plan150	150	2 867	13	13	–	10	88.248	0.0273	0.041

Table 6: Experimental results for the WRD problem on planar graph instances

From Table 6 it can be concluded that the VNS algorithm reaches the solution value equal to the optimal solution value on all instances. Also, on instances where optimization solvers were unable to prove optimality of the found solution, the VNS solution is better. Again, running time for the instances where the optimal solution value is known is lower than 1 second. On "plan100", where optimization solvers were unable to prove optimality of the found solution, the proposed VNS algorithm finds solution value with σ equal to zero. On "plan150" the VNS solution is equal to 10 with $\sigma = 0.0417$, which can be considered as the solution of the good quality (solution value equal to 10 was reached in 14 of 20 runnings).

Instance				Solver		VNS			
Name	$\|V\|$	$\|E\|$	opt	val	t	sol	t	*err*	σ
Net-10-10	100	342	20	20	148.213	opt	4.29	0.0095	0.0190
Net-10-20	200	712	40	40	–	40	67.323	0.0146	0.0119
Net-20-20	400	1 482	83	83	–	81	2 066.577	0.0180	0.0132
Net-30-20	600	2 252	122	122	–	**123**	6 034.018	0.0474	0.0352

Table 7: Experimental results for the WRD problem on net graph instances

In Table 7 optimization solvers were able to find optimal solution value only for "Net-10-10". The same solution value was found by the proposed VNS algorithm with lower running time and with σ equal to 1.9 %. On "Net-10-20" and "Net-20-20" the VNS algorithm reaches the same and better solution value than optimization solvers, while for "Net-30-20" the VNS solution value is worse than the solvers' solution value.

From Table 8 it can be concluded that the VNS algorithm reaches the solution value equal to the optimal solution value on almost all instances (unsuccessful only on 1 among 25 instances of random type). On instance "Random-100-6", where the

Instance				Solver		VNS			
Name	$\|V\|$	$\|E\|$	opt	val	t	sol	t	*err*	σ
Random-50-1	50	49	24	24	0.281	opt	< 0.01	0	0
Random-50-2	50	49	23	23	0.343	opt	0.034	0	0
Random-50-3	50	58	24	24	0.39	opt	0.062	0	0
Random-50-4	50	54	24	24	0.484	opt	0.225	0	0
Random-50-5	50	67	22	22	0.968	opt	0.377	0.0196	0.0216
Random-50-6	50	86	19	19	2.053	opt	0.03	0	0
Random-50-7	50	84	19	19	3.171	opt	0.889	0.0175	0.0238
Random-50-8	50	95	17	17	3.093	opt	0.131	0.0333	0.0272
Random-50-9	50	108	17	17	26.373	opt	0.129	0.0028	0.0121
Random-50-10	50	112	16	16	6.781	opt	0.047	0	0
Random-50-20	50	248	9	9	346.264	opt	< 0.01	0	0
Random-50-30	50	373	7	7	476.278	opt	0.038	0	0
Random-50-40	50	475	6	6	1447.318	opt	0.092	0	0
Random-50-50	50	597	5	5	1545.06	opt	0.013	0	0
Random-50-60	50	739	4	4	210.71	opt	0.014	0	0
Random-50-70	50	860	3	3	156.14	opt	0.059	0	0
Random-50-80	50	980	3	3	90.813	opt	< 0.01	0	0
Random-50-90	50	1103	2	2	36.53	opt	0.03	0	0
Random-100-1	100	100	46	46	0.64	opt	157.329	0.0354	0.0145
Random-100-2	100	109	46	46	0.843	opt	36.052	0.0148	0.0117
Random-100-3	100	181	37	37	7.421	opt	23.64	0.0445	0.0261
Random-100-4	100	206	34	34	61.702	opt	12.367	0.0213	0.0175
Random-100-5	100	231	32	32	164.502	opt	60.361	0.0299	0.0186
Random-100-6	100	321	26	26	5 806.74	**27**	12.441	0.0265	0.0217
Random-100-7	100	317	25	25	4 009.377	opt	204.939	0.0434	0.0234
Random-100-8	100	317	23	23	–	23	313.924	0.0448	0.0279
Random-100-9	100	430	21	21	–	21	4.98	0.0269	0.0293
Random-100-10	100	498	19	19	–	19	460.905	0.0445	0.0260
Random-100-20	100	981	12	12	–	11	8.951	0.0250	0.0382
Random-100-30	100	1 477	11	11	–	8	1 462.462	0.1056	0.0242
Random-100-40	100	1 945	9	9	–	7	1.501	0	0
Random-100-50	100	2 483	7	7	–	5	37.134	0	0

Table 8: Experimental results for the WRD problem on random generated graph instances

optimal solution value was not reached, σ is equal to 2.17 %. Further, on instances "Random-100-40" and "Random-100-50", where optimization solvers were unable to prove optimality of the found solution, the VNS algorithm finds better solutions values with σ equal to zero for less than 38 seconds.

From Tables 5–8 we can see that optimization solvers were unable to provide an optimal solution value on instances of grid type with number of vertices larger than 84, on instances of planar and net type with number of vertices larger than

100 and on large number of instances of random type with 100 vertices. Also, we can see that, on the same set of instances, the VNS algorithm finds solutions of the WRD problem of good quality and, for many instances, faster than optimization solvers.

5 CONCLUSIONS

In this paper, the Variable Neighborhood Search approach for solving the Roman and the weak Roman domination problems is proposed. Tests were run on grid, net, planar and randomly generated graphs, with up to 600 vertices. The VNS was able to find solutions equal to the optimal ones for the RD problem on 218 of 231 tested instances and able to find solutions equal or better than CPLEX solutions for the WRD problem on 84 of 86 tested instances. Therefore, we can conclude that the VNS algorithm provides good quality solutions regardless of the type of instance and the type of problem, which makes it efficient for solving both the Roman and the weak Roman domination problems. Moreover, given the fact that optimization solvers were not able to solve the WRD problem on large scale instances (i.e., instances with more than 100 vertices) proposed algorithm can be used. Furthermore, given the fact that this algorithm does not contain any limitations on the number of variables and the number of conditions, it can be used for solving the RD problem on instances where optimization solvers are not able to provide an optimal solution value.

In future work, hybridization with some exact methods or application of some other heuristic could lead to possible better achievements in solving the Roman and the weak Roman domination problems.

Acknowledgments

This research has been partially supported by the Serbian Ministry of Education, Science and Technological Development under grants No. TR36006 and ON174010. The authors gratefully acknowledge the two referees of this paper for many excellent suggestions which have helped in improving this paper.

REFERENCES

[1] ReVelle, C. S.—Rosing, K. E.: Defendens Imperium Romanum: A Classical Problem in Military Strategy. The American Mathematical Monthly, Vol. 107, 2000, No. 7, pp. 585–594, doi: 10.2307/2589113.

[2] Cockayne, E. J.—Dreyer, P. A.—Hedetniemi, S. M.—Hedetniemi, S. T.: Roman Domination in Graphs. Discrete Mathematics, Vol. 278, 2004, No. 1-3, pp. 11–22, doi: 10.1016/j.disc.2003.06.004.

[3] Henning, M. A.—Hedetniemi, S. T.: Defending the Roman Empire – A New Strategy. Discrete Mathematics, Vol. 266, 2003, No. 1-3, pp. 239–251, doi: 10.1016/s0012-365x(02)00811-7.

[4] Burger, A. P.—de Villiers, A. P.—van Vuuren, J. H.: A Binary Programming Approach Towards Achieving Effective Graph Protection. Proceedings of the 2013 ORSSA Annual Conference, ORSSA, 2013, pp. 19–30.

[5] Ivanović, M.: Improved Mixed Integer Linear Programing Formulations for Roman Domination Problem. Publications de l'Institut Mathématique, Vol. 99, 2016, No. 113, pp. 51–58, doi: 10.2298/PIM1613051I.

[6] Ivanović, M.: Improved Integer Linear Programming Formulation for Weak Roman Domination Problem. Soft Computing, Vol. 22, 2018, No. 19, pp. 6583–6593, doi: 10.1007/s00500-017-2706-4.

[7] Liu, C. S.—Peng, S. L.—Tang, C. Y.: Weak Roman Domination on Block Graphs. Proceedings of the 27$^{\text{th}}$ Workshop on Combinatorial Mathematics and Computation Theory, Providence University, Taichung, Taiwan, April 30–May 1, 2010, pp. 86–89.

[8] Burger, A. P.—Cockayne, E. J.—Grundlingh, W. R.—Mynhardt, C. M.—Van Vuuren, J. H.—Winterbach, W.: Finite Order Domination in Graphs. Journal of Combinatorial Mathematics and Combinatorial Computing, Vol. 49, 2004, pp. 159–176.

[9] Stewart, I.: Defend the Roman Empire! Scientific American, Vol. 281, 1999, pp. 136–138, doi: 10.1038/scientificamerican1299-136.

[10] Chellali, M.—Haynes, T. W.—Hedetniemi, S. T.: Bounds on Weak Roman and 2-Rainbow Domination Numbers. Discrete Applied Mathematics, Vol. 178, 2014, pp. 27–32, doi: 10.1016/j.dam.2014.06.016.

[11] Dreyer Jr, P. A.: Applications and Variations of Domination in Graphs. Ph.D. thesis, Rutgers University, 2000.

[12] Cockayne, E. J.—Grobler, P. J. P.—Gründlingh, W. R.—Munganga, J.—van Vuuren, J. H.: Protection of a Graph. Utilitas Mathematica, Vol. 67, 2005, pp. 19–32.

[13] Currò, V.: The Roman Domination Problem on Grid Graphs. Ph.D. thesis, Università di Catania, 2014.

[14] Favaron, O.—Karami, H.—Khoeilar, R.—Sheikholeslami, S. M.: On the Roman Domination Number of a Graph. Discrete Mathematics, Vol. 309, 2009, No. 10, pp. 3447–3451, doi: 10.1016/j.disc.2008.09.043.

[15] Klobučar, A.—Puljić, I.: Some Results for Roman Domination Number on Cardinal Product of Paths and Cycles. Kragujevac Journal of Mathematics, Vol. 38, 2014, No. 1, pp. 83–94, doi: 10.5937/KgJMath1401083K.

[16] Klobučar, A.—Puljić, I.: Roman Domination Number on Cardinal Product of Paths and Cycles. Croatian Operational Research Review, Vol. 6, 2015, No. 1, pp. 71–78, doi: 10.17535/crorr.2015.0006.

[17] Xing, H.-M.—Chen, X.—Chen, X.-G.: A Note on Roman Domination in Graphs. Discrete Mathematics, Vol. 306, 2006, No. 24, pp. 3338–3340, doi: 10.1016/j.disc.2006.06.018.

[18] Wang, H.—Xu, X.—Yang, Y.—Ji, C.: Roman Domination Number of Generalized Petersen Graphs $p(n, 2)$. arXiv Preprint, arXiv:1103.2419, 2011.

[19] Pavlič, P.—Žerovnik, J.: Roman Domination Number of the Cartesian Products of Paths and Cycles. The Electronic Journal of Combinatorics, Vol. 19, 2012, No. 3, Art. No. P19.

[20] Liedloff, M.—Kloks, T.—Liu, J.—Peng, S.-L.: Efficient Algorithms for Roman Domination on Some Classes of Graphs. Discrete Applied Mathematics, Vol. 156, 2008, No. 18, pp. 3400–3415, doi: 10.1016/j.dam.2008.01.011.

[21] Liedloff, M.—Kloks, T.—Liu, J.—Peng, S. L.: Roman Domination over Some Graph Classes. In: Kratsch, D. (Ed.): Graph-Theoretic Concepts in Computer Science (WG 2005). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3787, 2005, pp. 103–114, doi: 10.1007/11604686_10.

[22] Shang, W.—Hu, X.: The Roman Domination Problem in Unit Disk Graphs. In: Shi, Y., van Albada, G. D., Dongarra, J., Sloot, P. M. A. (Eds.): Computational Science (ICCS 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4489, 2007, pp. 305–312, doi: 10.1007/978-3-540-72588-6_51.

[23] Pushpam, P. R. L.—Malini Mai, T. N. M.: Weak Roman Domination in Graphs. Discussiones Mathematicae Graph Theory, Vol. 31, 2011, No. 1, pp. 161–170, doi: 10.7151/dmgt.1532.

[24] Lai, Y. L.—Lin, C. T.—Ho, H. M.: Weak Roman Domination on Graphs. Proceedings of the 28th Workshop on Combinatorial Mathematics and Computation Theory, National Penghu University of Science and Technology, Penghu, Taiwan, May 27–28, 2011, pp. 224–214.

[25] Peng, S.-L.—Tsai, Y.-H.: Roman Domination on Graphs of Bounded Treewidth. Proceedings of the 24th Workshop on Combinatorial Mathematics and Computation Theory, 2007, pp. 128–131.

[26] Chapelle, M.—Cochefert, M.—Couturier, J.-F.—Kratsch, D.—Liedloff, M.—Perez, A.: Exact Algorithms for Weak Roman Domination. In: Lecroq, T., Mouchard, L. (Eds.): Combinatorial Algorithms (IWOCA 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 8288, 2013, pp. 81–93, doi: 10.1007/978-3-642-45278-9_8.

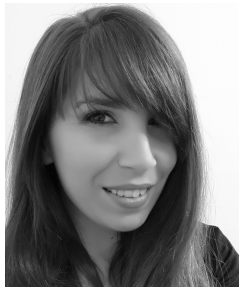
[27] Hansen, P.—Mladenović, N.—Urošević, D.: Variable Neighborhood Search for the Maximum Clique. Discrete Applied Mathematics, Vol. 145, 2004, No. 1, pp. 117–125, doi: 10.1016/j.dam.2003.09.012.

[28] Brimberg, J.—Mladenović, N.—Urošević, D.—Ngai, E.: Variable Neighborhood Search for the Heaviest k-Subgraph. Computers and Operations Research, Vol. 36, 2009, No. 11, pp. 2885–2891, doi: 10.1016/j.cor.2008.12.020.

[29] Mladenović, N.: A Variable Neighborhood Algorithm – A New Metaheuristic for Combinatorial Optimization. Papers Presented at Optimization Days, 1995, p. 112.

[30] Mladenović, N.—Hansen, P.: Variable Neighborhood Search. Computers and Operations Research, Vol. 24, 1997, No. 11, pp. 1097–1100, doi: 10.1016/s0305-0548(97)00031-2.

[31] Hansen, P.—Mladenović, N.: An Introduction to Variable Neighborhood Search. In: Voss, S., Martello, S., Osman, I. H., Roucairol, C. (Eds.): Meta-Heuristics. Springer, Boston, MA, 1999, pp. 433–458, doi: 10.1007/978-1-4615-5775-3_30.



Marija Ivanović finished her master studies at Faculty of Mathematics, University of Belgrade, in 2011. Since 2007 she has worked at the Faculty of Mathematics, Department for Numerical Mathematics and Optimization as Assistant. The main areas of research are game theory, combinatorial optimization and operations research. She participates in research projects financed by the Ministry of Education, Science and Technological Development, Serbia.

Dragan Urošević finished his Ph.D. studies at Faculty of Mathematics, University of Belgrade, in 2004. Since 1993 he has worked at the Mathematical Institute SANU. The main areas of research are combinatorial optimization and operations research. He is engaged in the development and implementation of heuristic methods to solve complex problems in graph theory and the development of methods for solving location problems. He participates in research projects financed by the Ministry of Education, Science and Technological Development, Serbia.

Computing and Informatics, Vol. 38, 2019, 85–114, doi: 10.31577/cai_2019_1_85

A PROBABILISTIC EXTENSION OF UML-B

Mohammad Nosrati, Hassan Haghighi

Shahid Beheshti University
Evin, Tehran,Iran
e-mail: mohammad.nosrati@gmail.com, h_haghighi@sbu.ac.ir

Abstract. This paper extends the graphical and formal language of UML-B to provide the ability to model probabilities. Discrete probabilities, interval probabilities, and stochastic delays are added to the UML-B's state-machine syntax, and their corresponding semantics are defined in Event-B. In addition, as a secondary contribution, UML-B (probabilistic) state-machine models are defined as MDP (Markov Decision Process) models in order to provide a means of quantitative verification in PRISM (Probabilistic Symbolic Model Checker). As an important feature of the proposed method, it does not change the Event-B syntax or semantics. To evaluate this work, as a case study, the Zeroconf protocol will be modeled in the extended UML-B using the Rodin tool, and its Event-B counterpart is converted to a PRISM model. The results of evaluations indicate that this study's additions provide the capability of modeling and verification of probabilistic and stochastic systems.

Keywords: UML-B, Event-B, probabilistic systems, interval probabilities, stochastic delay, probabilistic model verification, MDP, PRISM

1 INTRODUCTION

To facilitate software specification and design, and to simplify the communication between software stakeholders (especially, software engineers), graphical, semi-formal languages, such as UML, have been developed to model software artifacts.

In spite of the benefits that semi-formal languages provide, the lack of a precise formal semantics can lead to ambiguity and inconsistency. For example, Reggio et al. [27] have identified 31 problems concerning ambiguity, incompleteness, and inconsistency in UML 1.3. But then, the difficulty of writing formal specifications and understanding these specifications by software practitioners are serious problems. If

the priority of a modeler is to use a means for abstract communication, a less formal language is preferred. But in case the modeler is seeking semantic correctness and rigor, then formal method is favored.

One approach to increase formalism is to transform specifications notated in languages like UML to an intermediate formal language, such as Fiacre (Format Intermédiaire pour les Architectures de Composants Répartis Embarqués) [11] to perform formal analysis and verification.

As another approach to avoid the problems with semi-formal methods such as UML and to overcome the difficulties in applying formal methods like Event-B while retaining their benefits, a language called UML-B has been proposed that attempts to combine the simplicity and intuitiveness of UML and preciseness and unambiguity of Event-B.

Probability is one of the main concepts required for expressing aspects common in real-time, fault-tolerant, and distributed systems because it allows the designer to specify system's random behavior quantitatively. Due to the importance of modeling and verification of probabilistic systems and behaviors, various programming and modeling languages have covered this notion. Especially, there are a number of major approaches for formally modeling and analyzing probabilistic systems in the literature.

Probabilistic model checking is one of the commonly used techniques. In this approach, a state-based mathematical model of the probabilistic system is constructed from its description in a high level language, and then, it is examined whether this model satisfies a given probabilistic property. A number of tools have been developed based on this method, with PRISM being among the most notable ones. Güdemann et al. [14] introduced SAML (Safety Analysis Modeling Language) to unify probabilistic and logical analysis of models to decouple the model from the actual verification tool, by converting model.

Another formal approach to model probabilities is using proof-based methods and languages, such as Hoare Logic [8], B [18] and Event-B [15], which have been extended with probabilistic choice. In [18], Hoang et al. have extended Abrial's Generalized Substitution Language (GSL) [1] to get pGSL that includes random algorithms within its scope to initiate the development of probabilistic B (pB). Hallerstede et al. [15] have extended the Event-B formalism with a qualitative probabilistic choice operator. When extending proof-based methods, refinements or probabilistic invariants are used to reason about the probabilistic system. One major benefit of this approach is its automatic nature.

A third alternative approach is to use the Higher Order Logic (HOL) to formalize random systems. In this approach, random variables are expressed formally in the higher order logic, and probabilistic properties are verified in a theorem prover. Hurd's Ph.D. thesis [19] and [17] are among works in this area.

Since, despite its advantages and applications, there has been no research on the subject of modeling probabilities in UML-B, the main objective of this research is to provide the ability to model probabilistic requirements in UML-B. By this contribution, it will be possible to specify discrete and interval probabilities, as well

as stochastic delays in UML-B and transform the resulting models to the Event-B formal specification language. To achieve this goal, the syntax and semantics of the mentioned notions are defined. As a secondary contribution, this paper defines UML-B state-machines as MDP models to make it possible to convert them to PRISM models for quantitative and probabilistic verification purposes.

Despite similarities between UML and UML-B from a syntactic viewpoint, there are considerable differences between the semantics of these two languages. UML-B, concerned in our work, is based on the Event-B formal methods. Therefore, our main contribution in comparison to works like [20, 21] is our formal approach for defining the semantics of probabilistic constructs. Relying on UML-B and Event-B, the proposed method not only benefits from simplicity and intuitiveness of graphical modeling languages, but also offers the advantages of formal methods, such as lack of ambiguity in specifications, increasing accuracy and consistency, and providing means to formal verification and reasoning. In addition, in contrast to similar works, like [32], one of the main features of this paper is that it does not change the Event-B syntax and semantics to achieve its goals. Therefore, the proposed method can be directly used in the current Event-B tools.

The paper is organized as follows. In Section 2, the research background and an overview on the most related work are briefly presented. Section 4 introduces our extensions to UML-B. In Section 5, we show the applicability of the proposed method by applying it to a case study. Section 6 is dedicated to the conclusions and some directions for future work.

2 BACKGROUND

2.1 Event-B

As an evolution of B-Method developed by Jean-Raymond Abrial, Event-B [1] is a formal method for modeling and analyzing systems. In Event-B, machines model reactive (event-based) systems that continually execute enabled events. Event-based systems have no interfaces or parameters. Instead, inputs are modeled as nondeterministic changes. The state of the model is represented as a collection of variables. The dynamic behavior of the system is defined by a number of events. Events modify the system state, by executing an action [6]. An event has the following form:

$$e: \ \textbf{ANY}\ lv\ \textbf{WHERE}\ G_e\ \textbf{THEN}\ r\ \textbf{END}$$

where lv is a list of local variables, guard G_e is a predicate over the system state and local variables, and action r is a multiple assignment over the system variables. An event becomes enabled only when its corresponding guard becomes true. As a result of executing an action, one or multiple parallel assignments will be performed. Variable assignments can be deterministic or nondeterministic. The deterministic assignment is denoted by $x := E(v)$ where x is a variable and $E(v)$ is an expression over system variables. Nondeterministic assignments are denoted by $x :\in S$ or

$x :| BA_e(x, v, x')$. S is a set, and $BA_e(x, v, x')$ is a predicate over system variables. As a result of these assignments, x is assigned any value from the set S, or it gets a value x' such that $BA_e(x, v, x')$ is true [26].

An Event-B model is a tuple $(C, \Sigma, A, v, I, S, E, Init)$ where C is a set of model constants; Σ is a set of model sets; A is a set of axioms over C and Σ; v is a set of system variables; I is a set of invariant properties; S is a set of model states, defined by all possible values of v; E is a set of system events; and $Init$ is a predicate defining the set of initial states.

The semantics of events is defined using before-after (BA) predicates [2]. A before-after predicate describes a relationship between the variable values before and after the execution of an event. An event $e \in E$ is a tuple $e = (G_e, BA_e)$ where $G_e \in S \rightarrow BOOL$ is the guard and $BA_e \in S \times S \rightarrow BOOL$ is the before-after predicate [10].

Model correctness is demonstrated by generating and discharging a collection of Proof Obligations (POs). Every Event-B model should satisfy the event feasibility $((G_e(\sigma) \wedge I(\sigma')) \Rightarrow \exists \sigma'.BA_e(\sigma, \sigma'))$ and invariant properties $((G_e(\sigma) \wedge I(\sigma) \wedge BA_e(\sigma, \sigma')) \Rightarrow I(\sigma'))$, where σ and σ' are states and I is the invariant. The feasibility of an event means that whenever an event is enabled, there is some reachable after-state. Each event should also preserve the model invariant.

The behavior of an Event-B model is given by a transition system for which transition relations are given by this rule:

$$\frac{\exists (G_e, BA_e) \in E. \exists \sigma, \sigma' \in S. I(\sigma) \wedge G_e(\sigma) \wedge BA_e(\sigma, \sigma') \wedge I(\sigma')}{\sigma \rightarrow \sigma'}.$$

This rule states if there is a (G_e, BA_e) pair in the set of system events, and there are states σ and σ' such that the invariant and guard G_e are true in state σ, before-after predicate BA_e holds for states σ and σ', and finally, the invariant is true for state σ', then a transition exists from σ to σ'.

Event-B is supported by the Rodin tool [3]. The Rodin platform is an open source Eclipse-based IDE that effectively supports refinement and mathematical proof of models.

2.2 UML-B

UML-B [31] is a graphical formal notation based on the graphical notation of UML [29]. It relies on Event-B semantically, although, its initial version was translated to classical B [1].

UML-B provides four kinds of diagrams. They are package, context, class and state-machine diagrams. Package diagrams are used to describe the relationships between top level components (machines and contexts). The context diagram defines the static (constant) part of a model. Transitions of a state-machine represent events. Class diagrams are used to describe the behavioral part of a model. For further information see [30].

2.3 PRISM

Analysis and verification of a system involve establishing qualitative and quantitative properties of the system. For example, the property that states "the system eventually terminates" is a qualitative one. On the other hand, the property that states "the system terminates within a given time limit with a given probability" is a quantitative one. To perform probabilistic and quantitative verification on the resulting models of this work, we need to use a probabilistic model checker.

PRISM [24] is a model checking tool which supports verification of probabilistic models. This tool takes as input a description of a probabilistic system written in the PRISM language. It constructs a model, such as Deterministic-Time Markov Chain (DTMC), Continuous-Time Markov Chain (CTMC), or MDP from this description. It also accepts the properties specification in languages such as PCTL (Probabilistic Computation Tree Logic) [16] and performs model checking to determine which states of the model satisfy the specified property and with what probabilities. Model checking is reduced to a combination of reachability-based computation and the solution of linear equation systems. The PRISM kernel handles these computations using different engines [22].

The basic syntax of PCTL [23] is given by this grammar:

$$\Phi ::= \mathit{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid P_{\sim p}[\Box\Phi] \mid P_{\sim p}[\Phi \mathbf{U} \Phi]$$

where a is an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$, and $p \in [0, 1]$. The operator $P_{\sim p}[\Phi]$ means that the probability of Φ being true in a state s in a model (such as an MDP), is $\sim p$. $\Box\Phi$ expresses that Φ is satisfied in the next step and $\Phi_1 \mathbf{U} \Phi_2$ means that Φ_2 is eventually satisfied and Φ_1 is true until then. Other useful operators can be derived from the basic PCTL syntax, such as $\Diamond\Phi \equiv \mathit{true}\ \mathbf{U}\Phi$, meaning Φ will be eventually true.

2.4 Markov Decision Process (MDP)

In this paper, we use MDPs as models to which probabilistic state-machines are converted during the translation of probabilistic UML-B to the PRISM language. So, we give a brief overview on MDPs in this subsection. An MDP [4] is a tuple $M = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$, where S is a set of states, Act is a set of actions, $\mathbf{P} : S \times Act \times S \to [0, 1]$ is the transition probability function, $\iota_{init} : S \to [0, 1]$ is the initial distribution, AP is a set of atomic propositions (atomic propositions represent the basic properties that hold at some point of execution), $L : S \to 2^{AP}$ is a labeling function, and $L(s)$ are atomic propositions in AP satisfied in state s.

A Markov decision process is an extension of Markov chains that allows both probabilistic and nondeterministic choices. In any state, there is a nondeterministic choice between several discrete probability distributions over successor states.

3 RELATED WORK

In this subsection, a brief overview on the most related work is given.

3.1 Probabilities and Stochastic Delays in UML

Jansen et al. [20] have introduced randomness to UML statechart diagrams via enhanced statecharts, called probabilistic statecharts or P-statecharts. They have based the semantics of P-statecharts on Markov decision process models. Figure 1 shows an example of P-statechart models.

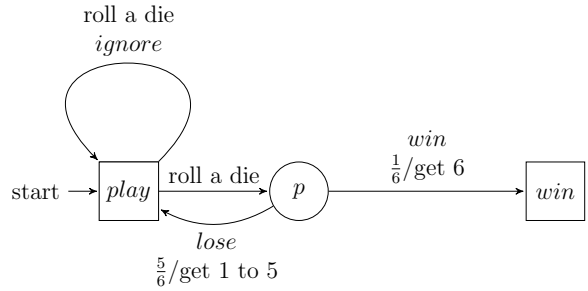


Figure 1. An example of P-statecharts

Furthermore, Jansen et al. [21] have presented StoCharts, which model random delays in statecharts. In these models, on entering a node with an outgoing edge labeled **after**(F), a sample is taken from distribution F and a timer is set accordingly. The corresponding edge becomes enabled once the timer expires. The semantics of StoCharts are defined using Stochastic I/O Automata (IOSA, for short).

3.2 Probabilities in Event-B

Hallerstede et al. [15] have extended the Event-B formalism with a new operator, qualitative probabilistic choice, denoted as $\oplus|$. The assignment $x \oplus| \ BA(v, x')$ assigns a value to x with a positive, but unknown probability. Similarly, Tarasyuk et al. [32] augmented Event-B models with the quantitative probabilistic assignment

$$x \oplus| \ x_1@p_1; \ldots; x_n@p_n$$

where $\sum_{i=1}^{n} p_i = 1$. This assignment allows for specifying exact numerical probabilities for each value x_i; variable x will have value x_i with probability p_i.

As stated before, the advantage of our approach compared to these works is that our work does not change the syntax and semantics of Event-B.

3.3 Time in Event-B

In order to model probabilistic time delays, we first need a method to model time in Event-B. In [5, 7, 28] a method for adding time to Event-B has been presented.

In this model, *time* is the current time, and *at* is the set of active times. Active times are times in future where an event might be activated. For a simple clock, *at* will be $\{time + 1, time + 2, \ldots\}$. An event called *tick_tock*, non-deterministically chooses a value for the current time, and so the progress of time is achieved. Constraint $at \neq \varnothing \Rightarrow time \leq \min(at)$ in the INVARIANT part enforces that active times are in the future, and time cannot be moved beyond the first active moment.

3.4 Probabilistic Model Checking of Event-B Models Using PRISM

Rodin [3] supports development and qualitative verification of Event-B and UML-B models, but it lacks the tools required for quantitative reasoning and verification of probabilistic systems. To enable quantitative analysis of Event-B models, one can convert Event-B models to the PRISM language. Tarasyuk et al. [34] have described the required mappings from Event-B models to the PRISM language. For example, the assignment using Tarasyuk's probabilistic choice operator $\oplus|$, (i.e. $x \oplus| \; x1@p1; \ldots xn@pn)$) [26] is expressed as the following command in PRISM:

$$[]\ true -> p1 : (x' = x1) + \ldots + pn : (x' = xn).$$

4 PROBABILISTIC UML-B

4.1 Overall Structure

In this section, in order to add abilities for modeling probabilistic and stochastic systems through the notion of state-machines in UML-B, a number of new structures are added to its graphical syntax. The corresponding semantics are also defined in Event-B.

The most basic probabilistic structure that needs to be addressed is the ability to specify discrete probabilities. This applies to scenarios where the set of possible outcomes is discrete, such as a coin toss. We take one step further and also take into account scenarios where the exact values of probabilities are unknown, but their intervals are known. We introduce a solution to model interval probabilities in this condition. Another important feature that can benefit modelers is the capability to model discrete stochastic delays, where one can specify a random amount of time before moving to the next state. Stochastic delays are present in many distributed and networking systems, and even in biological processes. We cover three types of stochastic delays: fixed time delay, uniform distribution delay and geometric distribution delay.

For every structure that we define, its semantics is also defined in Event-B. For discrete probabilities and interval probabilities, the overall approach is to update the

current state of the UML-B machine based on probabilities specified by the modeler. A random number will be generated, and the generated number will determine the target state. For stochastic delays, a timeout value is calculated upon entering a state which has an outgoing transition with a delay. The outgoing transition will have a guard to ensure a delay. The guard protects from entering the next state without first waiting in the current state for a duration of at least the calculated timeout.

The ability to generate random sequences of numbers which are uniformly distributed is essential in any work related to probabilities. Event-B does not have built-in support for generating random numbers. Therefore, we need to use an algorithm that generates sequences of numbers that are close enough to a true sequence of random numbers. Section 4.2 discusses our approach to define a random generator in Event-B. This random generator is one of the core elements of the semantics defined for most structures in this work.

Section 4.3 discusses adding discrete probabilities to the state-machine diagrams. Section 4.4 presents how to add interval probabilities to state-machines. Section 4.5 outlines stochastic delays and discusses Fixed-time, Uniform Distribution, and Geometric Distribution delays. Section 4.6 defines UML-B state-machines as MDPs to provide a theoretical basis for translating UML-B state-machines to PRISM models. Using this basis, we present the translation method in Section 4.7. The resulting PRISM models make it possible to verify probabilistic properties in the initial UML-B models, quantitatively.

4.2 The *random* Function

Throughout this section, we use the function *random* to generate uniform random numbers in the range $[l, u]$. This function can be defined using any common pseudo-random generators such as a linear congruential generator, which uses the recurrence

$$X_n = l + (aX_{n-1} + c) \mod (u - l + 1) \tag{1}$$

where X_n and X_{n-1} are respectively the next and current pseudo-random numbers, and a and c are large integer numbers. X_0 is called the seed or start value. Axioms in Figure 2 define a basic pseudo-random generator. *seed* can be a machine variable to store the last generated random value as a seed to the next iteration of the function.

4.3 Adding Discrete Probabilities to the State-Machine Diagram

We first introduce a structure for specifying discrete probabilities in UML-B. It is worth noting that only probabilities that have rational values are supported.

4.3.1 Discrete Probabilities Syntax

We propose to specify discrete probabilities using a new notion which we call pseudo-states. The difference between a pseudo-state and a normal state is that state-

AXIOMS
$random \in \mathbf{N} \times \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$
$\forall l, u, seed, r.$
$l \in \mathbf{N} \wedge u \in \mathbf{N} \wedge seed \in \mathbf{N} \wedge r \in \mathbf{N} \Rightarrow$
$(l \mapsto u \mapsto seed \mapsto r \in random \Leftrightarrow$
$r = l + (a \times seed + c) \mod (u - l + 1))$
END

Figure 2. The *random* function

machines will not stay in pseudo-states; the role of pseudo-state is only to determine how the transitions from previous states to next states are done. Figure 3 shows the new structure to specify discrete probabilities using a pseudo-state p.

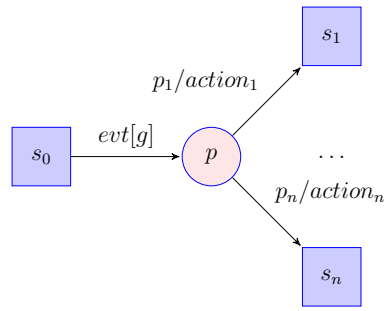


Figure 3. Discrete probabilities in UML-B

In Figure 3, if the edge with probability p_i is selected, $action_i$ will be performed. For each $i \in 1 \ldots n$, there exist numbers m_i and d, where $p_i = \frac{m_i}{d}$, and

$$d \neq 0 \wedge (m_i \in \mathbf{N}_1) \wedge \sum_{i=1}^{n} m_i = d. \tag{2}$$

Probabilities are defined by rational fractions, in form of natural numerators and denominators. If the model's probabilities are in form of $p_1 = q_1/d_1, \ldots, p_n = q_n/d_n$, the positive number d is the least common multiple of these denominators ($d = \mathbf{lcm}(d_1, \ldots, d_n)$) and $m_i = q_i \frac{d}{d_i}$.

4.3.2 Discrete Probabilities Semantics

Intuitively, semantics of the given structure in Figure 3 can be described as follows. When the state-machine is in state s_0, if the event evt is selected, and its guard g holds, the machine will enter into state s_i and will perform the correspondent $action_i$ with the probability p_i ($i \in 1 \ldots n$).

Formally, the semantics of the given structure in Event-B is defined as in Figure 4.

$$
\begin{array}{l}
evt: \\
\quad \textbf{WHERE} \\
\quad\ \mathrm{STATE} = s_0 \wedge g \\
\quad \textbf{THEN} \\
\quad\ \mathrm{STATE}, seed :| \exists r.r = random(1 \mapsto d \mapsto seed) \ \wedge \\
\quad\quad seed' = r \wedge \\
\quad\quad ((r \leq m_1 \Rightarrow \mathrm{STATE}' = s_1 \wedge action_1) \wedge \\
\quad\quad (r > m_1 \wedge r \leq m_1 + m_2 \Rightarrow \mathrm{STATE}' = s_2 \wedge action_2) \wedge \\
\quad\quad \ldots \\
\quad\quad (r > \sum_{j=1}^{i-1} m_j \wedge r \leq \sum_{j=1}^{i} m_j \Rightarrow \mathrm{STATE}' = s_i \wedge action_i) \wedge \\
\quad\quad \ldots \\
\quad\quad (r > \sum_{j=1}^{n-1} m_j \wedge r \leq \sum_{j=1}^{n} m_j \Rightarrow \mathrm{STATE}' = s_n \wedge action_n)) \\
\quad \textbf{END}
\end{array}
$$

Figure 4. Discrete probabilities in Event-B

Number r is randomly selected from numbers 1 to d. The next state will be selected as follows: If the random number r is less than or equal to m_1, the next state is s_1; if r is greater than m_1 and is less than or equal to $m_1 + m_2$, the next state is s_2 and so on.

Theorem 1. The semantics for discrete probabilities, given in Figure 4, provides the expected probabilities for the corresponding actions.

Proof. Variable r is randomly chosen from one of d numbers $(1 \ldots d)$ with equal probabilities. Now, whenever the condition:

$$r > \sum_{j=1}^{i-1} m_j \ \wedge r \leq \sum_{j=1}^{i} m_j \tag{3}$$

holds, the assignment $\mathrm{STATE} := s_i$ will be made. So, the probability of transition to s_i is:

$$\Pr\{\mathrm{STATE} := s_i\} = \frac{\sum_{j=1}^{i} m_j - \sum_{j=1}^{i-1} m_j}{d} = \frac{m_i}{d} \tag{4}$$

which is equal to the expected probability p_i. □

4.4 Interval Probabilities

Interval probabilities are used when the probabilistic design is abstract and underspecified [9, 12]. It is assumed that in the specification stage, the exact values of

probabilities are unknown, but their intervals are known, and will probably become exact in the next stages of the refinement.

4.4.1 Interval Probabilities Syntax

Similar to the discrete case, interval probabilities are specified here via a pseudo-state p. Figure 5 shows the structure we propose to model interval probabilities.

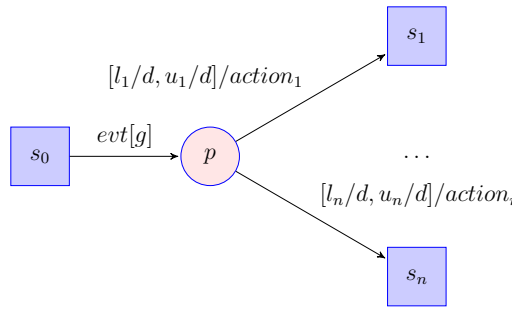


Figure 5. Interval probabilities in UML-B

In Figure 5, u_i and l_i $(i \in 1 \ldots n)$ are positive integers less than or equal to d. Intervals that the modeler chooses must allow for selecting a number from each interval such that the sum of the selected numbers is equal to 1. For example, suppose there are two branches in Figure 5. If the modeler chooses interval $[0, 0.3]$ for the first branch and interval $[0.2, 0.8]$ for the other branch, since the number 0.2 can be chosen from the first interval and 0.8 from the second one, and $0.2 + 0.8 = 1$, the selected intervals are allowed. But for $I_1 = [0, 0.3]$ and $I_2 = [0.2, 0.6]$, there are no two numbers $p_1 \in I_1$, $p_2 \in I_2$ such that $p_1 + p_2 = 1$ and therefore, these intervals are not allowed.

4.4.2 Interval Probabilities Semantics

The semantics of the structure given in Figure 5 is defined as in Figure 6. Number r is chosen from 1 to d, and each m_i $(i \in 1 \ldots n)$ is chosen from its respective interval. r will fall into one of the intervals formed by m_is and its value determines the next value for STATE.

Theorem 2. The semantics for interval probabilities, given in Figure 6, provides the expected probabilities for the corresponding actions.

Proof. We prove that the probability of moving to the state s_i will be within the specified interval.

Since there exists the condition $m_i \in l_i \ldots u_i$, the inequality

$$\frac{l_i}{d} \leq \frac{m_i}{d} \leq \frac{u_i}{d} \tag{5}$$

evt :
WHERE
STATE $= s_0 \wedge g$
THEN
STATE :| $\exists r, m_1, \ldots, m_n . r = random(1 \mapsto d \mapsto seed) \ \wedge$
$seed' = r \ \wedge$
$m_1 \in l_1 \ldots u_1 \ \wedge$
$\ldots$
$m_n \in l_n \ldots u_n \ \wedge$
$\sum_{j=1}^{n} m_j = d \ \wedge$

$((r \leq m_1 \Rightarrow \text{STATE}' = s_1 \wedge \ action_1) \ \wedge$
$(r > m_1 \wedge r \leq m_1 + m_2 \Rightarrow \text{STATE}' = s_2 \wedge action_2) \ \wedge$
$\ldots$
$(r > \sum_{j=1}^{i-1} m_j \ \wedge r \leq \sum_{j=1}^{i} m_j \Rightarrow \text{STATE}' = s_i \wedge action_i) \ \wedge$
$\ldots$
$(r > \sum_{j=1}^{n-1} m_j \ \wedge r \leq \sum_{j=1}^{n} m_j \Rightarrow \text{STATE}' = s_n \wedge action_n))$
END

Figure 6. Interval probabilities in Event-B

holds. Variable r is randomly chosen from one of d numbers $(1 \ldots d)$ with equal probabilities. Now, whenever the condition:

$$r > \sum_{j=1}^{i-1} m_j \ \wedge r \leq \sum_{j=1}^{i} m_j \tag{6}$$

holds, the assignment $\text{STATE}' = s_i$ will be made. Therefore, for the probability of the transition to s_i, the following holds:

$$\frac{l_i}{d} \leq \Pr\{\text{STATE}' = s_i\} = \frac{\sum_{j=1}^{i} m_j - \sum_{j=1}^{i-1} m_j}{d} = \frac{m_i}{d} \leq \frac{u_i}{d}. \tag{7}$$

Therefore, the probability of moving to the destination state is within the desired interval. □

4.4.3 An Alternative Method to Define the Semantics

In this subsection, one alternative semantics that can be used instead of the semantics presented in Section 4.4.2 is introduced. In Section 4.7.5, we will need this semantics to translate the probabilistic UML-B constructs to PRISM, because in PRISM it is not possible to resolve both the non-determinism and probabilities in one transition. This new semantics is defined using an additional state. We consider

an additional real state P_INTERVAL for resolving the non-determinism in probabilities, and computing discrete probabilities based on interval probabilities, and then, determining the next state by using these computed probabilities. In this way, the state-machine in Figure 5 is defined as the structure in Figure 7. The proof that the semantics given in Figure 8 provides the expected probabilities for the corresponding actions, is similar to the proof given in Theorem 2 with minor differences; so, we do not present this proof anymore. It should be noted that because of the introduction of a new real state, and the possibility of a delay, the definition given in Figure 8 is not exactly equivalent to the definition provided in Figure 6.

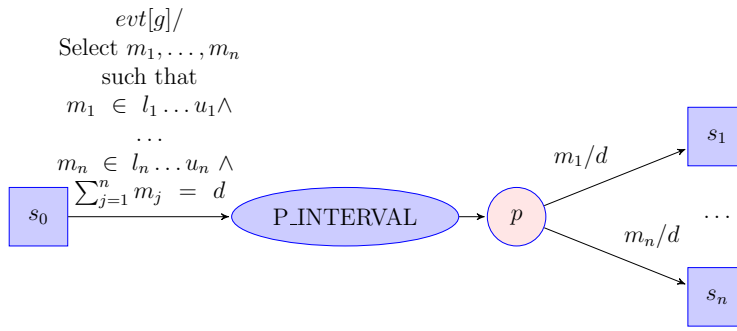


Figure 7. The semantics of interval probabilities by using an additional state

The pseudo-code for event evt is composed of evt_1 and evt_2 as shown in Figure 8; for each outgoing edge with label m_i/d, a machine variable m_i is defined.

$evt \mathrel{\hat{=}} evt_1 \ ; \ evt_2$

evt_1 :
WHERE
STATE $= s_0 \wedge g$
THEN
$m_1, \ldots, m_n :| \ m_1' \in l_1 \ldots u_1 \wedge \ldots$
$\wedge m_n' \in l_n \ldots u_n \wedge \sum_{j=1}^{n} m_j = d$
STATE := P_INTERVAL
END

evt_2 :
WHERE
STATE = P_INTERVAL
THEN
STATE $:| \ \exists r.r = random(1 \mapsto d \mapsto seed) \ \wedge$
$seed' = r \wedge$
$((r > 0 \wedge r \leq m_1 \Rightarrow \text{STATE}' = s_1 \wedge action_1) \wedge$
$(r > m_1 \wedge r \leq m_1 + m_2 \Rightarrow \text{STATE}' = s_2 \wedge action_2) \wedge$
$\ldots$
$(r > \sum_{j=1}^{i-1} m_j \wedge r \leq \sum_{j=1}^{i} m_j \Rightarrow \text{STATE}' = s_i \wedge action_i) \wedge$
$\ldots$
$(r > \sum_{j=1}^{n-1} m_j \wedge r \leq \sum_{j=1}^{n} m_j \Rightarrow \text{STATE}' = s_n \wedge action_n))$
END

Figure 8. The pseudo-code for evt as a composition of evt_1 and evt_2

4.5 Discrete Stochastic Delay

In some systems, an action may be performed when a specific amount of time is passed after reaching to a state. The duration of this delay can be fixed or can

be probabilistically selected based on a distribution function. In this subsection, the delay structure is added to UML-B. We restrict our work to discrete times and delays.

To indicate the activation of a transition after time t, guard **after**(t) is added to the UML-B syntax. In addition, for specifying the notion of probabilistic time, time t can be specified randomly. In other words, instead of **after**(t), **after**(F) is used where $F : \mathbf{N} \rightarrow [0, 1]$ is the distribution function of timeout. For delays corresponding to the geometric distribution with parameter $p = \frac{m}{d}$, and for delays with the uniform distribution, **after**$(\mathrm{G}(m/d))$ and **after**$(\mathrm{UNIF}(t_{min}, t_{max}))$ are respectively used, where, t_{min} and t_{max} are minimum and maximum values of delay. We restrict our work to the uniform and geometric distributions. We also consider distributions that have a random generator function. After passing the delay time, one of the output edges will be activated and the machine will transit to one of the target nodes.

Since Event-B does not have the notion of time, the method presented in [28] and reviewed in Section 3.3, is used to express the concept of time. The auxiliary variables and events in Figure 9 are added to the Event-B model to handle the notion of time:

VARIABLES
$time, at$
INVARIANT
$time \in 0 \ldots MAX\_TIME \ \wedge$
$at \subseteq MAX\_TIME \ \wedge$
$(at \neq \varnothing \Rightarrow time \leq \min(at))$
INITIALISATION
$time := 0$
$at := 0 \ldots MAX\_TIME$
EVENTS
$tick\_tock$:
ANY
tm
WHERE
$tm \in MAX\_TIME \wedge$
$tm > time \ \wedge$
$(at \neq \varnothing \Rightarrow tm \leq \min(at))$
THEN
$time := tm$
END

$process\_time$:
WHERE
$time \in at$
THEN
$at := at - time$
END
END

Figure 9. Auxiliary variables and events for handling time in Event-B

In Figure 9, variable *time* is the current value of time, and *at* is the remaining active times at which the *process_time* event will be triggered. The *tick_tock* event increases the value of *time*, and the *process_time* event removes the current value of *time* from active times. To avoid state explosion and keeping the model finite, times are restricted by constant *MAX_TIME*.

In the next subsections, we will use variable *time* to compute the timeout moment and also to obtain the time at which an event with a guard containing a delay can be executed.

4.5.1 Fixed Time Delay

In this subsection, at first the syntax of the fixed time delay is introduced in UML-B, and then, its semantics is presented.

a. Fixed Time Delay Syntax. The structure specified in Figure 10 is added to UML-B. The parameter $t \in \mathbf{N}$ is the amount of delay before actions of *evt* can be executed.

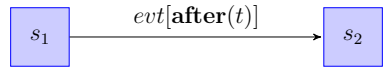


Figure 10. The fixed time delay in UML-B

b. Fixed Time Delay Semantics. The structure specified in Figure 10 is translated to the structure in Figure 11. We define T as the set of all transitions that their destinations are the starting state s_1 (e.g. STATE := s_1) in the machine. Every such statement must be accompanied by a parallel assignment $timeout := time + t$ to specify the amount of timeout.

For every transition, starting from the state s_1 that has an $after$ condition, the guard $time \geq timeout$ must be added. If there are more than one $after$ condition, different *timeout* variables must be defined (i.e. $timeout_0, timeout_1, \ldots$), and for each $after$ condition, the corresponding variable must be used. All these *timeout* variables should be initialized at the moment that STATE := s_0 is being done.

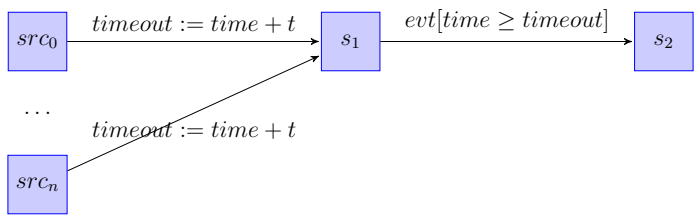


Figure 11. The structure equivalent to the syntax in Figure 10

The events in Figure 11 are defined as in Figure 12.

Theorem 3. The semantics for fixed time delay, given in Figure 12, provides the expected delays for the corresponding actions.

Proof. We prove that the lower bound of the delay for the transition STATE := s_2 is the constant t. If at any moment in the interval $[t_0, t_0 + t)$ (t_0 is the moment that the timeout is set), the event *evt* is chosen for execution, the guard $time \geq timeout = t_0 + t$, which is necessary for moving to the state s_2, will not hold and the transition will not be done. At the time $t_0 + t$, the evaluation of this guard will change to *true*. Therefore, the lower bound of the transition to the destination state is $t_0 + t - t_0 = t$. □

All transitions STATE := s_1 in T will be accompanied by an assignment setting the timeout:

STATE := s_1
$timeout := time + t$

evt :
WHERE
STATE = $s_1 \wedge time \geq timeout$
THEN
STATE := s_2
END

Figure 12. The fixed time delay in Event-B

4.5.2 Uniform Distribution Delay

In this subsection, at first the syntax of the uniform distribution time delay is introduced in UML-B, and then, its semantics is presented.

a. Uniform Distribution Delay Syntax. The structure in Figure 13 is added to UML-B. The parameters $U, L \in \mathbf{N}$ are the lower and upper bounds of delay before actions of *evt* can be executed.

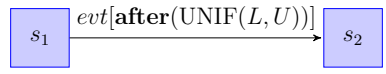


Figure 13. The uniform distribution delay in UML-B

b. Uniform Distribution Delay Semantics. Intuitively, the diagram in Figure 13 expresses that when the state-machine is in s_1, it will move to s_2 with a delay which equals to a random time selected uniformly in the interval L to U.

For defining the semantics formally, the action $timeout := time + t$ in Figure 12 is changed to the following action:

$$timeout, seed :| \exists r.r = random(L \mapsto U \mapsto seed) \wedge \\ timeout' = time + r \ \wedge \ seed' = r. \tag{8}$$

Theorem 4. The semantics for uniform distribution delay, given in Figure 11 with the change mentioned in Equation (8), provides the expected delays for the corresponding actions.

Proof. We prove that the lower bound of the delay for the transition STATE $:= s_2$ is a random number from the interval $[L, U]$. We designate t_0 to the moment the variable *timeout* is evaluated. At t_0, *timeout* will be assigned a random value in the interval $[t_0 + L, t_0 + U]$. If at any moment in the interval $[t_0, timeout)$, the event *evt* is chosen for execution, the guard $time \geq timeout$, which is necessary for moving to the state s_2, will not hold and the transition will not be done. At the time $t_0 + timeout$, the evaluation of this guard will change to *true*. Therefore, the lower bound of the delay is an integer number in $[L, U]$. □

4.5.3 Geometric Distribution Delay

Many continuous-time systems exhibit delays with exponential distribution. The exponential distribution describes the time for a continuous process to change state or an event to occur. The discrete analog for exponential distribution is the geometric distribution. This distribution describes the number of Bernoulli trials needed for the first success. Therefore, the geometric distribution can be seen as describing the number of steps a discrete process needs to change state. If X is a geometrically distributed random variable, and the probability of success on each trial is p, the probability of $X = k, k \in \mathbf{N}_1$ is $(1-p)^{k-1}p$. In this subsection we propose a syntax and semantics for the geometric delay.

a. Geometric Distribution Delay Syntax. Figure 14 shows the geometric distribution delay structure in UML-B. The fraction m/d is the parameter of the geometric distribution.

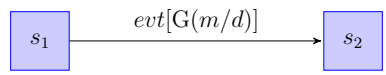


Figure 14. Geometric distribution delay in UML-B

b. Geometric Distribution Delay Semantics. The semantics for the geometric distribution is similar to the one for the uniform distribution except that it uses the function *geometric* instead of *random*. The main difficulty lies in the problem of generating numbers from the geometric distribution. We define the function

geometric as an axiom to generate numbers from the geometric distribution. Since the seed for the *random* function needs to be different for each iteration of *random*, we use recursion and ideas from dynamic programming for storing the previous values of the *random* function in the function *rand*. Figure 15 shows the definition of the function *geometric*.

The semantics is defined by changing the action $timeout := time + t$ in Figure 12 to the following action:

$$\begin{aligned} timeout, seed :| \ & \exists r, g.g \mapsto r = geometric(m \mapsto d \mapsto seed) \wedge \\ & timeout' = time + g \ \wedge \ seed' = r. \end{aligned} \tag{9}$$

AXIOMS

$\forall n, X, rand, m, d, seed, g, newSeed.$
$n \in 0 \ldots MAX\_TIME \wedge X \in \mathbf{N} \to 0 \ldots 1 \wedge m \in \mathbf{N} \wedge d \in \mathbf{N} \wedge$
$rand \in \mathbf{N} \to \mathbf{N} \wedge seed \in \mathbf{N} \wedge newSeed \in \mathbf{N} \wedge g \in 0 \ldots MAX\_TIME \Rightarrow$
$rand(0) = random(1 \mapsto d \mapsto seed) \wedge$
$rand(n) = random(1 \mapsto d \mapsto rand(n-1)) \wedge$
$((rand(n) \leq m \Leftrightarrow X(n) = 0) \wedge$
$(rand(n) > m \Leftrightarrow X(n) = 1)) \wedge$
$g = \min(\{j | j \in 0 \ldots MAX\_TIME \wedge j \mapsto 1 \in X\}) \wedge$
$newSeed = rand(g) \wedge$
$g \mapsto newSeed = geometric(m \mapsto d \mapsto seed)$

END

Figure 15. The definition of the *geometric* function

Theorem 5. The function *geometric* generates numbers from the geometric distribution.

Proof. The function *rand* is a sequence of randomly generated numbers. The function X is a sequence of 0 or 1s, with success probability $p = \frac{m}{d}$. X can be considered as a sequence of results of independent Bernoulli trials. The set $S = \{j | j \in 0 \ldots MAX\_TIME \wedge j \mapsto 1 \in X\}$ is the set of indices of all successful Bernoulli trials, and $g = \min(S)$ is the index of the first successful Bernoulli trial. For any j ($j \in 0 \ldots MAX\_TIME$), the probability of $g = j$ equals to $(1-p)^j p$. Therefore, *geometric* generates numbers with geometric distribution with parameter $p = \frac{m}{d}$. □

Theorem 6. The semantics for geometric distribution delay, given in Figure 11 with the change mentioned in Equation (9), provides the expected delays for the corresponding actions.

Proof. We prove that the lower bound of the delay for the transition STATE := s_2 is a random number with the geometric distribution. We designate t_0 to the moment the variable *timeout* is evaluated. At t_0, *timeout* will be assigned the value $t_0 + geometric(\frac{m}{d})$. As mentioned above, values of $geometric(\frac{m}{d})$ have the geometric distribution with parameter $p = \frac{m}{d}$. If at any moment in the interval $[t_0, timeout)$, the event *evt* is chosen for execution, the guard $time \geq timeout$, which is necessary for moving to the state s_2, will not hold and the transition will not be done. At the time $t_0 + timeout$, the evaluation of this guard will change to *true*. Therefore, the lower bound of the delay for the transition to the destination state is a number with the geometric distribution. □

4.5.4 Generalization – Arbitrary Discrete Distribution

For random delays with arbitrary distributions, if the generator function F is available, one can model the delay through a method similar to that of the previous subsection. It is sufficient to replace the timeout assignment in Figure 12 with the appropriate assignment.

4.6 UML-B State-Machine as a Probabilistic Transition System

In this subsection, the state-machine models are defined as MDPs. The objective is to have a theoretical ground for translating UML-B state-machines to PRISM models for quantitative and probabilistic model checking. The provided definition is similar to the definitions in [10] and [33] in which Event-B models are defined as Transition Systems. In addition to standard Event-B structures and assignments, defined in [10] and [33], our definition takes into account probabilities and the current state in UML-B state-machines.

In order to define UML-B state-machines as MDPs, we need to slightly change the definition of the projection operator π [13] to extract a component of a tuple by its name, not its index.

Definition 1. Let $A_1, \ldots, A_n$ be sets, and $i \in 1 \ldots n$. If $T \subseteq A_1 \times \ldots \times A_n$, then function $\pi_i : T \to A_i$ is defined by $\pi_i(a_1, \ldots, a_n) = a_i$. If the i^{th} component of tuple $(a_1, \ldots, a_n)$ is denoted by variable v, we define $\pi_v(a_1, \ldots, a_n) = a_i$.

Definition 2. Every UML-B state-machine model is defined as tuple $M = (S, E, \mathbf{P}, \iota_{init}, AP, L)$ where:

- S is the set of states of the state-machine. Let $v_1, \ldots, v_n$ be the variables of the machine. We define V_i to be the type of variable v_i $(i \in 1 \ldots n)$. Therefore, $S = V_1 \times \ldots \times V_n$. The state-machine has at least one variable STATE which specifies the current state of the state-machine.
- $E \subseteq S \times S$ is the set of all events of the state-machine. Any event $evt(s)$ is defined as:

$$evt(s) : \textbf{WHERE } G_{evt}(s) \textbf{ THEN } R_{evt}(s) \textbf{ END}$$

where s is the current state of the state-machine, $G_{evt} : S \to BOOL$ is the event guard, and $R_{evt} \subseteq S \times S$ determines the relationship between the current state and the next state (by one or multiple assignments). For example, for the following event:

$$evt : \textbf{WHERE } x = 1 \textbf{ THEN } x := 0 \textbf{ END}.$$

We have:

$$\forall s.s \in S \Rightarrow (\pi_x(s) = 1 \Leftrightarrow G_{evt}(s) = true),$$
$$\forall s, s'.s, s' \in S \Rightarrow (\pi_x(s') = 0 \Leftrightarrow (s, s') \in R_{evt}).$$

In a state-machine, there is an edge from SM_STATE to SM_STATE' (which are two states in the state-machine) if and only if:

$$\begin{aligned} \exists s, s'.G_{evt}(s) \wedge I(s) \wedge SM\_STATE = \pi_{\text{STATE}}(s) \wedge \\ (s, s') \in R_{evt} \wedge SM\_STATE' = \pi_{\text{STATE}}(s') \wedge I(s') \end{aligned} \tag{10}$$

where $I : S \to BOOL$ is the machine invariant. This predicate means there exists a transition from SM_STATE to SM_STATE', if and only if the invariant holds in states s and s', STATE is equal to SM_STATE, and s is R_{evt}-related to s'.

- $\mathbf{P} : S \times E \times S \to [0, 1]$ is the transition probability function:

$$\mathbf{P}(s, evt, s') = \begin{cases} 0, & \neg(I(s) \wedge G_{evt}(s) \wedge I(s') \wedge (s, s') \in R_{evt}), \\ p_{evt}(s, s'), & I(s) \wedge G_{evt}(s) \wedge I(s') \wedge (s, s') \in R_{evt} \end{cases} \tag{11}$$

  where $p_{evt}(s, s')$ is the probability of going from state s to s'.
- $\iota_{init} : S \to [0, 1]$ is the initial distribution. This is determined using the INITIALIZATION statements of the machine and determines the probability of being at various states of the machine at time 0.
- $AP = \varnothing$ is the set of atomic propositions. Since we are not labeling our MDP states, the set of atomic propositions is empty.
- $\forall s \in S. L(s) = \varnothing$. We simply choose not to label any MDP state.

In state s, event evt is selected according to the transition probability function $\mathbf{P}$, and the machine is transited to state s' according to $p_{evt}(s, s')$.

By this definition, every UML-B state-machine model can be described as a Markov Decision Process.

4.7 Translating Probabilistic UML-B State-Machines to PRISM

The PRISM tool can receive its inputs as MDP models. As indicated in Section 4.6, a state-machine in UML-B can be interpreted as an MDP. In this subsection, we

use this correspondence to present a method for translating a UML-B state-machine model to a PRISM model for the purpose of quantitative model checking. To automate the process of translating from UML-B state-machines to PRISM models, a number of straightforward conversions are presented. The proposed method is similar to the method presented in [34], in which a number of conversions from Event-B to PRISM are shown, including translating actions containing the $\oplus|$ operator. The conversions proposed in this section can be performed indirectly through the methods of the aforementioned work to translate models to PRISM models. But, we adapt the conversions to the specific structures present in UML-B and our probabilistic extension of it, to achieve a more readable and clear final PRISM model.

In what follows, we present the needed translation for each UML-B construct in order to translate a UML-B model to a PRISM model.

4.7.1 State-Machine and Its States

Let STATEMACHINE be the name of the state-machine, and its states be S1 to Sn; now, variable SM_STATE of type $[0 \dots n-1]$ is defined in PRISM. Constants $s1$ to sn with values 0 to $n-1$ are also defined to specify the different states of the machine. The initial state of the state-machine is defined in the **init** section (Figure 16).

const int $s1 = 0$;
$\dots$
const int $sn \ = n-1$;
global SM_STATE $[s1 \dots sn]$ **init** $s1$;

Figure 16. State-machine states in PRISM

For each state-machine, a module with the name of that state-machine is defined to model its transitions (i.e. **module** statemachine $\dots$ **endmodule**). The body of a module will be the transitions taking place in the module.

4.7.2 State Transition

Figure 17 indicates a state transition in UML-B and its translation in PRISM. Guard SM_STATE $= s1$ determines if the model is in $s1$. Transition SM_STATE$' = s2$ changes the current state.

When a transition also has guards and actions, they will be included, too (Figure 18).

Figure 17. Left: A state transition in UML-B. Right: Its translation in PRISM.

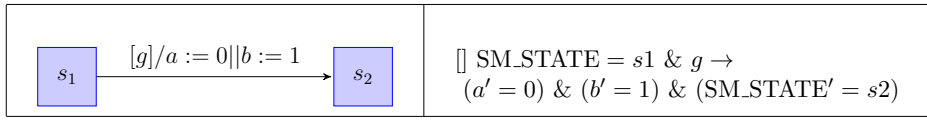


Figure 18. Left: A state transition with guards and actions. Right: Its translation in PRISM.

4.7.3 Nondeterministic Transition

Nondeterministic transitions which have the same guards are translated as shown in Figure 19.

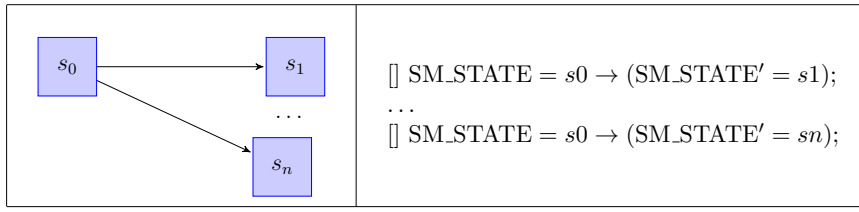


Figure 19. Left: Nondeterministic transitions. Right: Their translation in PRISM.

4.7.4 Discrete Probabilities

In order to translate the structure shown in Figure 20, the probabilistic selection in PRISM is used.

4.7.5 Interval Probabilities

We take the approach discussed in Section 4.4.3 and Figure 7 to perform the translation of interval probabilities to PRISM. Variable INTERVALP is used for maintaining the state to which the state-machine will move after the probabilistic selection. Variables m_1 to m_n are defined to keep each transition probability's fraction's numerator. Every combination of $m_1 + \ldots + m_n$, where $L_1 \leq m_1 \leq U_1, \ldots, L_n \leq m_n \leq U_n$, is considered; and $m_1, \ldots, m_n$ are assigned non-deterministically. d is the denominator of probabilities fractions.

In order to generate all valid transitions, for each numerator variable m_i ($i \in 1 \ldots n$), a value is taken from its corresponding interval $L_i \ldots U_i$. For example, the first transition shown in Figure 21 takes the lower bound of each numerator variable

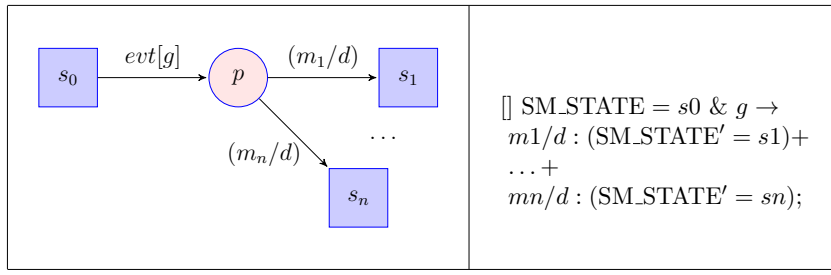


Figure 20. Left: A probabilistic transition in UML-B. Right: Its PRISM counterpart.

[] SM_STATE = $s0$ & $(L1 + \ldots + Ln = d) \rightarrow$
$(m1' = L1) \& \ldots \& (mn' = Ln) \&$
(SM_STATE′ = INTERVALP);
[] SM_STATE = $s0$ & $((L1 + 1) + \ldots + Ln = d) \rightarrow$
$(m1' = L1 + 1) \& \ldots \& (mn' = Ln) \&$
(SM_STATE′ = INTERVALP);
. . .
[] SM_STATE = $s0$ & $(U1 + \ldots + Un = d) \rightarrow$
$(m1' = U1) \& \ldots \& (mn' = Un) \&$
(SM_STATE′ = INTERVALP);

[] SM_STATE = INTERVALP & $m1 + \ldots + mn = d \rightarrow$
$m1/d$: (SM_STATE′ = $s0$)+
. . .
mn/d : (SM_STATE′ = sn);

Figure 21. Interval probabilities in PRISM

as its value, and adds a guard to check if $\sum_i m_i$ equals to d. From those transitions for which this guard is true, one is chosen non-deterministically and SM_STATE will move to INTERVALP. The last transition in Figure 21 computes the probability of moving to every state and performs the transition of SM_STATE.

4.7.6 Time

A module named *tick_tock* is defined for advancing the integer variable time. The transition [] $time < MAX\_TIME- > (time' = time + 1)$; is used for advancing time. Constant *MAX_TIME* is used to limit the possible values of variable time to avoid state explosion.

4.7.7 Fixed Time Delay

In every ingoing transition to a state in which a fixed time delay exists, the integer variable $after\_time$ is set to $time + t$, where t is the fixed delay. In the ongoing transitions that include guard **after**(t), condition $time >= after\_time$ is added to make sure those transitions are activated only when the specified time has passed.

4.7.8 Uniform Distribution Delay

This case is similar to the fixed time delay, but the *after_time* variable is set randomly. In the ingoing transitions to the state in which condition **after**(UNIF(L, U)) exists, *after_time* is set to one of the values $time + L$, $time + (L + 1)$, ..., $time + U$, with equal probability.

4.7.9 Geometric Distribution Delay

Because of the lack of recursive function support in the current version of PRISM, the translation is not yet possible.

5 CASE STUDY

In this section the applicability of the probabilistic extension of UML-B is illustrated through a case study.

5.1 Zeroconf Configuration Protocol

In this case study, we consider the Zeroconf configuration protocol for local addresses [25]. This protocol configures an IP address for a newly joined device to the local network. When a host connects to the network, it first randomly selects an IP address from a pool of 65 024 available addresses in the range of 169.254.1.0 to 169.254.254.255. The host waits for a random time between 0 and 2 seconds before starting to send four Address Resolution Protocol (ARP) packets, called probes, to all other hosts. These probes contain the IP address selected by the host, and are sent at 2 seconds intervals. A host which is already using this address will respond with an ARP reply packet, and the original host will restart reconfiguration. If the host encounters 10 IP conflicts, it remains idle for 1 minute. If the host sends four probes without receiving any ARP reply packet, then it starts to use the chosen IP address. This host sends two further messages, called gratuitous ARPs, at 2 seconds intervals. A host that has started using an IP address must reply to ARP packets containing the same IP address. It continues to use the address unless it receives a gratuitous ARP containing the same IP address. In this case, the host can either defend its IP address, or defer to the conflicting host. The host may only defend its address if it has not received a previous conflicting packet within the previous ten seconds; otherwise, it must defer. A defending host sends an ARP packet containing the IP address. A deferring host restarts the protocol and reconfigures.

5.2 Modeling the Zeroconf Protocol in UML-B

We consider one host, which is trying to configure its IP address in a network of N other hosts. If the number of all available IP addresses is IP, then the probability of the host choosing a fresh IP address is $(IP - N)/IP$. Possible values for IP addresses are abstracted to values 1 and 2. Value 1 represents an IP address already assigned to a host in the network. Value 2 represents a fresh IP address. Also, the delay over the interval $[0, 2]$ is abstracted to a choice over $\{0, 1, 2\}$.

Figure 22 shows the UML-B state-machine model for the host. In the RECONF state, the host chooses a new IP address denoted by *iph*, by moving to the CHOOSE state. If it has encountered 10 address conflicts, it moves to the CHOOSEWAIT state and chooses a new address after waiting for one minute. In states CHOOSE and CHOOSEWAIT, the host probabilistically selects an address. The random delay before sending probes is modeled using the after structure. In order to model probabilities and random delays, the methods discussed in Section 4 are used. In state WAITSP, the host sends K probes before moving to the WAITSG state. In this state, the host sends two more ARPs before moving to the USE state and using the selected IP address. If, while in WAITSG, the host receives a packet with the same IP address, the host moves to RESPOND. For further details refer to [25].

The model for the network is shown in Figure 23. If the network is in the IDLE state, it moves to the NET_SEND state after a delay of 0 or 1 second and probabilistically selects the value 0 or 1 for the IP address sent by one of the hosts of the network, denoted by *ip*.

Figure 23 shows the model for time. This is a simplified model that only advances *time* by one. At any time, the action $time := time+1$ is chosen nondeterministically among other active UML-B events.

5.3 Translating the Model to PRISM

Using the method presented in Section 4.7, the created UML-B model was translated to a PRISM model. The resulting model is an MDP with three modules, namely host, network and time. A number of constants have been defined to represent different states. In each module, a variable holds the current state. Different transitions start with a guard checking the current state.

The resulting PRISM model can be used to verify a number of probabilistic properties about our model. For instance, the property saying "the maximum probability that the host finally chooses value 2 for *iph* and moves to state USE" is expressed in PCTL as $P_{max}(\Diamond((host\_state = USE) \wedge iph = 2)))$.

6 CONCLUSION AND FUTURE WORK

We have added abilities for modeling probabilistic and random systems in UML-B. A number of new structures have been added to the graphical syntax of UML-B, and

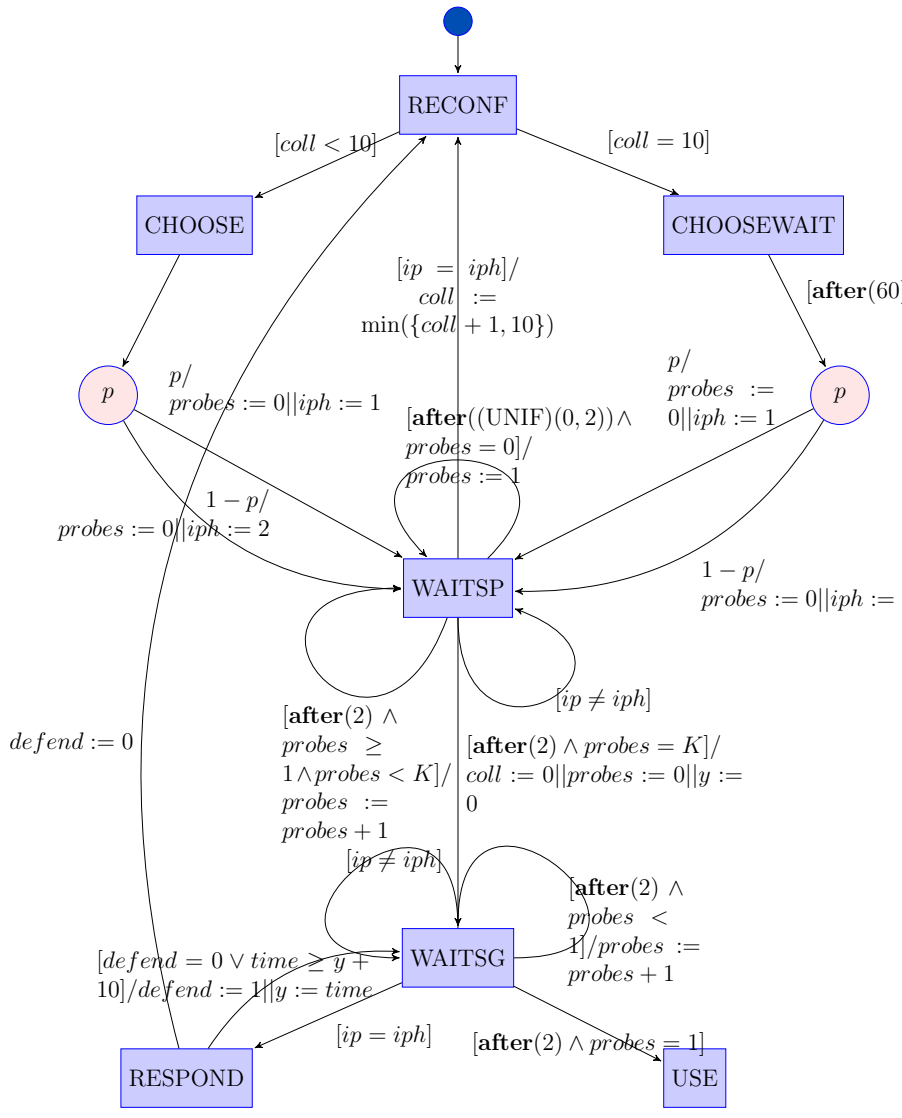


Figure 22. The UML-B state-machine model for the host component of the Zeroconf protocol

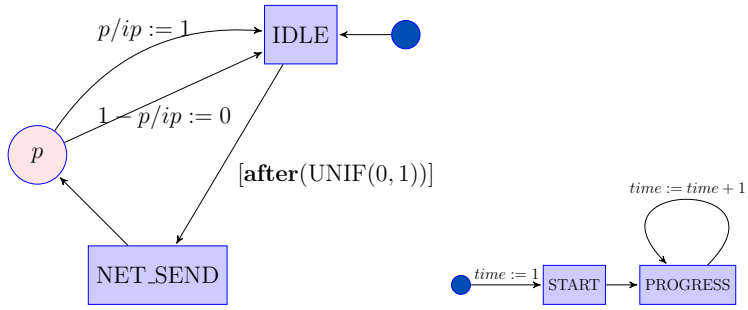


Figure 23. The UML-B state-machine model for the network and time components of the Zeroconf protocol

their semantics has been defined in Event-B. In addition, a method for translating UML-B models into PRISM language has been presented in order to perform quantitative and probabilistic model checking. To show the applicability of the proposed method, a case study on Zeroconf protocol was presented.

In future work, to increase the mathematical rigor of the proposed extensions, we would like to introduce rules and proof obligations for refinement of the proposed probabilistic structures. Furthermore, the proposed methods for translations and conversions need an automatic tool.

REFERENCES

[1] Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, 2005.

[2] Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010, doi: 10.1017/cbo9781139195881.

[3] Abrial, J.-R.—Butler, M.—Hallerstede, S.—Hoang, T. S.—Mehta, F.—Voisin, L.: Rodin: An Open Toolset for Modelling and Reasoning in Event-B. International Journal on Software Tools for Technology Transfer (STTT), Vol. 12, 2010, No. 6, pp. 447–466, doi: 10.1007/s10009-010-0145-y.

[4] Baier, C.—Katoen, J.-P.—Larsen, K. G.: Principles of Model Checking. MIT Press, 2008.

[5] Butler, M.—Falampin, J.: An Approach to Modelling and Refining Timing Properties in B. Refinement of Critical Systems (RCS), 2002.

[6] Cansell, D.—Méry, D.: The Event-B Modeling Method: Concepts and Case Studies. In: Bjørner, D., Henson, M. C. (Eds.): Logics of Specification Languages. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2008, pp. 47–152.

[7] Cansell, D.—Méry, D.—Rehm, J.: Time Constraint Patterns for Event B Development. In: Julliand, J., Kouchnarenko, O. (Eds.): B 2007: Formal Specification and

Development in B (B 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4355, 2006, pp. 140–154, doi: 10.1007/11955757_13.

[8] Corin, R.—Den Hartog, J.: A Probabilistic Hoare-Style Logic for Game-Based Cryptographic Proofs. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (Eds.): Automata, Languages and Programming (ICALP 2006). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4052, 2006, pp. 252–263, doi: 10.1007/11787006_22.

[9] Delahaye, B.—Larsen, K. G.—Legay, A.—Pedersen, M. L.—Wąsowski, A.: Decision Problems for Interval Markov Chains. In: Dediu, A. H., Inenaga, S., Martín-Vide, C. (Eds.): Language and Automata Theory and Applications (LATA 2011). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6638, 2011, pp. 274–285, doi: 10.1007/978-3-642-21254-3_21.

[10] Dotti, F. L.—Iliasov, A.—Ribeiro, L.—Romanovsky, A.: Modal Systems: Specification, Refinement and Realisation. In: Breitman, K., Cavalcanti, A. (Eds.): Formal Methods and Software Engineering (ICFEM 2009). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5885, 2009, pp. 601–619, doi: 10.1007/978-3-642-10373-5_31.

[11] Berthomieu, B.—Bodeveix, J.-P.—Farail, P.—Filali, M.—Garavel, H.—Gaufillet, P.—Lang, F.—Vernadat, F.: Fiacre: An Intermediate Language for Model Verification in the TOPCASED Environment. European Congress on Embedded Real-Time Software (ERTS), 2008.

[12] Fecher, H.—Leucker, M.—Wolf, V.: Don't Know in Probabilistic Systems. In: Valmari, A. (Ed.): Model Checking of Software (SPIN 2006). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3925, 2006, pp. 71–88, doi: 10.1007/11691617_5.

[13] Grimaldi, R. P.: Discrete and Combinatorial Mathematics: An Applied Introduction. 5th ed. Pearson, 2003.

[14] Güdemann, M.—Lipaczewski, M.—Struck, S.—Ortmeier, F.: Unifying Probabilistic and Traditional Formal Model Based Analysis. 8th Dagstuhl-Workshop MBEES 2012 – Model-Based Development of Embedded Systems, 2012.

[15] Hallerstede, S.—Hoang, T. S.: Qualitative Probabilistic Modelling in Event-B. In: Davies, J., Gibbons, J. (Eds.): Integrated Formal Methods (IFM 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4591, 2007, pp. 293–312, doi: 10.1007/978-3-540-73210-5_16.

[16] Hansson, H.—Jonsson, B.: A Logic for Reasoning about Time and Reliability. Formal Aspects of Computing, Vol. 6, 1994, No. 5, pp. 512–535, doi: 10.1007/bf01211866.

[17] Hasan, O.—Tahar, S.: Formal Probabilistic Analysis: A Higher-Order Logic Based Approach. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (Eds.): Abstract State Machines, Alloy, B and Z (ABZ 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5977, 2010, pp. 2–19, doi: 10.1007/978-3-642-11811-1_2.

[18] Hoang, T. S.: The Development of a Probabilistic B-Method and a Supporting Toolkit. Ph.D. thesis, The University of New South Wales, 2005.

[19] Hurd, J.: Formal Verification of Probabilistic Algorithms. Technical Report UCAM-CL-TR-566, University of Cambridge, Computer Laboratory, 2003.

[20] Jansen, D. N.—Hermanns, H.—Katoen, J.-P.: A Probabilistic Extension of UML Statecharts. In: Damm, W., Olderog, E. R. (Eds.): Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 2002). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2469, 2002, pp. 355–374, doi: 10.1007/3-540-45739-9_21.

[21] Jansen, D. N.—Hermanns, H.—Katoen, J.-P.: A QoS-Oriented Extension of UML Statecharts. In: Stevens, P., Whittle, J., Booch, G. (Eds.): "UML" 2003 – The Unified Modeling Language. Modeling Languages and Applications (UML 2003). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2863, 2003, pp. 76–91, doi: 10.1007/978-3-540-45221-8_7.

[22] Kwiatkowska, M.—Norman, G.—Parker, D.: Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach. In: Katoen, J. P., Stevens, P. (Eds.): Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2280, 2002, pp. 52–66, doi: 10.1007/3-540-46002-0_5.

[23] Kwiatkowska, M.—Norman, G.—Parker, D.: Advances and Challenges of Probabilistic Model Checking. 2010 48$^{\text{th}}$ Annual Allerton Conference on Communication, Control, and Computing (Allerton), IEEE, 2010, pp. 1691–1698, doi: 10.1109/allerton.2010.5707120.

[24] Kwiatkowska, M.—Norman, G.—Parker, D.: Prism 4.0: Verification of Probabilistic Real-Time Systems. In: Gopalakrishnan, G., Qadeer, S. (Eds.): Computer Aided Verification (CAV 2011). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6806, 2011, pp. 585–591, doi: 10.1007/978-3-642-22110-1_47.

[25] Kwiatkowska, M.—Norman, G.—Parker, D.—Sproston, J.: Performance Analysis of Probabilistic Timed Automata Using Digital Clocks. Formal Methods in System Design, Vol. 29, 2006, No. 1, pp. 33–78, doi: 10.1007/978-3-540-40903-8_9.

[26] Lopatkin, I.—Iliasov, A.—Romanovsky, A.—Prokhorova, Y.—Troubitsyna, E.: Patterns for Representing FMEA in Formal Specification of Control Systems. 2011 IEEE 13$^{\text{th}}$ International Symposium on High-Assurance Systems Engineering (HASE), IEEE, 2011, pp. 146–151, doi: 10.1109/hase.2011.10.

[27] Reggio, G.—Wieringa, R. J.: Thirty One Problems in the Semantics of UML 1.3 Dynamics. Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '99), 1999.

[28] Rehm, J.: A Method to Refine Time Constraints in Event B Framework. Automatic Verification of Critical Systems (AVoCS 2006), 2006, pp. 173–177.

[29] Rumbaugh, J.—Jacobson, I.—Booch, G.: Unified Modeling Language Reference Manual. Pearson Higher Education, 2004.

[30] Said, M. Y.—Butler, M.—Snook, C.: Class and State Machine Refinement in UML-B. Proceedings of Workshop on Integration of Model-Based Formal Methods and Tools (associated with IFM 2009), 2009.

[31] Snook, C.—Butler, M.: UML-B and Event-B: An Integration of Languages and Tools. Proceedings of the IASTED International Conference on Software Engineering, 2008, pp. 336–341, doi: 10.1145/1125808.1125811.

[32] Tarasyuk, A.—Troubitsyna, E.—Laibinis, L.: Towards Probabilistic Modelling in Event-B. In: Méry, D., Merz, S. (Eds.): Integrated Formal Methods (IFM 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6396, 2010, pp. 275–289, doi: 10.1007/978-3-642-16265-7_20.

[33] Tarasyuk, A.—Troubitsyna, E.—Laibinis, L.: Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B. In: Derrick, J., Gnesi, S., Latella, D., Treharne, H. (Eds.): Integrated Formal Methods, 2012 (IFM 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7321, pp. 237–252, doi: 10.1007/978-3-642-30729-4_17.

[34] Tarasyuk, A.—Troubitsyna, E.—Laibinis, L.: Quantitative Reasoning about Dependability in Event-B: Probabilistic Model Checking Approach. In: Petre, L., Sere, K., Troubitsyna, E. (Eds.): Dependability and Computer Engineering: Concepts for Software-Intensive Systems. IGI Global, 2012, pp. 459–472, doi: 10.4018/978-1-60960-747-0.ch019.

Mohammad Nosrati is a Ph.D. student in software engineering at Shahid Beheshti University, Tehran, Iran. He holds a master's degree in software engineering from Shahid Beheshti University. His research interests are software testing, formal methods, machine learning and image processing.

Hassan Haghighi received his Ph.D. in computer engineering from Sharif University of Technology. He is Associate Professor at the Faculty of Computer Science and Engineering in Shahid Beheshti University, Tehran, Iran. His research focus is on software testing, formal methods, and software architecture.

Computing and Informatics, Vol. 38, 2019, 115–150, doi: 10.31577/cai_2019_1_115

NETWORKED ONTOLOGIES WITH CONTEXTUAL ALIGNMENTS

Sihem Klai

LabGed, Department of Computer Science
University of Badji Mokhtar of Annaba
PO Box 12, 2300, Algeria
e-mail: klai@labged.net

Antoine Zimmermann

École Nationale Supérieure des Mines
FAYOL-ENSMSE, LSTI, F-42023
Saint-Étienne, France
e-mail: antoine.zimmermann@emse.fr

Mohamed Tarek Khadir

LabGed, Department of Computer Science
University of Badji Mokhtar of Annaba
PO Box 12, 2300, Algeria
e-mail: khadir@labged.net

Abstract. The problem of knowledge heterogeneity in the Semantic Web or in the context of information systems remains a major challenge for the scientific community, in particular when several ontologies developed independently and separately have to be exploited to exchange their knowledge. Several works have addressed the semantic heterogeneity issue in ontologies and proposed to align them with additional knowledge. Recently a formalism taking into account the challenge of applied techniques to represent and reason on aligned ontologies was proposed by the authors. The authors proposed a contribution that can be seen as an extension of existing work on the heterogeneous ontologies integration. This formalism allows dealing with contextual representation and reasoning where ontologies and alignments by pairs of ontologies are developed in different and incompatible context. In this paper, some aspects of multi-level networked knowledge are recalled, detailing its semantics and discussing the comparison of the two semantics, DL-approach and

DDL-approach, according to certain criteria, in order to measure their relevance and to give to readers a way to choose one semantics rather than another according to the context or the intended application.

Keywords: Networked knowledge semantics, contextual ontologies, contextual alignments

1 INTRODUCTION

Recently, a Multi-Level Networked Knowledge (MLNK) formalism was proposed to allow contextualization of alignment representation [25, 24]. This formalism attempts to solve the problem of alignments semantic heterogeneity using multiple alignment levels. This favours dealing with the alignment complexity going up in abstraction instead of trying to force alignment experts to provide coherent alignments at the lowest level of detail (increasingly hard as networks grow due to the cognitive limits of humans). Syntactically, this formalism is defined in a very general way and is independent of the ontologies underlying logic, exploiting the recursive technique to build a hierarchically structured knowledge base in levels. An instantiation of the generic formalism was evoked, with the interest put on OWL ontologies.

In the literature, one may find three basic semantic languages for the interpretation of Network of Aligned Ontologies: Non-Contextual and Centralized Semantics; Contextual and Distributed Semantics; and Contextual and Integrated Semantics. But none of those semantics can be applied directly for interpreting the MLNK formalism.

Inspired by those, this paper proposes an extended semantics for the interpretation of Network of Aligned Ontologies on several levels. The advantage of the extended semantics lies in the fact that each alignment expressed between a source and target ontology is independently treated, as each one possesses its own distinct vocabulary and semantics. The first proposed semantic, Extended Non-Contextual and Centralized Semantics (ENCACS), favours the fact that ontologies and the alignment set expressed in pairs are heterogeneous, either expressed in the same context or different compatible ones. The second proposed semantic, Distributed and Contextual-on-Several-Levels Semantics (DACOSLS), is defined in order to support ontologies and alignments heterogeneity, even if those are expressed in distinct and incompatible contexts. This semantic favours the contextualization of ontologies as well as alignments.

The DL-approach applies Extended Non-Contextual and Centralized Semantics (ENCACS), which was developed and implemented with the obtained results presented in [25].

The approach applying Distributed and Contextual-on-Several-Levels Semantics (DACOSLS) was developed and presented in a previous work [24]. In the present one, the approach concepts are recalled, then we describe the prototype used to

reason on the MLNK following the DDL-approach. Results from our test protocol are presented and compared with the DL-approach prototype results.

In order to show the difference between the different approaches, a thorough comparison is made in this paper. The resulting comparative study focuses on the adaptability of one semantic over another. This will allow readers to justify the choice of either for a given application, taken into account its context.

The organization of the rest of the article is as follows: In Section 2 the notion of networked ontologies while highlighting the specific definitions to multi-level networked knowledge is recalled. Section 3 describes the semantic approaches of MLNK interpretation. Section 4 provides detailed information on the implementation of the DDL-based MLNK reasoner prototype. In Section 5, the semantic approaches (DL-approach and DDL-approach) are compared showing how they are different from each other and in which cases one is more interesting than the other. Section 6 gives a synthesis of related works and discussion. Finally, Section 7 addresses a general conclusion.

2 NETWORK OF ALIGNED ONTOLOGIES

Generally, Network of Aligned Ontologies (NAO) formalisms were introduced with one or more motivations. Syntactically, they are composed of a family of local ontologies and alignments that bind them. They are endowed with one or more semantics for possible reasoning on aligned knowledge.

In this section, formalisms that can handle reasoning on NAOs are presented with their motivations, syntactic and semantic representations. Table 6 summarizes the latter, presenting motivations, syntax and semantics of the formalisms described in this paper.

2.1 Motivation

We start by identifying the different motivations behind existing formalisms, then we define the motivation for the introduction of Multi-Level Networked Knowledge.

2.1.1 Motivations Behind Network of Aligned Ontologies

There are four important motivations associated with NAOs:

Ontology combination: this motivation is favoured to combine several non-heterogeneous ontologies, where each one describes a separated, very different viewpoints and complementary portions of a complex domain. In general, *links* are used in order to link entities belonging to different ontologies (e.g., $\mathtt{O}_1$:`France` $\overset{\textbf{is-part-of}}{\rightarrow}$ $\mathtt{O}_2$:`Europe`) (see Section 2.2.2). As an example, E-connection [27] is a formalism proposing a syntactic representation and a formal semantics for reasoning on a Network of Aligned Ontologies, where entities in different ontologies are connected by *links*.

Resolution of semantic heterogeneity between ontologies: in order to resolve the semantic heterogeneity problem between ontologies. It is necessary to use ontology *mappings* which are semantic relations between entities (e.g., $\texttt{O}_1\texttt{:java} \overset{\perp}{\rightarrow} \texttt{O}_2\texttt{:java}$). As an example, DDL formalism [8] proposes a syntactic and semantic representation that permits reasoning on a Network of Aligned Ontologies using *mappings*.

Ontology import: The import of ontologies is mainly used to promote the reuse of the concepts, roles or individuals defined in other ontologies. The notion of importing entities belonging to other ontologies with the goal of reusing them was introduced in [7]. This is mainly interesting, as it permits reusing a number of entities from a given ontology without importing it as a whole.

Mediation of alignment: This motivation ensures an independent management of the alignments. As an example, one may cite the alignments composition for exchanging and a better reusing of the latter through the network of ontologies. The main goal is still to reuse existing alignments in order to obtain newer ones. The IDDL formalism [33] proposes a syntactic and semantic representation in order to manage and exploit alignments to ensure mediation through the knowledge network.

2.1.2 Motivation Behind MLNK

The set of pair ontology alignments have their own vocabulary. They are developed independently from each other by domain experts with different viewpoints, being then possibly heterogeneous. In order to solve the heterogeneity problem between alignments, the latter's, need to be linked in the higher levels.

A real-life application example of gas turbine ontological representation is presented. Due to their wide usage in electricity production, the gas turbine is often found in the center of large power systems that need to be managed in terms of knowledge and maintenance. Four ontologies describing gas turbine have been developed for the purpose of this example, namely:

- an ontology for equipment (`eq`), modelling the turbine technical and hierarchical knowledge. This information is provided by the constructor and contains 5 033 concepts, where each concept describes an equipment or turbine component, such as the concept `flame-detector` given by instance $\texttt{FD}_1$;
- an ontology termed (`Pr`), modelling spare parts, such as the concept `trim` given by the instance $\texttt{T}_1$;
- an ontology for modelling the position of the equipment in the turbine hierarchy (`zn`);
- an ontology created from an existing database `mt`, using a semi-automatic approach, covering, maintenance operations (both preventive and currative). The `mt` ontology exploits the first ontologies (`eq`), `Pr` and `zn`) in order to provide details on equipments and spare parts concerned by maintenance operations.

These ontologies are independent and heterogeneous; we aim to exploit them via a common interface without constraining or altering their internal representation. We propose for that effect, to insert ontology alignments separately without favouring any of the local representations. Correspondences of the *mappings* type are produced via independent tools, the case for the following correspondences: $\texttt{mt:belong} \overset{\perp}{\longleftrightarrow} \texttt{eq:belong}$ between $(\texttt{mt}, \texttt{eq})$ ontologies pair and $\texttt{pr:trim} \overset{\sqsubseteq}{\longleftrightarrow} \texttt{eq:instrumentation}$ between $(\texttt{pr}, \texttt{eq})$. The set of produced *mappings* may be enriched semi-automatically by new links (terms linking two different ontologies). This operation is performed by experts, understanding one expert for each ontologies pair. Alignments are then developed independently by domain expert expressing different viewpoints. It is then observed that the semantic heterogeneity problem occurs at the alignment level. It is the case for alignments $A_{\texttt{pr-eq}}$ and $A_{\texttt{eq-zn}}$, with the terms $A_{\texttt{pr-eq}}$:**compose** and $A_{\texttt{eq-zn}}$:**part-of**, these *links* have similar semantics. In order to reduce semantic heterogeneity between alignments and enable knowledge inference across the global network, it is necessary to insert an equivalence relation between the two *links* $A_{\texttt{pr-eq}}$:**compose** and $A_{\texttt{eq-zn}}$:**part-of**. This comes to align ontology alignments.

Example 1. An excerpt of ontologies and associated alignments are presented in Table 1.

Ontologies	Axioms
eq:	$\texttt{flame-detector}(\texttt{FD}_1)$
	$\texttt{flame-detector} \sqsubseteq \exists \texttt{belong.instrumentation}$
pr:	$\texttt{trim}(\texttt{T}_1)$
zn:	$\texttt{zone}(\texttt{ANNA1TG01})$
mt:	$\texttt{intervention}(\texttt{I}_1)$
	$\texttt{team}(\texttt{TE}_1)$
	$\texttt{intervene}(\texttt{TE}_1, \texttt{I}_1)$
	$\texttt{member} \sqsubseteq \exists \texttt{belong.team}$
Alignments	
$A_{\texttt{eq-zn}}$:	$\texttt{eq:FD}_1 \overset{\textbf{part-of}}{\longleftrightarrow} \texttt{zn:ANNA1TG01}$
$A_{\texttt{pr-eq}}$:	$\texttt{pr:trim} \overset{\sqsubseteq}{\longleftrightarrow} \texttt{eq:instrumentation}$
	$\texttt{pr:T}_1 \overset{\textbf{compose}}{\longleftrightarrow} \texttt{eq:FD}_1$
$A_{\texttt{mt-eq}}$:	$\texttt{mt:I}_1 \overset{\textbf{concern}}{\longleftrightarrow} \texttt{eq:FD}_1$
	$\texttt{eq:belong} \overset{\perp}{\longleftrightarrow} \texttt{mt:belong}$
$A_{\texttt{A}_{\texttt{pr-eq}}\texttt{-A}_{\texttt{eq-zn}}}$:	$\texttt{pr-eq:}\textbf{compose} \overset{\equiv}{\longleftrightarrow} \texttt{eq-zn:}\textbf{part-of}$

Table 1. An excerpt of ontologies and associated alignments

In order to solve the heterogeneity problem occurring between alignment's vocabularies, alignment at a higher level is proposed. This, however, necessitates the introduction of a formalism permitting a representation of MLKN. Figure 1 represents the turbine example showing alignment levels.

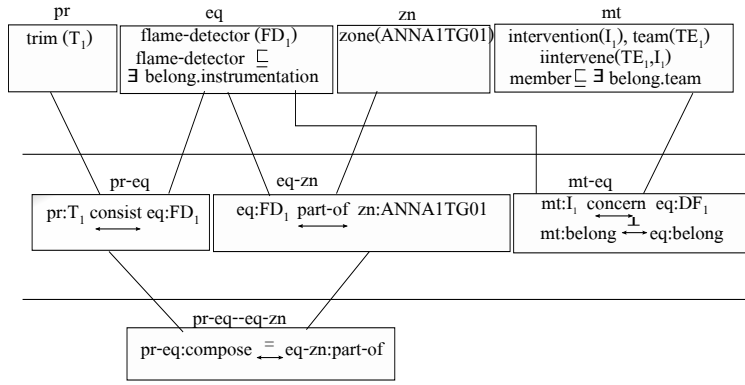


Figure 1. Knowledge representation levels

None of the existing formalisms treats alignments separately and independently with respect to ontologies and the other alignments. As a result, no proposition was made to align alignments, making all existing formalisms not able to support alignment's contextualization.

2.2 The Network of Aligned Ontologies Syntax

A network of aligned ontologies is composed of a family of local ontologies also called modular ontologies or source knowledge bases and a family of alignments. Knowledge node is a new concept defined to formalize MLNK syntax.

2.2.1 Local Ontology

The local ontologies $\{O_i\}$ of a network of aligned ontologies are indexed by a finite set of indices I. Ontologies are developed and designed in different contexts. The notion of information context has been extensively discussed in several works like [28, 16] and recently [35], with a general definition of the context being a given "point of view" or "provenance" or even "a temporal valid information". Each ontology O_i is represented in a knowledge representation language defined by:

- a syntax, that is a set of symbols and sentences (or formulas) that can be built with them;
- a notion of interpretations, which defines a domain of interpretation and associate symbols with structures over the domain;
- a satisfaction relation, which relates interpretations to the sentences they satisfy.

There are many languages for knowledge representation applied to local ontologies definition, one may cite First-Order Logic, Modal Logic, Description Logic, etc.

The proposed syntax for MLNK is generic and independent from any ontologies language (see Section 2.2.3). In order to interpret it, the choice of existing logic is given to the user, such as First-Order Logic, Modal logic, DL, etc. In the presented work we focused on DL ontologies, as DL is fundamental for semantic web and OWL ontologies. Table 6 resumes local ontologies languages for existing formalisms.

Let us recall some basics formulation and concepts of DL [5] that will be used for the remainder of the paper.

DL ontology is composed of concepts, roles and individuals, as well as axioms built out of these elements. A concept is either a primitive concept A, or, given concepts C, D, role R, individuals $a_1, \ldots, a_k$, and natural number n, $\bot$, $\top$, $C \sqcup D$, $C \sqcap D$, $\exists R.C$, $\forall R.C$, $\leq nR.C$, $\geq nR.C$, $\neg C$ or $\{a_1, \ldots, a_k\}$. A role is either a primitive role P, or, given roles R and S, $R \sqcup S$, $R \sqcap S$, $\neg R$, R^-, $R \circ S$ and R^+.

Interpretations are pairs $\langle \Delta^I, \cdot^I \rangle$, where Δ^I is a non-empty set (the domain of interpretation) and $\cdot^I$ is the function of interpretation such that for all primitive concepts A, $A^I \subseteq \Delta^I$, for all primitive roles P, $P^I \subseteq \Delta^I \times \Delta^I$, and for all individuals a, $a^I \in \Delta^I$.

Interpretations of complex concepts and roles is inductively defined by $\bot^I = \emptyset$, $\top^I = \Delta^I$, $(C \sqcup D)^I = C^I \cup D^I$, $(C \sqcap D)^I = C^I \cap D^I$, $(\exists R.C)^I = \{x | \exists y.y \in C^I \wedge \langle x, y \rangle \in R^I\}$, $(\forall R.C)^I = \{x | \forall y.\langle x, y \rangle \in R^I \Rightarrow y \in C^I\}$, $(\leq nR.C)^I = \{x | \sharp\{y \in C^I | \langle x, y \rangle \in R^I\} \leq n\}$, $(\geq nR.C)^I = \{x | \sharp\{y \in C^I | \langle x, y \rangle \in R^I\} \geq n\}$, $(\neg C)^I = \Delta^I \setminus C^I$, $\{a_1, \ldots, a_k\} = \{a_1^I, \ldots, a_k^I\}$, $(R \sqcup S)^I = R^I \cup S^I$, $(R \sqcap S)^I = R^I \cap S^I$, $(\neg R)^I = (\Delta^I \times \Delta^I) \setminus R^I$, $(R^-)^I = \{\langle x, y \rangle | \langle y, x \rangle \in R^I\}$, $(R \circ S)^I = \{\langle x, y \rangle | \exists z.\langle x, z \rangle \in R^I \wedge \langle z, y \rangle \in S^I\}$ and $(R^+)^I$ is the reflexive-transitive closure of R^I.

Axioms are either subsumption $C \sqsubseteq D$, sub-role axioms $R \sqsubseteq S$, instance assertions $C(a)$, role assertions $R(a, b)$ and individual identities $a = b$, where C and D are concepts, R and S are roles, and a and b are individuals. An interpretation I *satisfies* axiom $C \sqsubseteq D$ if and only if $C^I \subseteq D^I$; it satisfies $R \sqsubseteq S$ if and only if $R^I \subseteq S^I$; it satisfies $C(a)$ if and only if $a^I \in C^I$; it satisfies $R(a, b)$ if and only if $\langle a^I, b^I \rangle \in R^I$; and it satisfies $a = b$ if and only if $a^I = b^I$. When I satisfies an axiom α, it is denoted by $I \models \alpha$.

An ontology O is composed of a set of terms (primitive concepts/roles and individuals) called the signature of O and denoted by $\mathrm{Sig}(O)$, and a set of axioms denoted by $\mathrm{Ax}(O)$. An interpretation I is a model of an ontology O if and only if for all $\alpha \in \mathrm{Ax}(O)$, $I \models \alpha$. In this case, we write $I \models O$. The set of all models of an ontology O is denoted by $\mathrm{Mod}(O)$. A semantic consequence of an ontology O is a formula α such that for all $I \in \mathrm{Mod}(O)$, $I \models \alpha$.

An ontology is logically consistent if the ontology has a model.

2.2.2 Alignments

The correspondences represent relations between entities (terms or formulas) belonging to different ontologies. The set of correspondences is termed ontology alignment. Let us recall that there are two types of correspondences:

- The first type of alignment (*mapping*) concerns the correspondences which are associated with a predefined set of relations such as subsumption, equivalence, disjunction, etc. where the given semantic is fixed for all interpretations (e.g., $\mathtt{O_1{:}java} \overset{\perp}{\longleftrightarrow} \mathtt{O_2{:}java}$). Which means that the `java` entity in the ontology O_1 is semantically different from the `java` entity in O_2.
- The second type of alignment (*links*) is used to link ontologies covering complementary domains, it is the case of E-connection [27], $E - \mathcal{SHIQ}$ [32] and MLNK [25]. It is represented by inter-ontological roles between entities, termed simply links (e.g., $\mathtt{O_1{:}France} \overset{\mathbf{is-part-of}}{\longleftrightarrow} \mathtt{O_2{:}Europe}$).

The syntax representation of correspondences differs from one formalism to another. As an example, DDL [8] is cited here, where *mappings* (DDL does not handle *links*) are represented by directional arrows expressed as the target ontology point of view (e.g., $\mathtt{O_1{:}A} \overset{\sqsubseteq}{\rightarrow} \mathtt{O_2{:}B}$) where the inverse of the correspondence (e.g., $\mathtt{O_2{:}B} \overset{\sqsupseteq}{\rightarrow} \mathtt{O_1{:}A}$) is not valid. In the case of the proposed formalism, as well as for IDDL, double arrows are used to express correspondences with an external "point of view" of target and source ontologies (e.g., $\mathtt{O_1{:}A} \overset{\sqsubseteq}{\longleftrightarrow} \mathtt{O_2{:}B}$) where the inverse correspondence (e.g., $\mathtt{O_2{:}B} \overset{\sqsupseteq}{\longleftrightarrow} \mathtt{O_1{:}A}$) is valid and can be inserted. However, IDDL express correspondences from a global point of view with respect to the whole ontology network. This is quite difficult to achieve, considering the limited expert's knowledge not allowing a complete understanding of all domain aspects. MLNK suggests expressing correspondences according to a global point of view with respect to a pair of ontologies.

Definition 1 (Initial alignment language representation)**.** The alignment language L_A that allows expressing correspondences is initially defined as a pair $\langle E, R\rangle$ where E is a function from any ontology $O \subseteq L_A$ which defines the matchable entities of ontology O and R is a set of symbols that allow relating these entities, with $R = \{\sqsubseteq, \equiv, \perp, \in, =\}$ [14].

Alignment language, in this case, is reduced to the terms of existing vocabularies and does not have its own vocabulary.

Definition 2. A correspondence expressed in this language L_A is given by a triplet $\langle e_1, r, e_2\rangle$ noted $e_1 \overset{r}{\rightarrow} e_2$ where e_1, e_2 are entities belonging respectively to $E(O_1)$, $E(O_2)$ and $r \in R$ or r is a *link*.

These definitions do not constitute a problem if all correspondences are of *mapping* types, on the other hand, if some of them are *mappings* and others are *links*, the problem arises necessarily. This is due to the fact that the links are terms likely to have several interpretations, and can vary from one pair of ontologies to another.

The previous definitions of alignment language and correspondences do not permit alignment contextualization. To remedy to the problem, recent definitions have been given where the alignment language has its own vocabulary allowing to express distinctly *mappings* and *links*.

Definition 3 (Proposed alignment language)**.** An alignment language L_A permits the description of correspondences between two vocabularies. It is also characterized by a syntax (how correspondences are expressed) and a semantic (how correspondences are interpreted). The syntax of L_A is defined by:

- a set of terms, called links, specific to the alignment language noted $V(L_A)$;
- a function $E(L_A)$, which associates to each signature of a representation language L a set of entities that can be aligned;
- a set of relation's symbols $R(L_A)$.

Thus, the syntax of an alignment language L_A is defined by the triple $\langle V(L_A), E(L_A), R(L_A)\rangle$, denoted $\langle V, E, R\rangle$ when no ambiguity exists. Two types of correspondences might be defined as *mapping* and *link* correspondences.

Definition 4 (*Mapping* correspondence)**.** Let V_1 and V_2 be two aligned vocabularies and let the triplet $\langle V, E, R\rangle$ denote an alignment language. A *mapping* correspondence is a triple $\langle e_1, e_2, r\rangle$ noted $e_1 \overset{r}{\longleftrightarrow} e_2$ where:

- $e_1 \in E(V_1)$ and $e_2 \in E(V_2)$ are matchable entities;
- $r \in R$ denotes a relation that holds between e_1 and e_2 with $R = \{\sqsubseteq, \equiv, \perp, \in, =\}$.

Definition 5 (*Link* correspondence)**.** Let us consider V_1 and V_2 two aligned vocabularies and $\langle V, E, R\rangle$ an alignment language. A *link* correspondence is a formula in the form $e_1 \overset{l}{\longleftrightarrow} e_2$ where:

- $e_1 \in E(V_1)$ and $e_2 \in E(V_2)$ are matchable entities;
- $l \in V$ denotes a relation that holds between e_1 and e_2.

Definition 6 (Alignment)**.** Let V_1 and V_2 be two vocabularies. An *alignment of* V_1 *and* V_2 is a tuple $\Lambda = \langle V, \kappa, \lambda\rangle$ where:

- V is an alignment vocabulary;
- κ is a set of *mapping* correspondences, $e_1 \overset{r}{\longleftrightarrow} e_2$ where $e_1 \in E(V_1)$, $e_2 \in E(V_2)$ and $r \in R$;
- λ is a set of *link* correspondences, $e_1 \overset{l}{\longleftrightarrow} e_2$ where $e_1 \in E(V_1)$, $e_2 \in E(V_2)$ and $l \in V$.

2.2.3 Knowledge Node

The syntactic formalization of MLNK is defined in a very general way, independently of any language, using a recursion technique to build a knowledge base, hierarchically structured in levels. In other words, it is composed of a family of knowledge nodes and alignments between any pair of nodes where each node is self-composed of a pair of aligned sub-nodes. Hence a dynamic construction of knowledge nodes where the most elementary node is an ontology. An ontology is therefore, a level 0 knowledge

node, while each knowledge node of level $m > 0$ is constructed from a number of nodes from an inferior level, linked using alignment. Formally the node is defined as:

Definition 7 (Knowledge node)**.** A *knowledge node* is a pair $K = \langle V_K, A_K \rangle$ where V_K is a vocabulary, also written $\mathrm{Voc}(K)$, and both V_K and A_K are defined recursively:

- an ontology O is a knowledge node with vocabulary $\mathrm{Voc}(O) = \mathrm{Sig}(O)$ and A_K is the set of axioms;
- for $n \geq 1$, if $K_1, \ldots, K_n$ are knowledge nodes with vocabularies $\mathrm{Voc}(K_1), \ldots, \mathrm{Voc}(K_n)$, and for all $i, j \in [1, n]$, Λ_{ij} is an alignment of $\mathrm{Voc}(K_i)$ and $\mathrm{Voc}(K_j)$, then $K = \langle V_K, A_K \rangle$ is a knowledge node with the vocabulary:

$$V_K = \bigcup_{i,j \in [1,n]} \{ij : l \mid l \in \mathrm{Voc}(\Lambda_{ij})\} \cup \bigcup_{i \in [1,n]} \{i : e \mid e \in \mathrm{Voc}(K_i)\}$$

and $A_K = \langle (K_i)_{i \in [1,n]}, (\Lambda_{ij})_{i,j \in [1,n]} \rangle$.

If a knowledge node includes only ontologies and ontology alignments, we call it a Network of Aligned Ontologies. If a knowledge node is neither a single ontology nor a network of aligned ontologies, we call it a Multi-Level Networked Knowledge base.

2.3 The Network of Aligned Ontologies Semantics

Three basic semantics associated to Network of Aligned Ontologies are defined in [34]. Two other extended semantics inspired by basic semantics are presented in what follows.

2.3.1 Non-Contextual and Centralized Semantics (NCACS)

This semantic is formalized by classical logic, there is a unique interpretation domain for the whole network which is the union of all local interpretation domains (Δ_i for all $i \in [1, n]$). Interpretation is a model if it satisfies all the axioms of local ontologies (O_i for all $i \in [1, n]$) and alignments (A_{ij} for all $i, j \in [1, n]$). See Figure 2.

2.3.2 Contextual and Distributed Semantics (CADS)

There are two variants of CADS:

- Variant 1: This semantic is formalized by distinct and separate local interpretations ($\mathcal{I}_i$ for all $i \in [1, \ldots, n]$), but linked by domain relations (r_{ij} for all $i, j \in [1, \ldots, n]$). The distributed interpretation $\mathcal{I}$ is composed of local interpretations and domain relationships, $\mathcal{I} = \langle \{\mathcal{I}_i\}, \{r_{ij}\} \rangle$ for all $i, j \in [1, \ldots, n]$. It is a model of the network if (see Figure 3):

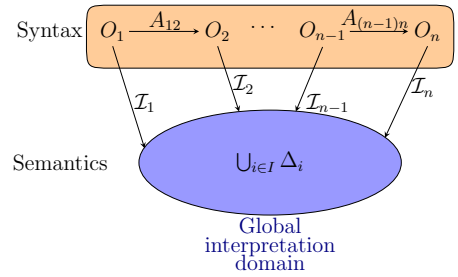


Figure 2. Non-Contextual and Centralized Semantics (NCACS)

– Each local interpretation $\mathcal{I}_i$ satisfies the axioms of the corresponding ontology (O_i for all $i \in [1, \ldots, n]$);
– The local interpretations and the domain relationships satisfy the constraints imposed by the alignments (A_{ij} for all $i, j \in [1, \ldots, n]$).

- Variant 2: This semantic is formalized by distinct and separate local interpretations ($\mathcal{I}_i$ for all $i \in [1, \ldots, n]$), and a special interpretation ($\mathcal{I}_{ij}$ for all $i, j \in [1, \ldots, n]$) assigns to each *link* R^{ij} from i to j a domain relation, that is, a subset of $\Delta_i \times \Delta_j, i, j \in [1, \ldots, n]$. The combined interpretation $\mathcal{I}$ is composed of local interpretations and a special interpretations, $\mathcal{I} = \langle\{\mathcal{I}_i\}, \{\mathcal{I}_{ij}\}\rangle$ for all $i, j \in [1, \ldots, n]$. It is a model of the network if:

  – each local interpretation $\mathcal{I}_i$ satisfies the axioms of the corresponding ontology (O_i for all $i \in [1, \ldots, n]$);
  – the local interpretations and the special interpretation satisfy the constraints imposed by the alignments (A_{ij} for all $i, j \in [1, \ldots, n]$).

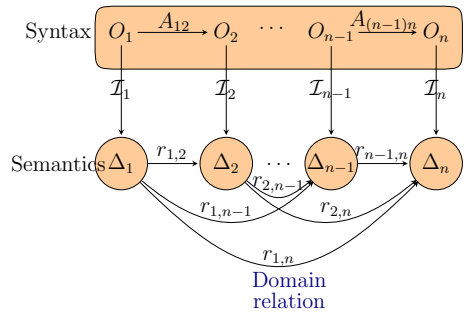


Figure 3. Contextual and Distributed Semantics (CADS)

2.3.3 Contextual and Integrated Semantics (CAIS)

CAIS can be seen as the combination of centralized semantics (on the alignment side) and distributed semantics (on the local ontologies side). The local interpretations are distinct and separate but not directly related. They are connected by means of the equalizing functions to an additional interpretation domain. The equalizing function is a projection function from local interpretation domain to a virtual global domain. The global domain is used to interpret inter-ontological knowledge (alignment) from a global point of view. It is the first idea that defines an independent interpretation of the alignments but the centralization of the alignment interpretation in a single additional domain does not allow alignment contextualization. Distributed, integrated interpretation is composed of local interpretations and equalizing functions $\mathcal{I} = \langle \{\mathcal{I}_i\}, \epsilon_i \rangle$ for all $i \in [1, \ldots, n]$ (see Figure 4). It is a model of the network if:

- local interpretations $\mathcal{I}_i$ satisfy source ontologies (O_i for all $i \in [1, \ldots, n]$);
- the pairs of source interpretation and target with the equalizing functions (ϵ_i for all $i \in [1, \ldots, n]$) satisfy the constraints imposed by the alignments (A_{ij} for all $i, j \in [1, \ldots, n]$).

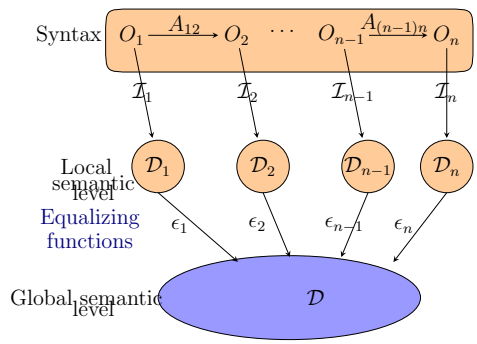


Figure 4. Contextual and Integrated Semantics (CAIS)

Inspired from the basic semantics, an extended semantics for the interpretation of Network of Aligned Ontologies on several levels is proposed. The proposed semantics have the ability to support independent alignment interpretations as well as their contextualization.

2.3.4 Extended Non-Contextual and Centralized Semantics (ENCACS)

Extended Non-Contextual and Centralized Semantics considers that the set of ontologies with corresponding alignments are interpreted in a single domain. The interpretation domain is the result of the union of the existing interpretation domains consisting of ontologies and alignments. An interpretation is a model of the

network if it satisfies all the axioms of local ontologies and alignments. These solutions are adapted for the integration of independent ontologies, independently aligned and developed in different but compatible and not contradictory contexts. Figure 5 shows an extension of the centralized semantics with the integration of alignment interpretation.

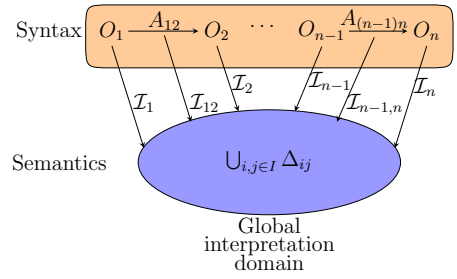


Figure 5. Extended Non-Contextual and Centralized Semantics (ENCACS)

2.3.5 Distributed and Contextual-on-Several-Levels Semantics (DACOSLS)

This semantic is an extension of CADS semantics, where alignments are interpreted from the target ontology point of view. In order to interpret alignments of the source and target ontologies independently, the idea is to generate an alignment-interpretation domain (see Figure 6). Then, local interpretations are related to alignment-interpretations through domain relationships. The notion of independent alignment-interpretations by a pair of ontologies which ensures the contextualization of the alignments. A distributed interpretation is a model if:

- the local interpretations satisfy the local ontologies;
- the alignment-interpretations satisfy the constraints posed by the alignments;
- the local interpretation, the alignment-interpretations with the domain relations satisfy the contradictions posed by the equivalence bridge rules.

3 SEMANTIC APPROACHES

Two semantic approaches are usually associated with MLNK. DL-approach is defined to interpret and reason on multi-levels networked ontologies according to ENCACS (see [25] for more details). Where the DDL-approach is defined to interpret and reason on multi-levels networked ontologies according to Distributed and Contextual-on-Several-Levels Semantics.

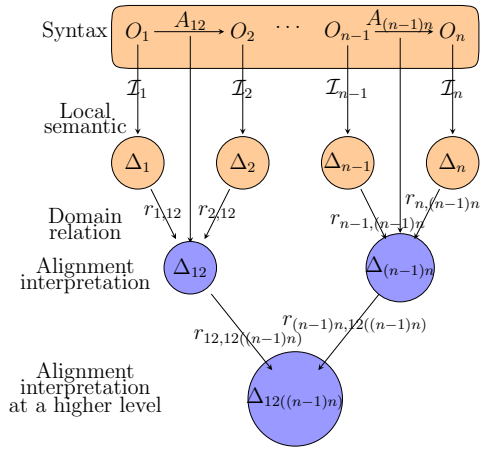


Figure 6. Distributed and Contextual-on-Several-Levels Semantics (DACOSLS)

3.1 DL-Approach

This approach consists in the transformation of the multi-levels networked ontologies into a unique description logic ontology "DL-ontology" following the steps below:

- prefix the ontologies which consist in assigning the indexes of the source ontologies to their corresponding entities;
- transformation of alignment into description logic axioms "DL-axioms";
- generation of the global ontology, also known as a multi-level knowledge node, obtained recursively by the union of the source ontologies with the integration of the axioms originating from alignments;
- testing the MLNK consistency through the DLMLNKR prototype.

3.2 DDL-Approach: Syntax and Semantics

This approach consists in the transformation "SystDis" of the multi-levels networked ontologies to a DDL system, following the steps below:

1. generation of alignment-ontology;
2. generation of equivalence bridge rules between terms of alignment-ontology and terms belonging to corresponding source ontologies.

Let us recall the necessary definitions, so that the reader can better understand implementation details of the DDLMLNKR prototype presented in Section 4.

Definition 8 (Indexing the ontology element)**.** Let i be an index. We define the function prefix on the terms, axioms and ontologies, such that $\mathsf{prefix}(X, i) = \{i{:}X\}$

when X is an atomic concept, atomic role or an individual, and if X is a formula, $\mathsf{prefix}(X, i)$ is a formula where all terms are prefixed by i.

Definition 9 (Alignment-ontology signature)**.** Let us consider a multi-level knowledge node K, alignment-ontology signature Σ_A is defined as follows according to the case:

- if K is an ontology then $\Sigma_A = \emptyset$;
- if K is a multi-level knowledge node composed of sub nodes $K_1, \ldots, K_n$ and A_{ij} which is alignment between K_i and K_j for $i, j \in [1, n]$, then:

$$\Sigma_A(K) = \bigcup_{i,j \in [1,n]} \{\mathsf{prefix}(X, i), \mathsf{prefix}(Y, j) \mid i{:}X \xleftrightarrow{r} j{:}Y \in A_{ij}\} \cup \bigcup_{i,j \in [1,n]} \mathrm{Voc}(A_{ij})$$

where X and Y are the concepts, roles or individuals and $r \in \{\sqsubseteq, \equiv, \bot, \in, =\}$, and $\mathrm{Voc}(A_{ij})$ means the alignment vocabulary, the *links* of A_{ij}.

Alignment-ontology formulas are the set of generated formulas from correspondences. Firstly, the function associating each correspondence to an axiom is defined.

Definition 10 (Correspondence transformation into axioms)**.** Let us consider an alignment A_{ij} between a node i and a node j, for $i, j \in [1, n]$. We define trans a function which assigns to each correspondence of A_{ij} a DL axiom: $\mathsf{trans}(\{i{:}A \xleftrightarrow{\sqsubseteq} j{:}B\}) = \{\mathsf{prefix}(A, i) \sqsubseteq \mathsf{prefix}(B, j)\}$; $\mathsf{trans}(\{i{:}A \xleftrightarrow{\equiv} j{:}B\}) = \{\mathsf{prefix}(A, i) \equiv \mathsf{prefix}(B, j)\}$; $\mathsf{trans}(\{i{:}A \xleftrightarrow{\bot} j{:}B\}) = \{\mathsf{prefix}(A, i) \sqsubseteq \neg\mathsf{prefix}(B, j)\}$; $\mathsf{trans}(\{i{:}u \xleftrightarrow{\in} j{:}A\}) = \{\mathsf{prefix}(A, j)(i{:}u)\}$; $\mathsf{trans}(\{i{:}u \xleftrightarrow{=} j{:}u'\}) = \{i{:}u = j{:}u'\}$; $\mathsf{trans}(\{i{:}u \xleftrightarrow{l} j{:}u'\}) = \{\mathsf{role}(l)(i{:}u, j{:}u')\}$; $\mathsf{trans}(\{i{:}A \xleftrightarrow{l} j{:}B\}) = \{\mathsf{prefix}(A, i) \sqsubseteq \exists\mathsf{role}(l).\ \mathsf{prefix}(B, j)\}$, where A, B, u and u' are the matchable entities and l is a *link*.

Definition 11 (Alignment-ontology formulas)**.** Let us consider a multi-level knowledge node K, the set of alignment-ontology formulas F_A is defined, according to the cases as follows:

- if K is an ontology then $F_A = \emptyset$;
- if K is a multi-level knowledge node composed of sub nodes $K_1, \ldots, K_n$ and alignments A_{ij} between K_i and K_j for $i, j \in [1, n]$ and trans is the function that associates to any correspondence of A_{ij} a DL-axiom (see Definition 10) then alignment-ontology-formula set $F_A(K) = \{f \mid f \in \mathsf{trans}(A_{ij})\}$.

Definition 12 (Alignment-ontology)**.** Let us consider a node $K = \langle\{K_i\}, \{A_{ij}\}\rangle$ for $i, j \in [1, n]$, K_i are local nodes and A_{ij} is an alignment between K_i and K_j. We define $\mathsf{OntoAlign}$ the alignment-ontology generated from A_{ij} of K, $\mathsf{OntoAlign}(K) = \langle\Sigma_A(K), F_A(K)\rangle$.

The bridge rules of multi-level knowledge node represent the equivalence correspondences established between the terms of alignment-ontology and terms belonging to the corresponding local ontologies.

Definition 13 (Bridge rules toward alignment-ontology)**.** Let us consider a knowledge node K. The case dependant bridge rules oriented towards the alignment-ontology (noted $B(K)$) are defined as follows:

- if K is an ontology then $B(K) = \emptyset$;
- if K is a multi-level knowledge node composed of sub nodes $K_1, \ldots, K_n$ and A_{ij} which is alignment between K_i and K_j for $i, j \in [1, n]$ then $B(K)$ contains a bridge rules defined as follows, for $i \in [1, n]$:
  - if K_i is an ontology and X is a concept or a role of K_i then $i{:}X \xrightarrow{\equiv} \mathsf{OntoAlign}(K){:}i{:}X \in B(K)$;
  - if K_i is an ontology and a is an individual of K_i then $i{:}a \xrightarrow{=} \mathsf{OntoAlign}(K){:}i{:}a \in B(K)$;
  - if K_i is a composed node and X a concept or role of $\mathsf{OntoAlign}(K_i)$ then $\mathsf{OntoAlign}(K_i){:}X \xrightarrow{\equiv} \mathsf{OntoAlign}(K){:}k_i{:}X \in B(K)$;
  - if K_i is a composed node and a an individual of $\mathsf{OntoAlign}(K_i)$ then $\mathsf{OntoAlign}(K_i){:}a \xrightarrow{=} \mathsf{OntoAlign}(K){:}k_i{:}a \in B(K)$.

The MLNK interpreted as a DDL system is composed of several local nodes connected to their alignment-ontology through a family on bridge rules.

Definition 14 (MLNK in DDL form)**.** Let us consider a knowledge node K. $\mathsf{SystDis}$ is a DDL system of K, $\mathsf{SystDis}(K) = \langle \mathsf{Onto}(K), \mathsf{Bridge}(K) \rangle$ with $\mathsf{Onto}(K)$ a family of local ontologies which is recursively defined as follows:

- $\mathsf{Onto}(K) = \{K\}$, if K is a DL-ontology;
- $\mathsf{Onto}(K) = \mathsf{Onto}(K_1) \cup \mathsf{Onto}(K_2) \cup \cdots \cup \mathsf{Onto}(K_n) \cup \mathsf{OntoAlign}(K)$ if K is a node with K_i local nodes.

$\mathsf{Bridge}(K)$ is a family of bridge rules of K recursively defined as follows:

- $\mathsf{Bridge}(K) = \emptyset$ if K is an ontology;
- $\mathsf{Bridge}(K) = \mathsf{Bridge}(K_1) \cup \cdots \cup \mathsf{Bridge}(K_n) \cup B(K)$.

We will illustrate this transformation with examples:

Example 2. Let us consider a networked ontologies $K = \langle \{O_1, O_2\}, \{A_{12}\} \rangle$, with $A_{12} = \{1{:}A \xleftarrow{\sqsubseteq} 2{:}B, 1{:}a \xleftarrow{L} 2{:}b\}$ where A, B are concepts or roles, a, b are individuals and L is a *link*. We can say that an interpretation $\mathcal{I}$ satisfies K if $\mathcal{I}$ satisfies O_1 and O_2 and it also satisfies A_{12}. To interpret K according to the DDL-approach, we transform it into a distributed system $\mathsf{SystDis}(K) = \langle \{O_1, O_2, O_3\}, \{b_1, b_2, b_3, b_4\} \rangle$ with $O1$, $O2$ being the source ontologies, O_3 is an alignment-ontology generated from the alignments and b_1, b_2, b_3, b_4 are equivalence bridge rules.

- $b_1 = 1{:}A \xrightarrow{\equiv} 3{:}1{:}A$;
- $b_2 = 1{:}a \xrightarrow{\equiv} 3{:}1{:}a$;

- $b_3 = 2{:}B \xrightarrow{\equiv} 3{:}2{:}B$;
- $b_4 = 2{:}b \xrightarrow{\equiv} 3{:}2{:}b$;

b_1, b_2, b_3, b_4 are interpreted by the domain relations that bind the corresponding local interpretations according to the DDL semantics:

- $\mathcal{I}_1, \mathcal{I}_3 \models b_1$ if $r_{13}(A^{\mathcal{I}_1}) = 1 : A^{\mathcal{I}_3}$;
- $\mathcal{I}_1, \mathcal{I}_3 \models b_2$ if $r_{13}(a^{\mathcal{I}_1}) = 1 : a^{\mathcal{I}_3}$;
- $\mathcal{I}_2, \mathcal{I}_3 \models b_3$ if $r_{23}(B^{\mathcal{I}_2}) = 1 : B^{\mathcal{I}_3}$;
- $\mathcal{I}_2, \mathcal{I}_3 \models b_4$ if $r_{23}(b^{\mathcal{I}_2}) = 1 : b^{\mathcal{I}_3}$.

The interpretation K satisfies the correspondences of K if:

- $\mathcal{I} \models 1{:}A \xleftrightarrow{\sqsubseteq} 2{:}B$ if $\mathcal{I}_1, \mathcal{I}_3 \models b_1$ and $(1 : A)^{I_3} \subseteq (2 : B)^{I_3}$ and $\mathcal{I}_2, \mathcal{I}_3 \models b_3$;
- $\mathcal{I} \models 1{:}a \xleftrightarrow{L} 2{:}b$ if $\mathcal{I}_1, \mathcal{I}_3 \models b_2$ and $L(1 : a^{I_3}, 2 : b^{I_3}) \in L^{\mathcal{I}_3}$ and $\mathcal{I}_2, \mathcal{I}_3 \models b_4$.

K distributed interpretation, $\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, r_{13}, r_{23}\}$ where $\mathcal{I}_1, \mathcal{I}_2$ are the local interpretations of O_1, O_2, $\mathcal{I}_3$ is the interpretation of generated alignment-ontology and r_{13}, r_{23} are domain relations for interpreting generated rule bridges.

$\mathcal{I}$ satisfies the ontologies network K in the DDL-approach if $\mathcal{I}$ satisfies $\mathsf{SystDis}(K) = \langle\{O_1, O_2, O_3\}, \{b_1, b_2, b_3, b_4\}\rangle$ in the basic semantics DDL.

Example 3. Ontologies and alignments of Example 1 are used to build a DDL system. Table 2 details the contents of those nodes.

4 DDLMLNKR PROTOTYPE

The DDLMLNKR prototype exploits the distributed reasoner DRAGO [30], that can handle OWL ontologies and RDF/XML files containing *mappings* and *links* as inputs.

4.1 DDLMLNKR Prototype Architecture

The main components of this tool are illustrated in Figure 7 which describes the general architecture of the DDL-approach implementation.

Each component is then described as follows:

- **Alignments loading**: It allows loading alignments saved in RDF files, resulting from alignment discovery tools available on the World Wide Web. Alignment may be enriched in a semi-automatic manner using links.
- **Parser**: It allows parsing RDF/XML files containing alignments, it also allows recognizing *mappings* which are converted into axioms and *links* converted into specific roles.

Node	Distributed System
level 0	
$K_1 = \texttt{pr}$	$B(K_1) = \emptyset$, $\mathsf{Onto}(K_1) = \{K_1\}$, $\mathsf{Bridge}(K_1) = \emptyset$, $\mathsf{SystDis}(K_1) = \{\{K_1\}, \emptyset\}$
$K_2 = \texttt{eq}$	$B(K_2) = \emptyset$, $\mathsf{Onto}(K_2) = \{K_2\}$,$\mathsf{Bridge}(K_2) = \emptyset$, $\mathsf{SystDis}(K_2) = \{\{K_2\}, \emptyset\}$
$K_3 = \texttt{zn}$	$B(K_3) = \emptyset$, $\mathsf{Onto}(K_3) = \{K_3\}$,$\mathsf{Bridge}(K_3) = \emptyset$, $\mathsf{SystDis}(K_3) = \{\{K_3\}, \emptyset\}$
level 1	
$K_4 = \{K_1, K_2, A_{\texttt{K1-K2}}\}$	$\mathsf{OntoAlign}(K_4) = \texttt{oa}_4 = \langle \Sigma_4, F_4 \rangle$, where $\Sigma_4 = \{k_1{:}\texttt{G}_1, k_2{:}\texttt{DF}_1, \textbf{compose}\}$ and $F_4 = \{\textbf{compose}(k_1{:}\texttt{G}_1, k_2{:}\texttt{DF}_1)\}$ $B(K_4) = \{k_1{:}\texttt{G}_1 \stackrel{=}{\rightarrow} \texttt{oa}_4{:}k_1{:}\texttt{G}_1,$ $k_2{:}\texttt{DF}_1 \stackrel{=}{\rightarrow} \texttt{oa}_4{:}k_2{:}\texttt{DF}_1\}$; $\mathsf{Onto}(K_4) = \{K_1, K_2, \texttt{oa}_4\}$ $\mathsf{Bridge}(K_4) = B(K_4)$; $\mathsf{SystDis}(K_4) = \langle \mathsf{Onto}(K_4), \mathsf{Bridge}(K_4) \rangle$
$K_5 = \{K_2, K_3, A_{\texttt{K2-K3}}\}$	$\mathsf{OntoAlign}(K_5) = \texttt{oa}_5 = \langle \Sigma_5, F_5 \rangle$, where $\Sigma_5 = \{k_2{:}\texttt{DF}_1, k_3{:}\texttt{ANNA1TG01}, \textbf{part-of}\}$ and $F_5 = \{\textbf{part-of}(k_2{:}\texttt{DF}_1, k_3{:}\texttt{ANNA1TG01})\}$ $B(K_5) = \{k_2{:}\texttt{DF}_1 \stackrel{=}{\rightarrow} \texttt{oa}_5{:}k_2{:}\texttt{DF}_1,$ $k_3{:}\texttt{ANNA1TG01} \stackrel{=}{\rightarrow} \texttt{oa}_5{:}k_3{:}\texttt{ANNA1TG01}\}$; $\mathsf{Onto}(K_5) = \{K_2, K_3, \texttt{oa}_5\}$ $\mathsf{Bridge}(K_5) = B(K_5)$ $\mathsf{SystDis}(K_5) = \langle \mathsf{Onto}(K_5), \mathsf{Bridge}(K_5) \rangle$
level 2	
$K_6 = \{K_4, K_5, A_{\texttt{K4-K5}}\}$	$\mathsf{OntoAlign}(K_6) = \texttt{oa}_6 = \langle \Sigma_6, F_6 \rangle$ where $\Sigma_6 = \{\texttt{oa}_4{:}\textbf{compose}, \texttt{oa}_5{:}\textbf{part-of}\}$ and $F_6 = \{\texttt{oa}_4{:}\textbf{compose} \equiv \texttt{oa}_5{:}\textbf{part-of}\}$ $B(K_6) = \{\texttt{oa}_4{:}\textbf{compose} \stackrel{=}{\rightarrow} \texttt{oa}_6{:}\textbf{compose},$ $\texttt{oa}_5{:}\textbf{part-of} \stackrel{=}{\rightarrow} \texttt{oa}_6{:}\textbf{part-of}$ $\mathsf{Onto}(K_6) = \mathsf{Onto}(K_4) \cup \mathsf{Onto}(K_5) \cup \{\texttt{oa}_6\}$ $= \{K_1, K_2, K_3, \texttt{oa}_4, \texttt{oa}_5, \texttt{oa}_6\}$ $\mathsf{Bridge}(K_6) = \mathsf{Bridge}(K_4) \cup \mathsf{Bridge}(K_5) \cup B(K_6)$ $= B(K_4) \cup B(K_5) \cup B(K_6)$ $\mathsf{SystDis}(K_6) = \langle \mathsf{Onto}(K_6), \mathsf{Bridge}(K_6) \rangle$

Table 2. Example of an MLNK in DDL form. We rename $\mathsf{OntoAlign}(K_i)$ in $\texttt{oa}_i$ for $i \in [4, 6]$.

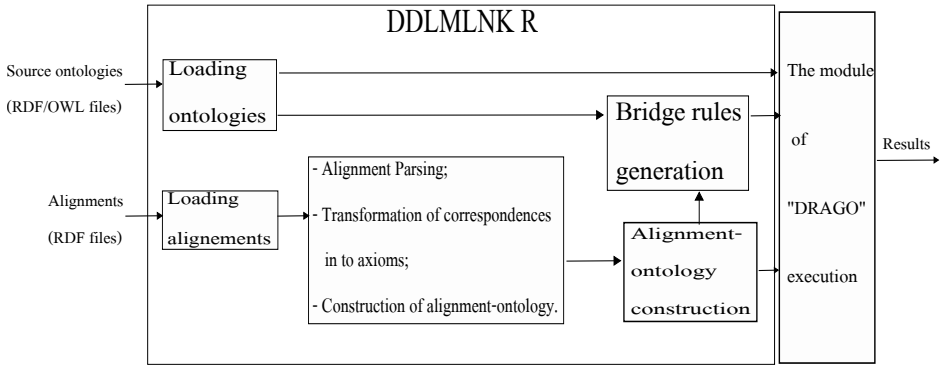


Figure 7. DDLMLNKR architecture

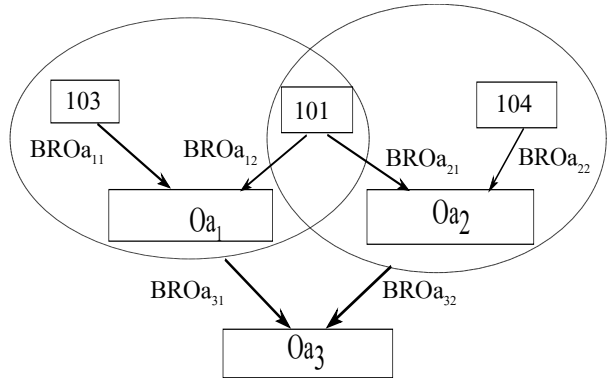


Figure 8. MLNK transformation into a distributed system

- **Alignment-ontology generating**: In this module, the construction of an ontology in DL whose entities appear to the left and right of the alignment-correspondences is performed. This module also integrates the axioms produced from the transformation of *mappings* and roles from *links*.
- **Bridges rules generating**: This component is used to generate the bridge rules between the entities belonging to the local ontologies and the corresponding entities belonging to the alignment-ontologies. They are then stored as C-OWL [9] files. C-OWL (Contextualized OWL) is an extension of OWL language designed to express *mappings* in DDL [8] formalism.
- **Executing module of distributed reasoner DRAGO**: URLs of the target ontology (alignment-ontology) and bridges rules are introduced and the source ontologies are determined by DRAGO. Subsequently, it will then be possible to determine the consistency of the networked ontologies.

Algorithm 1 Transformation of MLNK into a distributed system

load($\{A_{ij}\}$) // $i, j \in [1, \ldots, n]$
for all $A_{ij} \in \{A_{ij}\}$ **do**
  create($O_{ak}, BRO_{ak1}, BRO_{ak2}$) // $k \in [1, \ldots, n]$
  for all correspondence $c \in A_{ij}$ **do**
    read i:entity1, j:entity2
    O_{ak}.add(O_{ak}:i:entity1)
    O_{ak}.add(O_{ak}:j:entity2)
    if $c = map$ **then**
      transform c into axiom
      O_{ak}.add(O_{ak}:axiom)
    else
      transform c into Object-property // ($c = link$)
      O_{ak}.add(O_{ak}:ObjectProperty)
    end if
    create equiv-map between i:entity1 and O_{ak}:i:entity1
    BRO_{ak1}.add(equiv-map)
    create equiv-map between j:entity2 and O_{ak}:j:entity2
    BRO_{ak2}.add(equiv-map)
  end for
end for

4.2 Implementation and Experimentation of DDLMLNKR Prototype

Experimentation tests were performed on Benchmark ontologies[1]. Table 3 describes the size of the used ontologies and alignments constituting the MLNK. Inter-Ontology alignments $A_{101-103}, A_{101-104}$ were enriched by new *links* as they did not contain any vocabulary. Then an Alignment, $A_{101-103-101-104}$ is created between inter-ontology alignments $A_{101-103}, A_{101-104}$, enriched by *mappings* between the *links* existing in the alignments $A_{101-103}, A_{101-104}$. Having the "Alignment API" format [13] extended earlier, in order to store *links*. A part of $A_{101-103-101-104}$ alignments is shown in Listing 1. A *mapping* representing an equivalence relation is inserted between the *links* "evaluate" and "reviewed".

Considering we have a MLNK, with existing alignments at several levels, $K = \langle\{101, 103, 104\}, \{A_{101-103}, A_{101-104}, A_{101-103-101-104}\}\rangle$. The transformation of the network to a distributed system SystDis(K) consists in generating (see Algorithm 1):

- Ontologies O_{a1}, O_{a2}, O_{a3} for the respective alignments $A_{101-103}, A_{101-104}$, $A_{101-103-101-104}$.
- Equivalence Bridge Rules between generated ontology alignments and source ontologies: $BRO_{a11}, BRO_{a12}, BRO_{a21}, BRO_{a22}, BRO_{a31}, BRO_{a32}$.

[1] `http://oaei.ontologymatching.org/2014/`

$\mathsf{SystDis}(K) = \langle\{101, 103, 104, O_{a1}, O_{a2}, O_{a3}, BRO_{a11}, BRO_{a12}, BRO_{a21}, BRO_{a22}, BRO_{a31}, BRO_{a32}\}\rangle$ is the distributed obtained system. The transformation is depicted in Figure 8). The steps implemented during the transformation of the network, K, following the DDLMLNKR prototype are:

1. Load alignments $A_{101-103}, A_{101-104}, A_{101-103-101-104}$.
2. The prototype parses the alignments, identifies correspondences of *mapping* types and transforms them into axioms. The correspondences of *link* types are transformed into roles.
3. The prototype generates alignment-ontologies O_{a1}, O_{a2}, O_{a3}, having as a signature, entities being on the left and right of correspondences and roles resulting from *links* transformation. O_{a1}, O_{a2}, O_{a3} contains also axioms resulting from correspondences transformation.
4. The prototype generates bridge rules BRO_{a11}, BRO_{a12}, BRO_{a21}, BRO_{a22}, BRO_{a31}, BRO_{a32}. This step consists of creating the correspondences of *mapping* type between entities in alignment-ontologies and their images in source ontologies.
5. The execution of the DRAGO reasoner for consistency test is handled as follow:
   (a) Construction of the first $Peer1$ inserting the target ontology O_{a1} and Bridge rules BRO_{a11}, BRO_{a12}. For each bridge rule, the source ontology is identified and automatically inserted. As an example, for BRO_{a11} ontology 101 is inserted, as for BRO_{a12} it is ontology 103.
   (b) The second $Peer2$ is constructed by inserting the target ontology O_{a2} and bridge rules BRO_{a21}, BRO_{a22}. Source ontologies 101 and 104 are identified and inserted automatically.
   (c) The third $Peer3$ is constructed by inserting the target ontology O_{a3} and bridge rules BRO_{a31}, BRO_{a32}, Source ontologies O_{a1} and O_{a2} are identified and inserted automatically.
   (d) Run the consistency test for each $Peer$.

A $Peer$ is a concept of the DRAGO reasoner [30] consisting in regrouping for each target ontology, its own mappings as well as associated ontologies.

The distributed system, $\mathsf{SystDis}(K)$, is consistent if and only if: the Peer1, Peer2 and Peer3 are consistent. Results with respect to the transformation time and consistency time for the Network K, are presented in Section 5.4 for comparative analysis with DLMLNKR results presented in the paper [25].

5 DL-APPROACH AND DDL-APPROACH COMPARISON

In this section, DL and DDL-approaches are compared, with respect to specific criteria in order to determine for which cases one is more suitable than the other.

The two approaches are then studied with respect to both evaluation criteria and comparative summary tables are presented in Tables 4 and 5.

Ontologies/Alignments	Size
101	71.5 kB
103	80.4 kB
104	46.1 kB
$A_{101-103}$	44.4 kB
$A_{101-104}$	49.3 kB
$A_{101-103-101-104}$	4.45 kB

Table 3. Ontologies/alignments size

```
<Alignment>
<alignment IRI = "http://.../alignment-101-103-101-104.rdf"/>
 <xml>yes</xml>
 <level>0</level>
  <type>11</type>
  <onto1>http://.../alignment-101-103.rdf</onto1>
  <onto2>http://.../alignment-101-104.rdf</onto2>
  <map>
  <Cell>
  <entity1 rdf:resource=
           'http://.../alignment-101-103#evaluate'/>
  <entity2 rdf:resource=
           'http://.../alignment-101-104#reviewed'/>
<measure rdf:datatype='http://...#float'>1.0</measure>
      <relation>=</relation>
</Cell>
</map>
```

Listing 1. A part of alignment ($A_{101-103-101-104}$)

5.1 Consistency Comparison

For consistency, the goal is to try to prove that an inconsistent multi-level networked knowledge expressed in the DL-approach, could be consistent in the DDL-approach.

Theorem 1. If a Multi-Level Networked knowledge is inconsistent when expressed in DL-approach, it can be consistent when expressed in DDL-approach.

This theorem can be proved by showing that the multi-level networked knowledge, in the example is inconsistent according to DL-approach semantics (ENCACS) and is consistent according to DDL-approach semantics (DACOSLS).

Example 4. Let us consider the ontologies $O_1 = \{A_1 \sqsubseteq \neg B_1, A_1(a)\}$, $O_2 = \{A_2 \sqsubseteq B_2\}$ and the alignment $A_{12} = \{1{:}A_1 \overset{\equiv}{\longleftrightarrow} 2{:}A_2, 1{:}B_1 \overset{\equiv}{\longleftrightarrow} 2{:}B_2\}$.

Lemma 1. DL-approach consistency: Constitute a global ontology whose elements are prefixed from source ontologies and the *mappings*, *links* are transformed into

axioms $O_G = \{1{:}A_1 \sqsubseteq \neg 1{:}B_1, 1{:}A_1(1{:}a), 2{:}A_2 \sqsubseteq 2{:}B_2, 1{:}A_1 \equiv 2{:}A_2, 1{:}B_1 \equiv 2{:}B_2\}$.

$$1{:}A_1 \sqsubseteq \neg 1{:}B_1 \tag{1}$$

$$1{:}A_1(1{:}a) \tag{2}$$

$$2{:}A_2 \sqsubseteq 2{:}B_2 \tag{3}$$

$$1{:}A_1 \equiv 2{:}A_2 \tag{4}$$

$$1{:}B_1 \equiv 2{:}B_2 \tag{5}$$

$$1, 4, 5 \Rightarrow 2{:}A_2 \sqsubseteq \neg 2{:}B_2 \tag{6}$$

$$2, 4 \Rightarrow 2{:}A_2(1{:}a) \tag{7}$$

$$6, 7 \Rightarrow \neg 2{:}B_2(1{:}a) \tag{8}$$

$$7, 3 \Rightarrow 2{:}B_2(1{:}a) \tag{9}$$

Contradiction according to (8) and (9) and this implies that O_G is DL-approach inconsistent.

Lemma 2. DDL-approach consistency: Let us take the same Example 4, construct a distributed system S according to the DDL-approach, with an alignment-ontology constructed from the correspondences, noted O_{12}, generating then the corresponding bridges rules $\mathcal{B}$.

We obtain an ontology $O_{12} = \{1{:}A_1 \equiv 2{:}A_2, 1{:}B_1 \equiv 2{:}B_2\}$ and the bridges rules $\mathcal{B} = \{b_1, b_2, b_3, b_4\}$ where: $b_1 = \{1{:}A_1 \stackrel{\equiv}{\rightarrow} 12{:}(1{:}A_1)\}$; $b_2 = \{1{:}B_1 \stackrel{\equiv}{\rightarrow} 12{:}(1{:}B_1)\}$; $b_3 = \{2{:}A_2 \stackrel{\equiv}{\rightarrow} 12{:}(2{:}A_2)\}$; $b_4 = \{2{:}B_2 \stackrel{\equiv}{\rightarrow} 12{:}(2{:}B_2)\}$.

To show that $S = \{O_1, O_2, O_{12}, \mathcal{B}\}$ is consistent, then we must find a model that satisfies all axioms and bridges rules of S.

Supposing that a model of S exists then there is a distributed interpretation $I = \{I_1, I_2, I_3, r_{12}, r_{13}, r_{23}, r_{21}, r_{31}, r_{32}\}$ such that $I \models S$.

This implies that:$I_1 \models O_1$; $I_2 \models O_2$; $I_3 \models O_{12}$; $I_1, I_3, r_{13} \models b_1$; $I_1, I_3, r_{13} \models b_2$; $I_2, I_3, r_{23} \models b_3$; $I_2, I_3, r_{23} \models b_4$.

This is equivalent to showing that there exists an interpretation $I = \{I_1, I_2, I_3, r_{13}, r_{23}\}$ such that: $A_1^{I_1} \subseteq \neg B_1^{I_1} \subseteq \Delta_1$; $a^{I_1} \in A^{I_1}$; $A_2^{I_2} \subseteq B_2^{I_2} \subseteq \Delta_2$; $r_{13}(A_1^{I_1}) = (1{:}A_1)^{I_3}$; $r_{13}(B_1^{I_1}) = (1{:}B_1)^{I_3}$; $r_{23}(A_2^{I_2}) = (2{:}A_2)^{I_3}$; $r_{23}(B_2^{I_2}) = (2{:}B_2)^{I_3}$.

Consider the following domain of interpretation: $\Delta_1 = \{1\}$, $\Delta_2 = \{2\}$, $\Delta_3 = \{3\}$, and interpretation functions defined as follows: $A_1^{I_1} = \{1\}$; $a^{I_1} = 1$; $B_1^{I_1} = \emptyset$; $A_2^{I_2} = \emptyset$; $B_2^{I_2} = \emptyset$; $(1{:}A_1)^{I_3} = \emptyset$; $(1{:}B_1)^{I_1} = \emptyset$; $(2{:}A_2)^{I_3} = \emptyset$; $(2{:}B_2^{I_3}) = \emptyset$; $r_{13}(A_1^{I_1}) = \emptyset$; $r_{13}(B_1^{I_1}) = \emptyset$; $r_{23}(A_2^{I_2}) = \emptyset$; $r_{23}(B_2^{I_2}) = \emptyset$.

So for $I = \langle(\{1\}, I_1), (\{2\}, I_2), (\{3\}, I_3), \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset\rangle$, we have $I \models S$, then S is consistent.

It can then be concluded that the way the alignments are treated, when expressed in DL, is fixed and thus allows a reconciliation of local ontologies and alignments. This approach can be used in the case of modular ontologies, alignments, where each module is part of a global perspective in a broader domain. However, It has limits when it comes to the Word Wide Web, where ontologies, alignments can have contradictory points of view. On the other hand, when expressed in DDL, ontologies with different viewpoints may collaborate, even if they are considered incompatible.

5.2 Transformation Complexity

- The complexity of transforming the multi-level networked knowledge into a DL-ontology is linear in terms of ontologies and corresponding alignments (*comptDL*). It can be calculated using the number of prefix (ontology and links terms), noted ($nbprefix$) and the generated axiom number noted (*nbaxiom*). Let the variables n_i, l, m and p, be respectively the number of local terms belonging to the local ontology O_i, the number of links, the number of levels and the number of correspondences;

$$nbprefix = (m-1) * l + m * \sum n_i, \tag{10}$$

$$nbaxiom = p, \tag{11}$$

$$comptDL = nbprefix + p. \tag{12}$$

- The transformation complexity in a DDL distributed system (comptDDL) is calculated according to the number of operations performed to create axioms in the alignment ontology (Axioms are obtained from the transformation of correspondences), and the number of bridge rules creation operations (nbbr). Let us recall that for a correspondence there are two terms (the terms on the right and the terms on the left of the correspondence) and for each term, a bridge rule is created;

$$nbaxioma = p, \tag{13}$$

$$nbbr = 2p, \tag{14}$$

$$comptDDL = nbaxioma + nbbr. \tag{15}$$

The transformation complexity in the case of updating local ontologies expressed in DL is proportional to the number of updates, bearing in mind that updating local ontologies leads to the reconstruction of a global ontology. For the DDL-approach, the update of the local ontologies does not affect the transformation. Thus, it can be concluded that DDL-approach is more appropriate in the case where the evolution of local ontologies is more important than that of the correspondences.

5.3 Reasoning Complexity

Reasoning complexity of MLNK semantic-approaches is based on the reasoning complexities of the basic semantics of DL and DDL. Multi-Level Networked Knowledge in DL-approach is transformed into DL-ontology constructed from a fusion of local ontologies whose terms have been prefixed and alignments transformed into axioms. The local ontologies can be formalized in different logics, with the expressivity of the axiom' origin alignment being very simple and possibly formalized in the decidable $\mathcal{EL}$ language whose complexity is *NPcomplete*. Thus, the decidability and the complexity of the MLNK interpreted in DL can be given by studying the decidability and the complexity fusion of the local description logics and the integrated axioms logics. In that context, a recent work addressing the reasoning complexity in multi-viewpoint ontologies, via import from other ontologies may be of interest [23].

This aspect has not been dealt with in this paper, however, the reader is redirected to [6] for a more comprehensive description. First, this work shows that the fusion of two description logic is a fragment of the union of the latter because reasoning on the union of the two logics requires the implementation of a new reasoning method. However, reasoning on the merger can be reduced to reasoning on logical components. Moreover, reasoning on the union of two decidable logics can be undecidable, whereas reasoning on the fusion of the same logic remains decidable.

For example, the union of logics $\mathcal{ALCF}$ (which is an extension of $\mathcal{ALC}$ by the addition of functional roles) and $\mathcal{ALC}^{+,\circ,\cup}$ (which is an extension of $\mathcal{ALC}$ by the addition of transitivity, composition and union of roles), is undecidable. While their fusion is decidable. According to the same paper, the complexity of the description logics merge, whose complexity is **Pspace** is also **Pspace** [6]. This is not valid for the union of these logics. For example, the complexity of the union of logics $\mathcal{ALCFOQ}$ (which is an extension of $\mathcal{ALC}$ by adding functional role, nominal and number restriction) and the $\mathcal{ALCI}$ logic (which is an extension of $\mathcal{ALC}$ by the addition of inverse role) is **NExpTime** whereas the complexity of the component logic is **PSpace** [6]. This is different for the DDL-approach, where the logics are not merged but connected by relationships, Ghidini and al. in [18] present a study showing that the inference on *mappings* is decidable and the complexity ranges between **ExpTime** and **2ExpTime**. It can then be concluded, that the complexity of the MLNK interpreted in DDL can be equal to the highest complexity among local ontologies and mappings inferences.

5.4 Comparison of MLNK Prototypes

The results given by the MLNK transformation test performed by the two prototypes DLMLNKR and DDLMLNKR on the initial ontologies (Case 1) show that the transformation time in a distributed system is slightly improved over the one obtained constructing a global DL ontology, see Table 4 and Figure 9 (Case 1).

Case 2 evaluates the impact of the source ontology evolution on transformation time. Ontologies have been enriched by new entities, independent from alignments.

This permits to enlarge the source ontology sizes, keeping the alignment size unchanged. Then results presented in Table 4 and Figure 9 (Case 2) show that the transformation time of the MLNK using the DDLMLNKR prototype remains unchanged. This concludes that the DDL-approach is transparent with respect to ontology evolution.

In Case 3, the impact of alignment evolution is tested, with the insertion of *mappings* and *links* performed between existing entities. The goal is to increase alignments size while keeping ontology size unchanged. The results show that transformation results using both prototypes are affected. This concludes that MLNK transformation time evolves with respect to the evolution of alignments size. Table 4 and Figure 9 (case 3) show that reasoning upon distributed semantic is context depending, and more computationally expensive than reasoning based on a non contextual one. However, according to Section 5.1, it has been proven that the consistency test for contextual semantics is more efficient than that of not-contextual semantics. Let us suppose that for a given case, the consistency test following a DL-approach is inconsistent and that entities causing the inconsistency belong to different ontologies, however, not concerned by alignments. In that case, the network is consistent following the DDL-approach.

Based on the consistency test for all three studied cases, it is clear that the evolution of ontology and alignment sizes does not affect consistency at all. In other words, evolution does not affect complexity (Table 4 and Figure 10).

	Ontologies/ Alignments	Size (kB)	DLMLNKR Time (ms)	DDLMLNKR Time (ms)
Case 1	101	71.5		
	103	80.4		
	104	46.1	Transformation = 1 140	Transformation = 1 087
	$A_{101-103}$	44.4	Consistency = 460	Consistency = 8 200
	$A_{101-104}$	49.3		
	$A_{101-103-101-104}$	4.45		
Case 2	101	104.2		
	103	110.3		
	104	78.8	Transformation = 1 161	Transformation = 1 087
	$A_{101-103}$	44.4	Consistency = 464	Consistency = 8 222
	$A_{101-104}$	49.3		
	$A_{101-103-101-104}$	4.45		
Case 3	101	71.5		
	103	80.4		
	104	46.1	Transformation = 1 232	Transformation = 1 189
	$A_{101-103}$	78.2	Consistency = 477	Consistency = 8 302
	$A_{101-104}$	84.5		
	$A_{101-103-101-104}$	9.01		

Table 4. Comparison of MLNK prototypes results

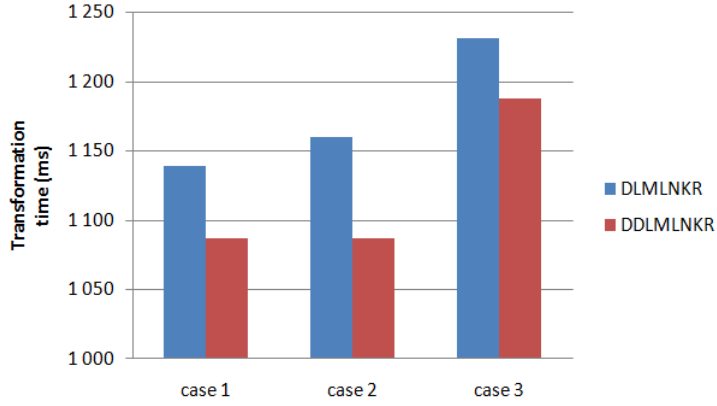


Figure 9. MLNK transformation test results

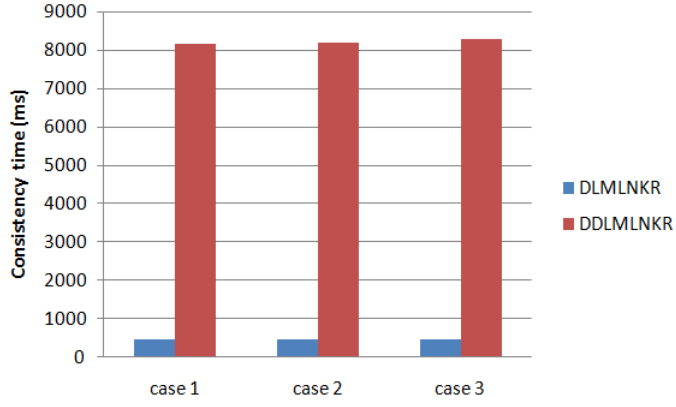


Figure 10. MLNK consistency test results

6 STATE OF ART SYNTHESIS AND DISCUSSION

In their previous works, the authors have surveyed research in relation to the topic of MLNK formalisms [25, 24], and do not wish to develop them further again in the present work, stating only the most recent ones. Previous research have been classified into two main research categories: "aligned knowledge networks" and "contextual knowledge modelling". In the first category "aligned knowledge networks", research focuses on representation and reasoning on heterogeneous ontologies built independently however still aligned. This is the case in Distributed Description Logic [8], Integrated Distributed Description Logics [33], Package-based Description

Approach	Point of View of Ontologies and Alignments	Impact of Updating Ontologies	Reasoner Implementation	Transformation Complexity	Reasoning Complexity
DL-approach	compatible viewpoints	leads to the network updating	supports multiple logics	increases with increasing sizes of ontologies and alignments	= complexity of the local logics merge
DDL-approach	inconsistent viewpoints	transparent to the network	a reasoner by logic	increases with increasing sizes of alignments	= the highest among the complexities of local logics or mappings inference

Table 5. Comparative table of DL and DDL-approaches

Logics [7], E-connection [27] and $E - \mathcal{SHIQ}$ [32], as well as the proposed formalism. Works classified in the second category "contextual knowledge modelling" model the contexts, linking those via a meta description. Each context posseses then its own instances and uses aggregation relations in order to link instances. As examples, [26, 22, 21], and recently [4] as well as [19], fall into this category, with the latter reference proposing reasoning on a hierarchical structure of the contexts.

The difference between the categories vision "contextual knowledge modelling" and "aligned knowledge networks" is similar to the difference between the Global-As-View (GAV) and Local-As-View (LAV) approaches used in integration data systems formalized and expressed in terms of requests [10, 15].

The modelling principle of works in "contextual knowledge modelling" category is the same as that of GAV where a top-down design approach is applied, proceeding from global to local. On the other hand, for the works in "aligned knowledge networks" category and LAV approaches, the upward design method is applied from local to global.

Other works consider that every local source in a network is treated as an independent module, permitting reasoning on the latter [20, 29].

In this paper, stress is put on formalisms that represent and reason on independent and aligned ontologies. Differences between presented formalisms will be discussed, with a special attention given to the contribution of the proposed formalism. A summary of the above is depicted in Table 6.

6.1 Multi-Level Networked Knowledge Representation

Multi-level networked knowledge is composed of a set of aligned nodes, these in turn are composed of the aligned sub-nodes and so on, where the most elementary nodes are ontologies. The alignment of the nodes composed of sub-nodes and alignments between them makes it possible to align the alignments and thanks to this structure the alignments can be formalized. No formalism cited below tolerates a dynamic representation of local and aligned knowledge. In addition, the syntactic formalization of local knowledge (ontology and nodes) in the proposed formalism is described in an abstract and independent way from any language and, consequently, can be adapted to any logic. DDL [8], P-DL [7], E-SHIQ and IDDL [33] are developed for a network of description logic ontologies. The ontologies in DFOL [17] formalism can be expressed in first-order logic. In E-connection [27], the local ontologies of the same network can be represented in various logics along with an abstract description system.

6.2 Alignment Contextual Representation

In multi-level networked knowledge, alignments are expressed using an alignment language independently from ontology languages. These have their own vocabularies, consisting of *mappings* and/or *links* and expressed according to the point of view of the pair of ontologies combination. In other words, according to the global

point of view in relation to a pair of ontologies. Unlike DDL and IDDL that only define and interpret *mappings*, E-connection [27] and $E - \mathcal{SHIQ}$ [32] express *links* but do not take into account the conflict of alignment heterogeneity. This is mainly because they are oriented and interpreted according to the target ontology correspondence point of view. The definition of the correspondences for a global point of view has already been presented in the IDDL formalism, but given the absence of *links* (therefore of alignment vocabulary), it does not require alignment of higher levels.

6.3 The Semantics Associated with Multi-Level Networked Knowledge Formalism

For interpretation, an instantiation of the generic formalism is carried out. We are interested in the case where ontologies are expressed in description logics (DL).

- The DL-approach that adopts ENCACS, the basic Non-Contextual and Centralized Semantics is applied by SomeWhere [1] and SomeRDFS [3], SomeOWL [2] and OWL's import semantics;
- The DDL-approach adopts Distributed and Contextual-on-several-levels Semantics, the basic distributed and contextual semantics is applied using DDL, PDL, E-connection and $E - \mathcal{SHIQ}$. In our case, the alignments are not interpreted according to the target ontology correspondence viewpoint, but they are interpreted in an external level. Independently of local ontologies, this external level is represented by an interpretation domain associated to generated alignment-ontology.

6.4 Reasoning

Several reasoning prototypes may be associated with MLNK. DLMLNKR prototype [25] allows reasoning on the proposed formalism adopting the DL-approach. The SomeWhere and SomeRDF algorithms can also be exploited (but only when links are ignored) to ensure a distributed and not-contextual reasoning. The DDL-approach implementation (DDLMLNKR prototype, Section 4.2) is ensured using the DRAGO reasoner and allows a distributed and contextual reasoning on the MLNK.

7 CONCLUSIONS

This work is the extension of previous works [25, 24], and proposes an extended semantics that can be associated with MLNK. The main advantage of those semantics is their ability to handle separately alignment interpretations. The DACOSLS is not only suitable for contextual ontology reasoning, but also for contextual alignment reasoning. In order to prove the feasibility and efficiency of the DDL-approach which adopts DACOSLS, a prototype based on the DRAGO reasoner and termed

Formalism: DDL Motivation: resolution of semantic heterogeneity between ontologies Local sources: DL ontologies Alignments: *mappings*, view point of the target ontology Semantics: CADS variant 1 Reasoning: distributed in peer-to-peer system Drago distributed reasoner [30]
Formalism: E-connection Motivation: ontologies combination Local sources: Logic ontologies with Abstract Description System Alignments: *links*, view point of the target ontology Semantics: CADS variant 2 Reasoning: distributed Extended Pellet reasoner [31]
Formalism: P-DL Motivation: ontologies import Local sources: DL ontologies Alignments: *foreign term*, view point of the target ontology (e.g., $\mathtt{O_i} \xrightarrow{\mathtt{t}} \mathtt{O_j}$) ontology O_j imports term t defined in ontology O_i Semantics: CADS variant 1 Reasoning: distributed P-DL distributed reasoner: `https://sourceforge.net/projects/p-dl-reasoner/`
Formalism: IDDL Motivation: resolution of semantic heterogeneity between ontologies, mediation of alignments Local sources: DL ontologies Alignments: *mappings*, global view point Semantics: CAIS Reasoning: distributed Draon distributed reasoner [12]
Formalism: $E-\mathcal{SHIQ}$ Motivation: resolution of semantic heterogeneity between ontologies, ontologies combination Local sources: DL ontologies Alignments: *mappings*, *links*, view point of the target ontology Semantics: CADS combination of variant 1 and variant 2 Reasoning: distributed $E-\mathcal{SHIQ}$ distributed reasoner [32]
Formalism: MLNK Motivation: resolution of semantic heterogeneity between ontologies and alignments, ontologies combination Local sources: nodes hierarchically composed of aligned sub-nodes, independent of any language Alignments: *mappings*, *links*, ontologies-pair view point Semantics: DL-approach: ENCACS DDL-approach: DACOSLS Reasoning: centralized for DL-approach distributed for DDL-approach Reasoner: DLMLNKR [25], DDLMLNKR

Table 6. Summary table of state of the art

DDLMLNKR was designed and implemented. Results on consistency tests and transformation time are assessed and commented, as well as compared to the ones obtained using the DL-approach. Based on the viewpoint notion, it can be concluded that DL-approach may be used in cases where interpretation domains of the network local sources are defined in different but compatible contexts. Each domain consists then in a portion of completing others in the larger domain. DDL-approach is therefore recommended in the case where local sources interpretation domains (Ontologies and Alignments) of the network are defined in different incompatible contexts, thus permitting contextualization of ontologies and alignments. Other comparison criteria may be useful to help users choose the most appropriate approach for their applications.

However, the introduction of such structures poses new practical and theoretical issues, which we would like to explore later, may be given by:

1. One can wonder about the problem of automatic correspondences discovery between alignments: are the tools and techniques used for ontology alignment construction adapted to all levels of a knowledge network? Can alignments be used at a certain level for the discovery of higher level alignments?
2. The need for a concise representation of such networks in a possible standardized format.
3. Knowledge management or visualization tools need to be built to organize and observe multi-level networks in order to maintain them throughout their life cycle. In addition, the hierarchical construction of multi-level networks requires re-evaluating knowledge modelling methodologies by detailing the steps to be followed for their development.
4. Concerning the semantic part, the use of existing paradigms was privileged. However, it would be interesting to reflect on another way of interpreting the MLNK semantics by defining a formal semantics constructed directly on this structure and then propose a correct and complete reasoning algorithm.
5. Finally, it would be important and useful to develop a system able of interrogating this type of network. A formalization of the federated request system is under development and will be presented later.

REFERENCES

[1] Adjiman, P.—Chatalic, P.—Goasdoué, F.—Rousset, M.-C.—Simon, L.: SomeWhere in the Semantic Web. In: Fages, F., Soliman, S.(Eds.): Principles and Practice of Semantic Web Reasoning (PPSWR 2005). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3703, 2005, pp. 1–16, doi: 10.1007/11552222_1.

[2] Adjiman, P.—Chatalic, P.—Goasdoué, F.—Rousset, M.-C.—Simon, L.: Distributed Reasoning in a Peer-to-Peer Setting: Application to the Semantic

Web. Journal of Artificial Intelligence Research, Vol. 25, 2006, pp. 269–314, doi: 10.1613/jair.1785.

[3] Adjiman, P.—Goasdoué, F.—Rousset, M.-C.: SomeRDFS in the Semantic Web. In: Spaccapietra, S. et al. (Eds.): Journal on Data Semantics VIII. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4380, 2007, pp. 158–181, doi: 10.1007/978-3-540-70664-9_6.

[4] Aljalbout, S.—Flaquet, G.—Buchs, D.: Practical Implementation of Contextual Reasoning on the Semantic Web. Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2018) – Volume 2: KEOD. 2018, pp. 255–262, doi: 10.5220/0006936802550262.

[5] Baader, F.—Calvanese, D.—McGuinness, D. L.—Nardi, D.—Patel-Schneider, P. F. (Eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003, doi: 10.1017/CBO9780511711787.

[6] Baader, F.—Lutz, C.—Sturm, H.—Wolter, F.: Fusions of Description Logics and Abstract Description Systems. Journal of Artificial Intelligence Research, Vol. 16, 2002, No. 1, pp. 1–58.

[7] Bao, J.—Voutsadakis, G.—Slutzki, G.—Honavar, V. G.: Package-Based Description Logics. In: Stuckenschmidt, H., Parent, C., Spaccapietra, S. (Eds.): Modular Ontologies. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5445, 2009, pp. 349–371, 2009, doi: 10.1007/978-3-642-01907-4_16.

[8] Borgida, A.—Serafini, L.: Distributed Description Logics: Assimilating Information from Peer Sources. In: Spaccapietra, S., March, S., Aberer, K. (Eds.): Journal on Data Semantics I. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2800, 2003, pp. 153–184, doi: 10.1007/978-3-540-39733-5_7.

[9] Bouquet, P.—Giunchiglia, F.—van Harmelen, F.—Serafini, L.—Stuckenschmidt, H.: C-OWL: Contextualizing Ontologies. In: Fensel, D., Sycara, K. P., Mylopoulos, K. (Eds.): The Semantic Web – ISWC 2003. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2870, 2003, pp. 164–179, doi: 10.1007/978-3-540-39718-2_11.

[10] Cali, A.—Calvanese, D.—de Giacomo, G.—Lenzerini, M.: On the Expressive Power of Data Integration Systems. Proceedings of the 21st International Conference on Conceptual Modeling (ER '02), Springer, Berlin, Heidelberg, 2002, pp. 338–350, doi: 10.1007/3-540-45816-6_33.

[11] Calvanese, D.—Franconi, E.—Haarslev, V.—Lembo, D.—Motik, B.—Tessaris, S.—Turhan, A.-Y. (Eds.): Proceedings of the 20th International Workshop on Description Logics (DL '07), June 2007, Brixen/Bressanone, Italy, Bolzano University Press, 2007. ISBN: 978-88-6046-008-0.

[12] Le Duc, C.—Lamolle, M.—Zimmermann, A.—Curé, O.: DRAOn: A Distributed Reasoner for Aligned Ontologies. In: Bail, S., Glimm, B., Gonçalves, R., Jiménez-Ruiz, E., Kazakov, Y., Matentzoglu, N., Parsia, B. (Eds.): Informal Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE-2013). CEUR Workshop Proceedings, Vol. 1015, 2013, pp. 81–86.

[13] Euzenat, J.: An API for Ontology Alignment. In: McIlraith, S., Plexousakis, D., van Harmelen, F. (Eds.): The Semantic Web – ISWC 2004. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3298, 2004, pp. 698–712, doi: 10.1007/978-3-540-30475-3_48.

[14] Euzenat, J.—Shvaiko, P.: Ontology Matching. Springer-Verlag, Heidelberg, 2007.

[15] Michel, F.: Integrating Heterogeneous Data Sources in the WEB of Data. Ph.D. thesis, Université Côte d'Azur, France, 2017.

[16] Ghidini, C.—Giunchiglia, F.: Local Models Semantics, or Contextual Reasoning = Locality + Compatibility. Artificial Intelligence, Vol. 127, 2001, No. 2, pp. 221–259, doi: 10.1016/s0004-3702(01)00064-9.

[17] Ghidini, C.—Serafini, L.: Distributed First Order Logic. In: Gabbay, D. M., de Rijke, M. (Eds.): Frontiers of Combining Systems 2. Research Studies Press, Studies in Logic and Computation, Vol. 7, 2000, pp. 121–139.

[18] Ghidini, C.—Serafini, L.—Tessaris, S.: Complexity of Reasoning with Expressive Ontology Mappings. In: Eschenbach, C., Grüninger, M. (Eds.): Formal Ontology in Information Systems. IOS Press, Frontiers in Artificial Intelligence and Applications, Vol. 183, 2008, pp. 151–163, doi: 10.3233/978-1-58603-923-3-151.

[19] Goczyła, K.—Waloszek, A.—Waloszek, W.: Contextualization of a DL Knowledge Base. Proceedings of the 2007 International Workshop on Description Logics (DL 2007), 2007, Art. No. 55.

[20] Goczyła, K.—Waloszek, A.—Waloszek, W.—Zawadzka, T.: Theoretical and Architectural Framework for Contextual Modular Knowledge Bases. In: Bembenik, R., Skonieczny, L., Rybinski, H., Kryszkiewicz, M., Niezgodka, M. (Eds.): Intelligent Tools for Building a Scientific Information Platform. Springer, Berlin, Heidelberg, Studies in Computational Intelligence, Vol. 467, 2013, pp. 257–280, doi: 10.1007/978-3-642-35647-6_18.

[21] Joseph, M.: Query Answering over Contextualized RDF/OWL Rules: Decidable Classes. Ph.D. thesis, University of Trento, 2015.

[22] Joseph, M.—Serafini, L.: Simple Reasoning for Contextualized RDF Knowledge. In: Kutz, O., Schneider, T. (Eds.): Modular Ontologies. IOS Press, Frontiers in Artificial Intelligence and Applications, Vol. 230, 2011, pp. 79–93, doi: 10.3233/978-1-60750-799-4-79.

[23] Kazakov, Y.—Ponomaryov, D.: On the Complexity of Semantic Integration of OWL Ontologies. In: Artale, A., Glimm, B., Kontchakov, R. (Eds.): Proceedings of the 30th International Workshop on Description Logics (DL 2017). CEUR Workshop Proceedings, Vol. 1879, 2017, Art. No. 59.

[24] Klai, S.—Zimmermann, A.—Khadir, M. T.: Multi-Level Networked Knowledge Base: DDL Reasoning. In: Bellatreche, L., Pastor, Ó., Almendros Jiménez, J., Aït-Ameur, Y. (Eds.): Model and Data Engineering (MEDI 2016). Springer, Cham, Lecture Notes in Computer Science, Vol. 9893, 2016, pp. 118–131, doi: 10.1007/978-3-319-45547-1_10.

[25] Klai, S.—Zimmermann, A.—Khadir, M. T.: Multi-Level Networked Knowledge: Representation and DL Reasoning. International Journal of Metadata, Semantics and Ontologies, Vol. 11, 2016, No. 1, pp. 1–15, doi: 10.1504/ijmso.2016.078101.

[26] Klarman, S.: Reasoning with Contexts in Description Logics. Ph.D. thesis, Vrije Universiteit, Amsterdam, January 2013.

[27] Kutz, O.—Lutz, C.—Wolter, F.—Zakharyaschev, M.: $\mathcal{E}$-Connections of Abstract Description Systems. Artificial Intelligence, Vol. 156, 2004, No. 1, pp. 1–73, doi: 10.1016/j.artint.2004.02.002.

[28] McCarthy, J. L.: Notes on Formalizing Context. Proceedings of the 13$^{\text{th}}$ International Joint Conference on Artificial Intelligence (IJCAI'93), Vol. 1, 1993, pp. 555–560.

[29] Paschke, A.—Schäfermeier, R.: OntoMaven – Maven-Based Ontology Development and Management of Distributed Ontology Repositories. In: Nalepa, G., Baumeister, J. (Eds.): Synergies Between Knowledge Engineering and Software Engineering. Springer, Cham, Advances in Intelligent Systems and Computing, Vol. 626, 2018, pp. 251–273, doi: 10.1007/978-3-319-64161-4_12.

[30] Serafini, L.—Tamilin, A.: DRAGO: Distributed Reasoning Architecture for the Semantic Web. In: Gómez-Pérez, A., Euzenat, J. (Eds.): The Semantic Web: Research and Applications (ESWC 2005). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3532, 2005, pp. 361–376, doi: 10.1007/11431053_25.

[31] Sirin, E.—Parsia, B.—Cuenca Grau, B.—Kalyanpur, A.—Katz, Y.: Pellet: A Pratical OWL-DL Reasoner. Journal of Web Semantics, Vol. 5, 2007, No. 2, pp. 51–53, doi: 10.1016/j.websem.2007.03.004.

[32] Vouros, G.—Santipantakis, G. M.: Distributed Reasoning with $\text{E}^{\text{DDL}}_{\text{HQ}^+}$ SHIQ. In: Schneider, J., Walther, D. (Eds.): Proceedings of the 6$^{\text{th}}$ International Workshop on Modular Ontologies, Graz, Austria, July 24, 2012. CEUR Workshop Proceedings, Vol. 875, 2012.

[33] Zimmermann, A.: Integrated Distributed Description Logics. In: Calvanese, D., Franconi, E., Haarslev, V., Lembo, D., Motik, B., Tessaris, S., Turhan, A.-Y. (Eds.): Proceedings of the 20$^{\text{th}}$ International Workshop on Description Logics (DL'07), 2007, pp. 507–514.

[34] Zimmermann, A.—Euzenat, J.: Three Semantics for Distributed Systems and Their Relations with Alignment Composition. In: Cruz, I. F., Decker, S., Allemang, D., Preist, Ch., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (Eds.): The Semantic Web – ISWC 2006. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4273, 2006, pp. 16–29, doi: 10.1007/11926078_2.

[35] Zimmermann, A.—Giménez-García, J. M.: Integrating Context of Statements within Description Logics. In arXiv preprint arXiv:1709.04970, 2017.

Sihem Klai is Assistant Professor at Computer Science Department and member of the Labged laboratory at the University of Badji Mokhtar Annaba, Algeria. Her research interests include semantic web and knowledge representation and reasoning with multi-contextual information on the Web.

Antoine Zimmermann is Associate Professor at École des mines de Saint-Étienne, France, and conducting research in Laboratoire Hubert Curien. He has been involved in semantic web research since 2004, focusing on knowledge representation and reasoning with multi-contextual information on the Web. He is a regular PC member of the major semantic web and AI conferences (IJCAI, WWW, ISWC, ESWC, ECAI) and is a member of the editorial board of the Journal of Web Semantics. His work notably applies to smart cities, and since 2015, he coordinates the national projects OpenSensingCity. He has participated in the W3C groups OWL 2, RDF 1.1, and Spatial Data on the Web.

Mohamed Tarek Khadir graduated from the University of Badji Mokhtar Annaba, Algeria, with a state engineering degree in electronics majoring in control, in 1995. After two years' work in the computer industry, he undertook an M. Eng. at Dublin City University, Ireland graduating with first class honours in 1998. He received his Ph.D. degree from National University of Ireland, Maynooth, Ireland in 2002. He then continued with this institution as Post-Doctoral Researcher until September 2003 when he joined the Department of Computer Science at University Badji Mokhtar Annaba, Algeria, as Senior Lecturer. He succeeded in obtaining the HDR (Habilitation to Direct Research) in January 2005, conferring him the title of Senior Lecturer. He was nominated Full Professor and Head of Research in December 2010.

Computing and Informatics, Vol. 38, 2019, 151–172, doi: 10.31577/cai_2019_1_151

LOSSY COMPRESSIVE SENSING BASED ON ONLINE DICTIONARY LEARNING

İrem Ülkü

Department of Electrical and Electronics Engineering
Çankaya University
Eskişehir Yolu 29. Km
06790 Ankara, Turkey
e-mail: iremulku@cankaya.edu.tr

Ersin Kizgut

Instituto Universitario de Matemática Pura y Applicada (IUMPA)
Universitat Politècnica de València
E-46071 Valencia, Spain
e-mail: erkiz@upv.es

Abstract. In this paper, a lossy compression of hyperspectral images is realized by using a novel online dictionary learning method in which three dimensional datasets can be compressed. This online dictionary learning method and blind compressive sensing (BCS) algorithm are combined in a hybrid lossy compression framework for the first time in the literature. According to the experimental results, BCS algorithm has the best compression performance when the compression bit rate is higher than or equal to 0.5 bps. Apart from observing rate-distortion performance, anomaly detection performance is also tested on the reconstructed images to measure the information preservation performance.

Keywords: Hyperspectral imaging, compression algorithms, dictionary learning, sparse coding

Mathematics Subject Classification 2010: 68U10, 94A08

1 INTRODUCTION

Hyperspectral images are composed of hundreds of contiguous narrow (generally $0.010\,\mu$m) spectral bands from the visible region (0.4–0.7 μm) to the near-infrared region (about $2.4\,\mu$m) of the electromagnetic spectrum. Hyperspectral images have a huge image size. Therefore, to cope with storage or transmission issues, and to match the available transmission bandwidth in the downlink operation, the hyperspectral image compression is compulsory. Compression can be realized as lossless or lossy. Lossy compression algorithms can reach high compression ratios while experiencing information loss. Quality metrics should capture the degradation which occurs in the image.

Classification of lossy and lossless compression methods is canonically fourfold: prediction-based [24, 31], transformation-based [9, 28], vector quantization (VQ)-based [32], and sparse representation-based [18]. One of the transformation-based algorithms is the principal component analysis (PCA). The PCA algorithm realizes the decorrelation of spectral bands. The improved version of this algorithm is compressive-projection principal component analysis (CPPCA) algorithm [11].

Sparse representation-based methods appear to distinguish among others with its scheme. Rather than using a pre-defined dictionary, such methods learn it directly from the observed data. Data-dependent dictionaries are gathered using dictionary learning [6, 27, 41, 48]. Two different learning schemes can be considered, a batch method which uses the whole training set in the learning process at each iteration and an online learning method which processes one sample from the entire training set at each iteration in an alternating fashion. By using the singular value decomposition (SVD), K-SVD algorithm is developed which can be given as a typical example of batch methods [7]. In the work [21], online dictionary learning algorithm is proposed.

Using dictionary learning in the lossy hyperspectral image compression algorithms is quite common [18, 38, 40]. In the literature [36, 37], it is shown that online dictionary learning algorithm is more effective in processing large datasets with sequentially arriving samples such as hyperspectral images. This sparse representation process finds the sparsest solution, which means solving the non-deterministic polynomial-time hard (NP-hard) ℓ_0-norm minimization problem [10].

Sparse representation algorithms are analyzed in three categories [44, 48]. These are greedy pursuit algorithms, ℓ_p-norm regularization based algorithms, and Bayesian inference algorithms [33, 45]. The most popular greedy pursuit algorithms are the matching pursuit (MP) algorithm [22], the orthogonal matching pursuit (OMP) algorithm [35], the generalized OMP (gOMP) algorithm [39] and the compressive sampling matching pursuit (CoSaMP) algorithm [25].

We may think of ℓ_p-norm regularization algorithms as of two kinds depending on the value of p, namely, for $p \geq 1$ and $0 < p < 1$. In $p \geq 1$ category, only the ℓ_1-norm minimization is sufficiently sparse [48]. The ℓ_1-norm minimization algorithms are divided into three such as constraint based, proximity based, and homotopy based optimization algorithms. The constraint based optimization algorithms category includes the truncated Newton based interior-point method (TNIPM) algorithm [20],

the alternating direction method of the multipliers (ADMM) algorithm [5] and the active-set algorithm with either a primal or dual type [12]. The ADMM algorithm is used to solve the least absolute shrinkage and selection operator (LASSO) problem. The dual active-set algorithm is used to solve the basis pursuit (BP) problem.

The proximity based optimization algorithms category covers the sparse reconstruction by separable approximation (SpaRSA) algorithm [42], the general iterative shrinkage and thresholding (GIST) algorithm [15], the Shotgun algorithm [13] and the augmented Lagrangian method (ALM) algorithm which consists of the primal ALM (PALM) and dual ALM (DALM) [43].

Basic homotopy based algorithms are the LASSO homotopy algorithm [8] and the basis pursuit denoising (BPDN) homotopy algorithm [3].

The generalized iterated shrinkage algorithm (GISA) and the focal underdetermined system solver (FOCUSS) algorithm [16] are analyzed under the non-convex ℓ_p-norm ($0 < p < 1$) regularization algorithms.

The Bayesian inference algorithms category includes the Bayesian compressive sensing projected Landweber based on three-dimensional bivariate shrinkage plus 3D wavelet packet transform (BCS PL-3DBS + 3DWPT) algorithm [17] and the sparse Bayesian learning (SBL) algorithm [34] increase the performance when OBD-BCS algorithm is used. This expectation is reasonable given the rate-distortion performance results, since OBD-BCS algorithm outperforms the others.

Model-based CS algorithms aim to integrate the structured sparsity models into CS algorithms [4]. An algorithm called JSM-2 is a model-based CS algorithm [25].

The blind compressed sensing (BCS) algorithm solves the CS problem without prior knowledge of the sparsity basis [46]. In this case, to guarantee the unique solution, three constraints are considered on the sparsity basis [14]. The algorithm used in the process is called an orthonormal block diagonal BCS (OBD-BCS) algorithm. Each iteration consists of an OMP algorithm and singular value decomposition (SVD) algorithm. There are a handful of studies [29, 23] which use BCS for hyperspectral image reconstruction purpose. In this paper, however, BCS is utilized only in the solution of the sparse coding equation. It is, indeed, not a part of dictionary learning, but rather a tool for finding the sparse coefficients. After finding the sparse coefficients, online dictionary learning is performed as usual. Therefore, previously the BCS algorithm is not implemented with online dictionary learning method. The main contributions of this study are as follows:

1. Different sparse representation algorithms are used to compress hyperspectral images based on online dictionary learning. The compression performances of these algorithms are compared with the performances of the state-of-the-art lossy compression algorithms. This study used the results from the previous studies and therefore only the best performing sparse representation algorithms are included.
2. This is the first time that the BCS algorithm is used in conjunction with the online dictionary learning method for hyperspectral image compression purposes.

3. The anomaly detection technique is applied to further test the information preservation performance of the lossy hyperspectral image compression.

The lossy compression of the hyperspectral images based on online dictionary learning is presented in Section 2. The results are presented in Section 3. The conclusions are given in Section 4.

2 LOSSY COMPRESSION OF HYPERSPECTRAL IMAGES BASED ON DICTIONARY LEARNING

In this section, online dictionary learning based sparse coding on hyperspectral image compression is explained. Sparse coding models the data as the sparse linear combination of the dictionary elements. Dictionary learning is based on learning the dictionary to adapt it to specific data. The online dictionary learning method relies on stochastic approximations and it is suitable for large scale datasets such as hyperspectral images [21]. In this study, the iterative online dictionary learning algorithm is used, which minimizes the surrogate function of the empirical cost under particular constraints at each iteration [21].

2.1 Notation and Problem Statement

In the analysis, the number of bands in the hyperspectral image is represented by n_b, the number of lines in the hyperspectral image is represented by nl, the number of samples in the hyperspectral image is represented by ns, and the number of columns in the dictionary is denoted k. The initial dictionary is expressed as $\mathbf{D}_0 \in \mathbb{R}^{n_b \times k}$. The auxiliary matrices for updating the dictionary are denoted $\mathbf{A}_0 \in \mathbb{R}^{k \times k}$ and $\mathbf{B}_0 \in \mathbb{R}^{n_b \times k}$. The number of iterations is represented by T, the error is expressed as $\mathbf{E} \in \mathbb{R}^{k \times 1}$, the regularization parameter is denoted $\lambda \in \mathbb{R}$, and the sparse coefficients are shown by $\boldsymbol{\alpha} \in \mathbb{R}^k$.

In the dictionary learning process, optimization is performed on the empirical cost by considering a finite training set $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$ in $\mathbb{R}^{n_b \times T}$ [26]. The empirical cost is given as

$$f_T(\mathbf{D}) := \frac{1}{T} \sum_{i=1}^{T} l(\mathbf{x}_i, \mathbf{D}), \tag{1}$$

where $\mathbf{D} \in \mathbb{R}^{n_b \times k}$ represents the dictionary and l expresses the loss function. This loss function corresponds to the optimal value of ℓ_1 norm sparse coding [21] given by the equation

$$l(\mathbf{x}_t, D) := \min_{\boldsymbol{\alpha} \in \mathbb{R}^k} \frac{1}{2} \|\mathbf{x}_t - \mathbf{D}\alpha_t\|_2^2 + \lambda \|\alpha_t\|_1 \tag{2}$$

where λ represents the regularization parameter, x_t expresses the training sample at iteration t and α_t defines the coefficient set at iteration t. In (2), ℓ_1 regularization ensures the sparsity.

A convex set of matrices C is defined to constraint arbitrarily large elements in $\mathbf{D} = [\mathbf{d}_1 \dots \mathbf{d}_k]$ as well as arbitrarily small values of α_t. This convex set C is given

$$C := \{\mathbf{D} \in \mathbb{R}^{n_b \times k} : \|\mathbf{d}_j\| \leq 1, \forall j = 1, \dots k\}. \tag{3}$$

In the optimization, the minimization of the empirical cost $f_T(\mathbf{D})$ with respect to $\mathbf{D}$ is not a convex operation. According to this issue, the process is modified as a joint optimization problem. The modified optimization problem is convex when the sparse coefficients $\mathbf{\Gamma} := [\alpha_1, \dots, \alpha_T] \in \mathbb{R}^{k \times T}$ are fixed, while the optimization is performed with respect to $\mathbf{D}$, and when $\mathbf{D}$ is fixed while the optimization is performed with respect to sparse coefficients $\mathbf{\Gamma}$. This joint optimization problem is as follows:

$$\min_{\mathbf{D} \in C, \mathbf{\Gamma} \in \mathbb{R}^{k \times T}} \frac{1}{T} \sum_{i=1}^{T} \left(\frac{1}{2} \|\mathbf{x}_i - \mathbf{D}\alpha_i\|_2^2 + \lambda \|\alpha_i\|_1 \right). \tag{4}$$

Equation (4) is solved as a convex optimization problem such that $\mathbf{D}$ is minimized when $\mathbf{\Gamma}$ is fixed, and $\mathbf{\Gamma}$ is minimized when $\mathbf{D}$ is fixed, respectively. Instead of minimizing the empirical cost $f_T(\mathbf{D})$, minimizing the expected cost $f(\mathbf{D})$ is much more computationally efficient. This expected cost is given as

$$f(\mathbf{D}) := E_x[l(\mathbf{x}, \mathbf{D})] = \lim_{T \to \infty} f_T(\mathbf{D}) \tag{5}$$

where the unknown probability distribution of the data is utilized to find out the expectation. In the literature, it has been proved that the equality in (5) converges with the probability one [21].

For large scale data sets such as hyperspectral images, stochastic gradient algorithms provide a better rate of convergence [21]. Therefore, in this study, dictionary learning is realized by using projected first order stochastic gradient descent algorithm. According to this algorithm, dictionary $\mathbf{D}$ is updated sequentially and is shown as [2].

$$\mathbf{D}_t = \prod_C [\mathbf{D}_{t-1} - \frac{\rho}{t} \nabla_D \, l(\mathbf{x}_t, \mathbf{D}_{t-1})] \tag{6}$$

where $\mathbf{D}_t$ represents the optimal dictionary at iteration t, ρ presents the gradient step, and $\prod_C$ shows the orthogonal projector on C. It is assumed that the training set $\mathbf{X}$ has i.i.d. samples of the unknown distribution of the particular data [21].

2.2 Algorithm

In this study, an algorithm which consists of two parts is used. These two parts, namely dictionary learning and dictionary update, are solved alternately. The sparse coding equation is solved by using $\mathbf{x}_t$ from the current iteration, and $\mathbf{D}_{t-1}$ from the previous iteration. When α_t is found, the following $\widehat{f}_t(\mathbf{D})$ function is minimized over

set C to obtain an updated dictionary $\mathbf{D}_t$:

$$\widehat{f}_t(\mathbf{D}) := \frac{1}{t}\sum_{i=1}^{t}\frac{1}{2}\|\mathbf{x}_i - \mathbf{D}\alpha_i\|_2^2 + \lambda\|\alpha_i\|_1 \tag{7}$$

where α_i values are obtained. In the literature, it has been proved that the empirical cost $f_t(\mathbf{D})$ and the function $\widehat{f}_t(\mathbf{D})$, which is quadratic in the $\mathbf{D}$ converge is almost surely to the same limit [21]. Therefore, the function $\widehat{f}_t$ is the surrogate for the function f_t. For the large values of t, the function $\widehat{f}_t$ is close to $\widehat{f}_{t-1}$ function. In these circumstances, $\mathbf{D}_t$ is also close to $\mathbf{D}_{t-1}$ such that it is effective to use $\mathbf{D}_{t-1}$ as a warm restart for finding $\mathbf{D}_t$. At each iteration Algorithm 1 finds the sparse coefficients, while Algorithm 2 uses these sparse coefficients to update the current dictionary. Using various different sparse representation algorithms for the solution of the sparse coding equation in Algorithm 1, best performing algorithm can be determined, enabling a comparison between state-of-the-art algorithms. Online dictionary learning, which is the main implementation in Algorithm 2 will be used for all scenarios.

2.2.1 Algorithm 1

In Algorithm 1 the sparse coding equation is solved. Equation (2) is called sparse coding equation. The value of λ is set to 0.1 while T equals 200.

Algorithm 1 Dictionary Learning

1: Construct random initial dictionary $\mathbf{D}_0$
2: Set initial values $\mathbf{A}_0$ and $\mathbf{B}_0$ matrices to zero
3: **for** $t = 1$ to T **do**
4: Choose $\mathbf{x}_t \in \mathbb{R}^{n_b}$ randomly from the image.
5: Solve sparse coding equation.
6: Update $\mathbf{A}_t = \mathbf{A}_{t-1} + \boldsymbol{\alpha}_t\boldsymbol{\alpha}_t^T$, $\mathbf{B}_t = \mathbf{B}_{t-1} + \mathbf{x}_t\boldsymbol{\alpha}_t^T$.
7: Find $\mathbf{D}_t$ using Algorithm 2.
8: **end for**
9: Obtain learned dictionary $\mathbf{D}_t$.

2.2.2 Algorithm 2

In Algorithm 2 dictionary is updated by utilizing the block-coordinate descent with $\mathbf{D}_{t-1}$ as a warm restart. Equation (7) is called as the dictionary update equation. Algorithm 1 and Algorithm 2 are applied in an alternating fashion which is the online learning strategy. Various algorithms are used to solve sparse coding equation (cf. Table 2).

Algorithm 2 Dictionary Update

Calculate $\mathbf{D}_t$ in dictionary update equation
repeat
  for $j = 1$ to k **do**
    Find jth column of $\mathbf{D}_t$, where $\mathbf{D} = [\mathbf{d}_1 \dots \mathbf{d}_k] \in \mathbb{R}^{n_b \times k}$, $\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_k] \in \mathbb{R}^{k \times k}$ and $\mathbf{B} = [\mathbf{b}_1 \dots \mathbf{b}_k] \in \mathbb{R}^{n_b \times k}$
    $\mathbf{u}_j := \frac{1}{A(j,j)}(\mathbf{b}_j - \mathbf{D}\mathbf{a}_j) + \mathbf{d}_j$
    $\mathbf{d}_j = \dfrac{1}{\max(\|\mathbf{u}_j\|_2, 1)}\mathbf{u}_j$
    $E_j = \sqrt{\sum_{n_b} |\mathbf{d}_j^t - \mathbf{d}_j^{t-1}|^2}$
  end for
  $E = \frac{1}{k}\sum_{j=1}^{k} E_j$
until $E <$ Treshold
Use $\mathbf{D}$ in Algorithm 1.

3 RESULTS

The online dictionary learning based hyperspectral image compression is applied by using AVIRIS and Hyperion datasets for all the different sparse representation algorithms [19]. The compression performances of these algorithms are compared with the performances of the state-of-the-art lossy compression algorithms such as BCS PL-3DBS + 3DWPT and CPPCA. BCS PL-3DBS + 3DWPT and CPPCA algorithms are not based on learning while the remaining ones are employed by an online learning scheme. The quality metric tool is the Peak Signal-to-Noise Ratio (PSNR). The bit rate r is calculated in terms of the bits per sample (bps), and the formulation is as follows

$$r = \frac{z}{n_b}(b_d), \quad z < k \tag{8}$$

where z represents the number of sparse coefficients, k defines the size of the dictionary, n_b is the number of bands, and b_d represents the bit depth.

3.1 Datasets

The information about the AVIRIS and Hyperion datasets which are used in this study are given in Table 1 [19].

3.2 AVIRIS Datasets Results

Low Altitude, Lunar Lake, and the Jasper Ridge are used as the AVIRIS datasets (cf. Table 1). In Table 2, the compression performances of different sparse representation algorithms are shown. The quality metric tool, which reflects the compression performance, is PSNR in terms of dB. The PSNR values are calculated against

the compression ratios in terms of the bps. The state-of-the-art algorithms BCS PL-3DBS + 3DWPT and CPPCA, which are given in Table 2, are used for the comparison [17]. The highest three PSNR values per each compression ratio are marked in boldface. If Table 2 is analyzed at the highest compression ratio of 0.5 bps, only the OBD-BCS algorithm involves among the algorithms with the best three PSNR values for all datasets.

AVIRIS HYPERSPECTRAL DATA					
Name	No. Samples	No. Lines	No. Bands	Bit Depth	Year
Jasper Ridge	614	2 587	224	16	1997
Lunar Lake	614	1 432	224	16	1997
Low Altitude	614	3 689	224	16	1996
HYPERION HYPERSPECTRAL DATA					
Name	No. Samples	No. Lines	No. Bands	Bit Depth	Year
Lake Monona	256	3 176	242	12	2009
Mt. St. Helens	256	3 242	242	12	2009
Erta Ale	256	3 187	242	12	2010
SALINAS-A HYPERSPECTRAL DATA					
	No. Samples	No. Lines	No. Bands	Bit Depth	Year
	83	86	204	12	1998
PAVIA UNIVERSITY HYPERSPECTRAL DATA					
	No. Samples	No. Lines	No. Bands	Bit Depth	Year
	200	200	103	12	2002
INDIANA HYPERSPECTRAL DATA					
	No. Samples	No. Lines	No. Bands	Bit Depth	Year
	145	145	220	12	1992

Table 1. Detailed information of AVIRIS, Hyperion, Salinas-A, Pavia and Indiana hyperspectral datasets

3.3 Hyperion Datasets Results

The Erta Ale, Mt. St. Helens, and Lake Monona images are used as Hyperion datasets (cf. Table 1). In Figures 1, 2 and 3, the PSNR values of these datasets against 0.1, 0.3, and 0.5 bps compression ratios for all sparse representation algorithms, are given. The PSNR values are expressed in terms of dB, and they are plotted against the compression ratios in terms of bps. The corresponding compression ratios of the algorithms with highest three PSNR values are shown in circles.

As seen from Figures 1, 2, and 3, at the highest compression ratio of 0.5 bps, the SpaRSA algorithm appears among the best three algorithms for all the datasets, while the OBD-BCS algorithm is situated among the best three algorithms for the Mt. St. Helens and Lake Monona datasets. Therefore, at high compression

	Lunar Lake Image													
	Sparse Representation Algorithms													
BPS	BCS PL_3DBS + 3DWPT	BP (Dual active set)	gOMP	LASSO (ADMM)	CPPCA	SpaRSA	GIST	BPDN (Homo-topy)	GISA $p = 0.4$	OBD-BCS	SBL	FOCUSS	Shotgun	JSM-2
0.1	54.74	**59.96**	59.79	59.59	47.47	59.88	59.85	59.82	59.74	**60**	**59.97**	59.93	59.79	59.62
0.3	61.74	**70.16**	**70.28**	68.85	60.98	69.78	69.79	69.04	67.64	**69.99**	68.15	68.52	69.73	67.74
0.5	67.08	**73.24**	72.68	**73.52**	70.01	72.82	73.21	72	71.9	**73.56**	72.74	73.09	71.81	68.9
	Jasper Ridge Image													
	Sparse Representation Algorithms													
BPS	BCS PL_3DBS + 3DWPT	BP (Dual active set)	gOMP	LASSO (ADMM)	CPPCA	SpaRSA	GIST	BPDN (Homo-topy)	GISA $p = 0.4$	OBD-BCS	SBL	FOCUSS	Shotgun	JSM-2
0.1	**61.34**	59.55	58.37	59.54	48.43	59.51	**59.68**	59.57	59.58	59.22	**59.6**	59.54	59.57	59.51
0.3	69.38	**73.85**	73.84	73.34	72.19	**73.89**	73.62	68.58	69.41	**73.97**	72.07	73.16	69.87	71.19
0.5	72.62	**76.55**	74.92	75.2	**76.82**	75.07	75.37	71.81	71.57	75.45	75.01	74.19	74.47	72.1
	Low Altitude Image													
	Sparse Representation Algorithms													
BPS	BCS PL_3DBS + 3DWPT	BP (Dual active set)	gOMP	LASSO (ADMM)	CPPCA	SpaRSA	GIST	BPDN (Homo-topy)	GISA $p = 0.4$	OBD-BCS	SBL	FOCUSS	Shotgun	JSM-2
0.1	56.78	59.41	59.4	59.3	30.2	**59.47**	58.83	59.32	59.28	**59.44**	58.6	59.37	59.42	**59.43**
0.3	64.21	69.23	70.01	70.67	**71.31**	70.69	70.15	69.56	68.45	70.57	**70.74**	**70.83**	70.73	69.42
0.5	69.95	71.71	71.14	**73.17**	**76.4**	72.49	72.24	72.44	70.46	**72.54**	72.46	72.17	72.27	71.26

Table 2. Compression performance comparison between sparse representation algorithms and state-of-the-art compression algorithms [17]

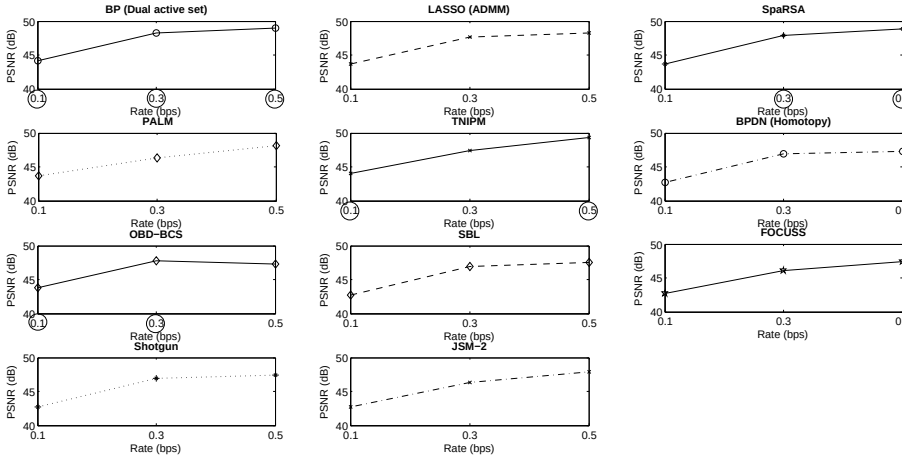


Figure 1. Compression performances of sparse representation algorithms for Erta Ale image (cf. Table 1)

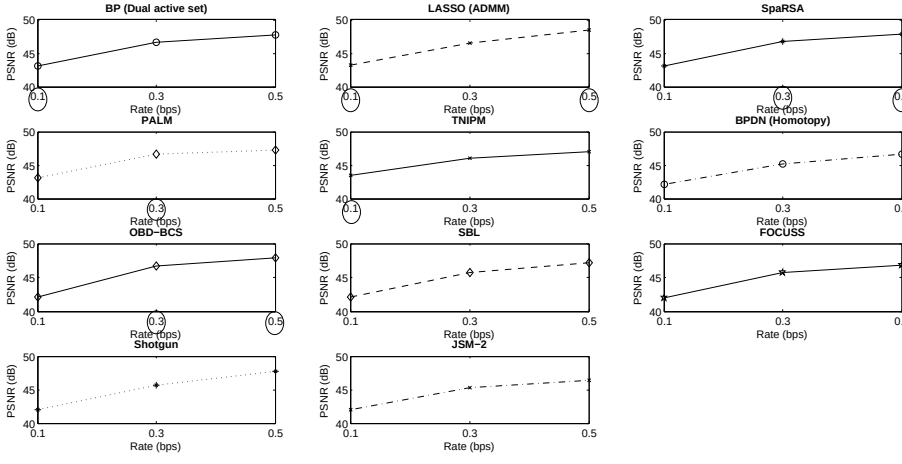


Figure 2. Compression performances of sparse representation algorithms for Mt. St. Helens image (cf. Table 1)

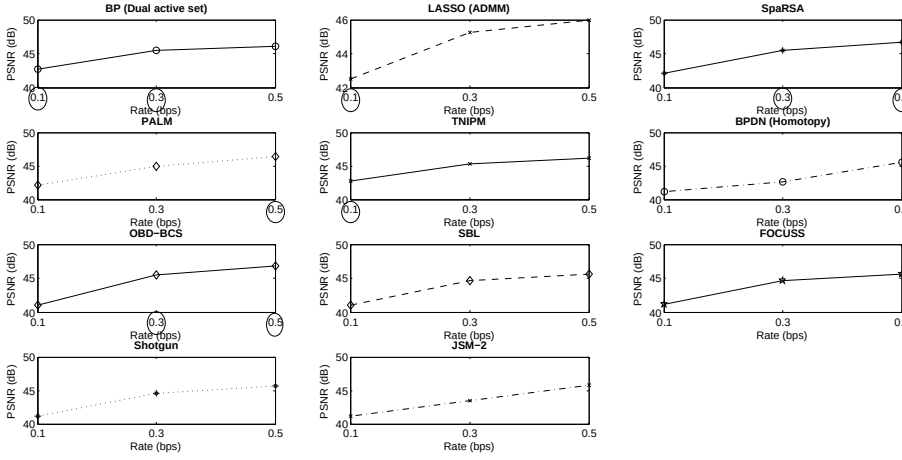


Figure 3. Compression performances of sparse representation algorithms for Lake Monona image (cf. Table 1)

ratios, the SpaRSA and OBD-BCS algorithms show better compression performances.

3.4 Comparison with Several HCS Methods

In literature, a novel reweighted Laplace prior based hyperspectral compressive sensing (RLPHCS) method named as RLPHCS_Cov outperforms several state-of-the-art HCS algorithms [47]. This compression method is not based on learning. For further comparison, compression performance of the algorithm RLPHCS_Cov is compared to that of the sparse representation algorithms based on online dictionary learning.

The signal to noise ratio (SNR) is fixed at 20dB. Pavia University and Indiana datasets are used (cf. Table 1). Figures 4 and 5 show PSNR curves of different algorithms at various bps levels when Pavia University and Indiana datasets are used, respectively. Online dictionary learning (ODL) and hyperspectral compressive sensing (HCS) abbreviations are used.

Figures 4 and 5 indicate that the reconstruction performance of the OBD-BCS (ODL) algorithm is superior to that of the other algorithms at 0.5 bps level. Although for 0.5 bps compression level the OBD-BCS (ODL) algorithms is better for both datasets, setting the compression ratio to moderate levels such as 0.3 bps yields better RLPHCS_Cov performance for Pavia University dataset.

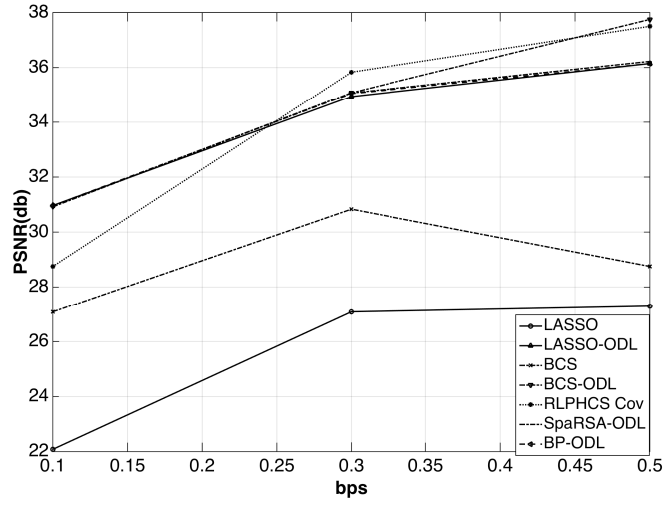


Figure 4. The reconstruction performances of different methods for Pavia University dataset when SNR is 20 db

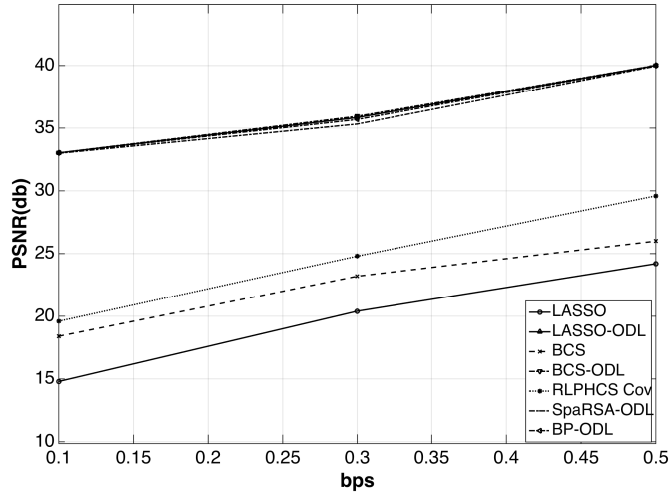


Figure 5. The reconstruction performances of different methods for Indiana dataset when SNR is 20 db

3.5 Compression Performance Analysis of OBD-BCS Algorithm

According to Table 2, the OBD-BCS algorithm involves among the top three algorithms at the highest compression ratio of 0.5 bps for all the datasets.

In Figures 1, 2, and 3, the OBD-BCS algorithm belongs to the top three algorithms with the highest PSNR values at the highest ratio of 0.5 bps for the Mt. St. Helens and Lake Monona datasets.

The results indicate that the OBD-BCS algorithm shows a better compression performance when the compression ratio gets higher. OBD-BCS algorithm is being considered as a compressive sensing framework. However, in this study it is only used in Algorithm 1 to solve the sparse coding equation. Since the OBD-BCS algorithm itself includes a dictionary learning process, learning is applied not just in Algorithms 2 for online dictionary learning, but also in Algorithm 1 while finding the sparse coefficients. It is expected that using these more accurate sparse coefficients in online dictionary learning will increase the performance when OBD-BCS algorithm is used. This expectation is reasonable given the rate-distortion performance results, since OBD-BCS algorithm outperforms the others.

3.6 Anomaly Detection Application

The anomaly detection is applied to make a further comparison between various sparse representation methods. It is a useful tool for assessing the information preservation ability of these methods. Reed-Xiaoli (RX) anomaly detection algorithm is used [30].

Spectral signature which belongs to the input signal is compared with the mean values of each spectral band by using Mahalanobis distance,

$$\delta_{RX}(\mathbf{x}_i) = (\mathbf{x}_i - \mathbf{M})^T \mathbf{Cov}^{-1}(\mathbf{x}_i - \mathbf{M}) \tag{9}$$

where $\mathbf{x}_i \in \mathbb{R}^{n_b}$, $\mathbf{M}$ represents the mean of each spectral band and $\mathbf{Cov}$ indicates the spectral covariance matrix. Covariance matrix $\mathbf{Cov}$ is as follows:

$$\mathbf{Cov} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \mathbf{M})(\mathbf{x}_i - \mathbf{M})^T \tag{10}$$

where $N = nl \times ns$ and $i = 1, \ldots N$.

Anomalous region is assumed to be present if $\delta_{RX}(\mathbf{x}_i) \geq \eta$ condition is satisfied, where η represents the threshold value. The most appropriate threshold value is the one that is obtained from the desired false alarm probability. Anomaly detection is applied on Salinas-A and Low Altitute hyperspectral datasets (cf. Table 1) [1]. Sparse representation based on online dictionary learning algorithms such as BP by using dual active set algorithm, LASSO by using ADMM algorithm, SpaRSA and OBD-BCS are utilized.

The anomaly detection results are illustrated in Figure 10 for Salinas-A dataset. First, the anomaly detection is applied on the original hyperspectral dataset whose results are presented in Figure 10 a). The desired anomaly is marked with a circle. Figure 10 b)–e) depict anomaly detection results for OBD-BCS, BP by using dual active set, SpaRSA and LASSO algorithms at 0.5, 0.3 and 0.1 bps levels, respectively. None of the algorithms is able to detect the desired anomaly at 0.1 bps bit rate. Among the anomaly detection results at 0.5 bps bit rate, OBD-BCS algorithm seems to provide the best performance.

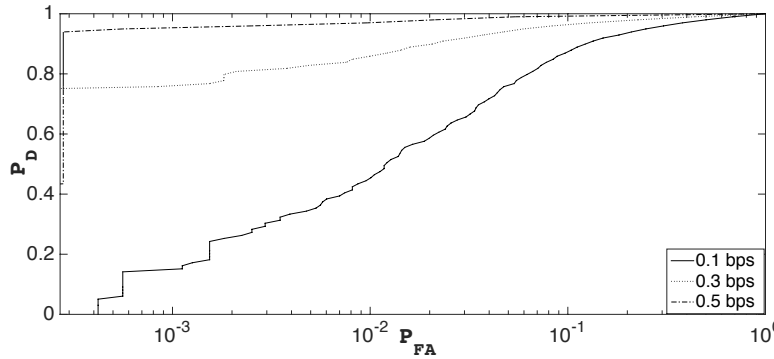


Figure 6. ROC Semilog curves for Salinas-A dataset at 0.1, 0.3 and 0.5 bps by using OBD-BCS

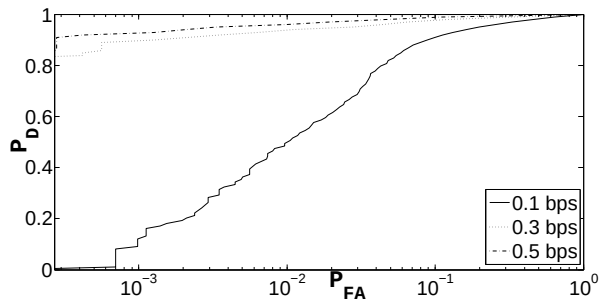


Figure 7. ROC Semilog curves for Salinas-A dataset at 0.1, 0.3 and 0.5 bps by using LASSO algorithm

The PSNR values of each sparse representation algorithms are also presented in Table 3 for 0.1, 0.3 and 0.5 bit rates in such a way to further strengthen the anomaly detection results obtained in Figure 10. The two highest PSNR values are marked in boldface.

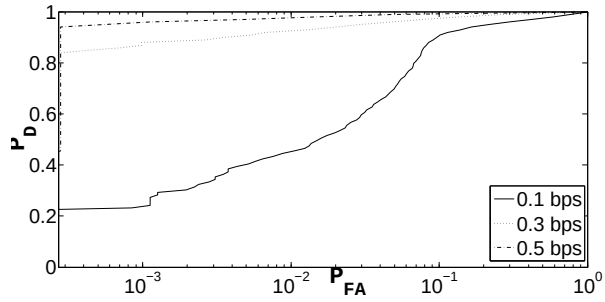


Figure 8. ROC Semilog curves for Salinas-A dataset at 0.1, 0.3 and 0.5 bps by using BP by using dual active set algorithm

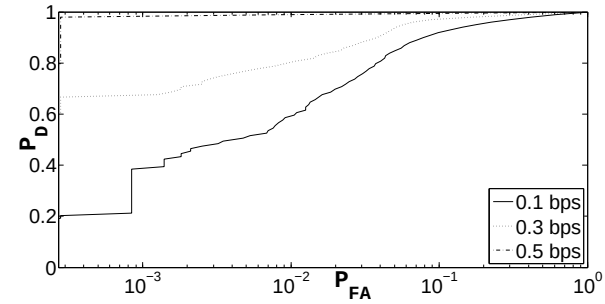


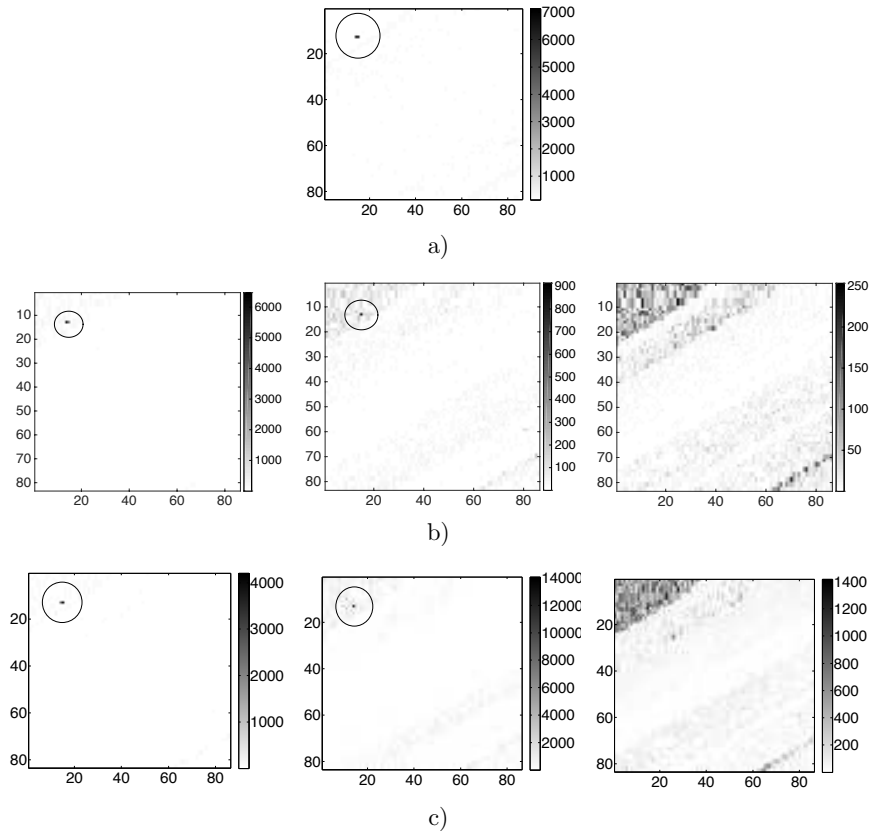
Figure 9. ROC Semilog curves for Salinas-A dataset at 0.1, 0.3 and 0.5 bps by using SpaRSA

Anomaly detection performances of different sparse representation algorithms can be assessed using receiver operating characteristic (ROC) curves. The ROC curves plot detection probability versus false alarm probability. ROC curves are plotted with a logarithmic x axis for better illustration.

Figure 6 shows the ROC Semilog curves of OBD-BCS algorithm at 0.1 bps, 0.3 bps and 0.5 bps rates when Salinas-A hyperspectral dataset is used. The probability of detection is denoted by PD and the probability of false alarm is denoted by PFA. Anomaly detection result at 0.5 bps rate is significantly better than those of the 0.3 bps and 0.1 bps levels.

The ROC Semilog curves of BP by using dual active set algorithm at 0.1, 0.3 and 0.5 bps bit rates are depicted in Figure 8. For the Salinas-A dataset, the ROC Semilog curves of SpaRSA and LASSO algorithm at various bit rates are illustrated in Figures 9 and 7, respectively.

In order to further evaluate the ROC curves, the area under curve (AUC) is employed as a performance metric that can be obtained by calculating the area



under the ROC curve. Calculated AUC values are presented in Table 3. In Table 3 for Salinas-A dataset, it can be seen that the best result is from OBD-BCS at 0.5 bps bit rate which is 0.9945.

Results in Table 3 and Figures 6–7 demonstrate that the detection performance of OBD-BCS algorithm is better than that of the other algorithms for the case where the bit rate is high such as 0.5 bps. The illustrations in Figure 10 also suggest that the detection performance of OBD-BCS algorithm is the best of all at 0.5 bps bit rate.

According to the values in Table 3, OBD-BCS algorithm is among the best two algorithms in terms of PSNR values at 0.5, 0.3 and 0.1 bps rates for Low-Altitude dataset. Particularly at 0.5 bps level, OBD-BCS has PSNR value of 73.56 which is the highest. The superiority of OBD-BCS algorithm at 0.5 bps rate is supported by the results in Table 3 for Low-Altitude dataset. At 0.5 bps, OBD-BCS algorithm has the highest AUC value which is 0.9943.

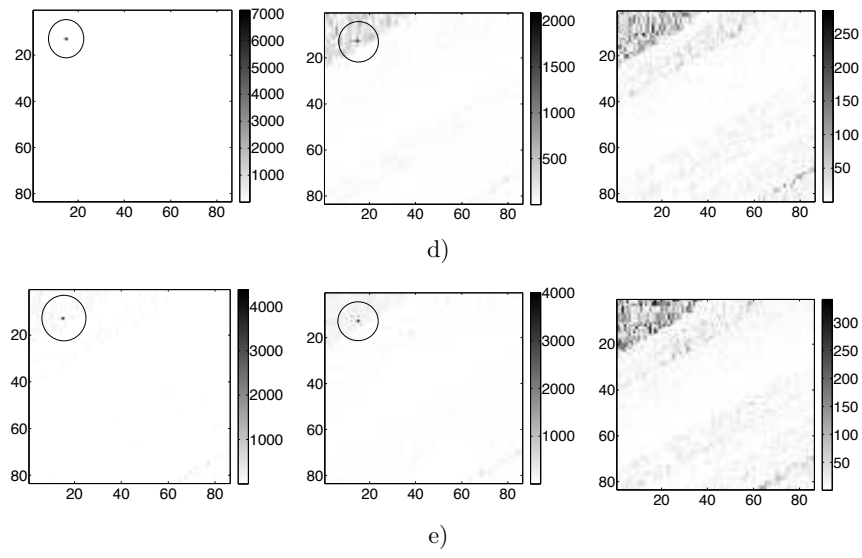


Figure 10. RX anomaly detection results of the Salinas-A hyperspectral image: a) original image, b) OBD-BCS with 0.5 bps, 0.3 bps and 0.1 bps, c) BP with 0.5 bps, 0.3 bps, and 0.1 bps, d) SpaRSA with 0.5 bps, 0.3 bps, and 0.1 bps, e) LASSO with 0.5 bps, 0.3 bps, and 0.1 bps

4 CONCLUSION

Sparse representation algorithms from many different categories are applied for the purpose of hyperspectral image compression based on online dictionary learning. The hyperspectral compression performance of these sparse representation algorithms are analyzed by further analyzing the OBD-BCS algorithm. By analyzing the results of all the datasets, the OBD-BCS algorithm shows the best compression performance at high compression ratios. At a 0.5 bps ratio, it involves among the best three algorithms at most for all the datasets. Other algorithms with good compression performances at high ratios are BP by using dual active set, the LASSO by using ADMM, and the SpaRSA algorithms.

According to the anomaly detection results, compressed image at bit rates of 0.5 bps or higher can be used as an estimate of the original hyperspectral image. Anomaly detection or similar real-world applications can be applied on the compressed hyperspectral image instead of the original one. Anomaly detection results further prove that the OBD-BCS algorithm has a better information preservation performance than that of the other algorithms as the bit rate gets higher.

	Salinas-A							
	Sparse Respresentation Algorithms							
BPS	BP		OBD-BCS		LASSO		SpaRSA	
	PSNR	AUC	PSNR	AUC	PSNR	AUC	PSNR	AUC
0.1	36.62	0.96	**36.67**	0.9464	**36.65**	0.9741	36.58	0.9424
0.3	41.54	0.9934	**41.89**	0.9929	41.16	0.9799	**42.61**	0.992
0.5	43.95	0.9943	**43.98**	0.9945	43.74	0.9928	**43.96**	0.9943
	Low Altitude							
	Sparse Respresentation Algorithms							
BPS	BP		OBD-BCS		LASSO		SpaRSA	
	PSNR	AUC	PSNR	AUC	PSNR	AUC	PSNR	AUC
0.1	**59.96**	0.9917	**60**	0.9887	59.59	0.9906	59.88	0.9896
0.3	**70.16**	0.9932	**69.99**	0.9936	68.85	0.9932	69.78	0.9914
0.5	73.24	0.9942	**73.56**	0.9943	**73.52**	0.9931	72.82	0.9937

Table 3. PSNR values of sparse representation algorithms

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful and constructive comments that greatly contributed to improving the final version of the paper. The authors would also like to thank Prof. Dr. Halil T. Eyyuboğlu for useful suggestions and comments. This research was partially supported by the Turkish Scientific and Technical Research Council.

REFERENCES

[1] Hyperspectral Remote Sensing Scenes. `http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes`, accessed March 2018.

[2] Aharon, M.—Elad, M.: Sparse and Redundant Modeling of Image Content Using an Image-Signature-Dictionary. SIAM Journal on Imaging Sciences, Vol. 1, 2008, No. 3, pp. 228–247, doi: 10.1137/07070156X.

[3] Asif, M. S.—Romberg, J.: Dynamic Updating for ℓ_1 Minimization. IEEE Journal of Selected Topics in Signal Processing, Vol. 4, 2010, No. 2, pp. 421–434, doi: 10.1109/JSTSP.2009.2039174.

[4] Baraniuk, R. G.—Cevher, V.—Duarte, M. F.—Hegde, C.: Model-Based Compressive Sensing. IEEE Transactions on Information Theory, Vol. 56, 2010, No. 4, pp. 1982–2001, doi: 10.1109/TIT.2010.2040894.

[5] Boyd, S.—Parikh, N.—Chu, E.—Peleato, B.—Eckstein, J.: Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. Foundations and Trends® in Machine Learning, Vol. 3, 2011, No. 1, pp. 1–122, doi: 10.1561/2200000016.

[6] Charles, A. S.—Olshausen, B. A.—Rozell, C. J.: Learning Sparse Codes for Hyperspectral Imagery. IEEE Journal of Selected Topics in Signal Processing, Vol. 5, 2011, No. 5, pp. 963–978, doi: 10.1109/jstsp.2011.2149497.

[7] Chen, G.—Needell, D.: Compressed Sensing and Dictionary Learning. Proceedings of Symposia in Applied Mathematics, Vol. 73, 2016, pp. 210–241, doi: 10.1090/psapm/073.

[8] Donoho, D. L.—Tsaig, Y.: Fast Solution of ℓ_1-Norm Minimization Problems when the Solution May Be Sparse. Online, Stanford University, Stanford, CA, 2006, `https://statistics.stanford.edu/sites/g/files/sbiybj6031/f/2006-18.pdf`.

[9] Dragotti, P. L.—Poggi, G.—Ragozini, A. R. P: Compression of Multispectral Images by Three-Dimensional SPIHT Algorithm. IEEE Transactions on Geoscience and Remote Sensing, Vol. 38, 2000, No. 1, pp. 416–428, doi: 10.1109/36.823937.

[10] Elad, M.: Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing. Springer, New York, 2010, doi: 10.1007/978-1-4419-7011-4.

[11] Fowler, J. E.: Compressive-Projection Principal Component Analysis. IEEE Transactions on Image Processing, Vol. 18, 2009, No. 10, pp. 2230–2242, doi: 10.1109/tip.2009.2025089.

[12] Friedlander, M. P.—Saunders, M. A.: A Dual Active-Set Quadratic Programming Method for Finding Sparse Least-Squares Solutions. Online, University of British Columbia, BC, Canada, 2012.

[13] Fu, W. J.: Penalized Regressions the Bridge versus the LASSO. Journal of Computational and Graphical Statistics, Vol. 7, 1998, No. 3, pp. 397–416, doi: 10.1080/10618600.1998.10474784.

[14] Gleichman, S.—Eldar, Y. C.: Blind Compressed Sensing. IEEE Transactions on Information Theory, Vol. 57, 2011, No. 10, pp. 6958–6975, doi: 10.1109/tit.2011.2165821.

[15] Gong, P.—Zhang, C.—Lu, Z.—Huang, J. Z.—Ye, J.: A General Iterative Shrinkage and Thresholding Algorithm for Non-Convex Regularized Optimization Problems. Proceedings of the 30th International Conference on Machine Learning (ICML '13), Vol. 28, 2013, JMLR, pp. 37–45.

[16] Gorodnitsky, I. F.—Rao, B. D.: Sparse Signal Reconstruction from Limited Data Using FOCUSS: A Weighted Minimum Norm Algorithm. IEEE Transactions on Signal Processing, Vol. 45, 1997, No. 3, pp. 600–616, doi: 10.1109/78.558475.

[17] Hou, Y.—Zhang, Y.: Effective Hyperspectral Image Block Compressed Sensing Using Thress-Dimensional Wavelet Transform. 2014 IEEE Geoscience and Remote Sensing Symposium (IGARSS), 2014, pp. 2973–2976, doi: 10.1109/IGARSS.2014.6947101.

[18] Huo, C.—Zhang, R.—Yin, D.—Wu, Q.—Xu, D.: Hyperspectral Data Compression Using Sparse Representation. 2012 4th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPER), IEEE, 2012, pp. 1–4, doi: 10.1109/WHISPERS.2012.6874259.

[19] Kiely, A. B.—Klimesh, M. A.: Exploiting Calibration-Induced Artifacts in Lossless Compression of Hyperspectral Imagery. IEEE Transactions on Geoscience and Remote Sensing, Vol. 47, 2009, No. 8, pp. 2672–2678, doi: 10.1109/tgrs.2009.2015291.

[20] Kim, S. J.—Koh, K.—Lustig, M.—Boyd, S.—Gorinevsky, D.: An Interior-Point Method for Large-Scaleℓ_1-Regularized Least Squares. IEEE Journal of Selected Topics in Signal Processing, Vol. 1, 2007, No. 4, pp. 606–617, doi: 10.1109/JSTSP.2007.910971.

[21] Mairal, J.—Bach, F.—Ponce, J.—Sapiro, G.: Online Learning for Matrix Factorization and Sparse Coding. Journal of Machine Learning Research, Vol. 11, 2010, pp. 19–60, doi: 10.1145/1553374.1553463.

[22] Mallat, S. G.—Zhang, Z.: Matching Pursuits with Time-Frequency Dictionaries. IEEE Transactions on Signal Processing, Vol. 41, 1993, No. 12, pp. 3397–3415, doi: 10.1109/78.258082.

[23] Martin, G.—Bioucas-Dias, J.—Plaza. A.: B-HYCA: Blind Hyperspectral Compressive Sensing. 2015 IEEE International Geoscience and Remote Sensing Symposim (IGARSS'15), Milan, Italy, July, 2015, doi: 10.1109/IGARSS.2015.7326410.

[24] Mielikainen, J.—Toivanen, P.: Lossless Compression of Hyperspectral Images Using a Quantized Index to Lookup Tables. IEEE Geoscience Remote Sensing Letters, Vol. 5, 2008, No. 3, pp. 474–478, doi: 10.1109/LGRS.2008.917598.

[25] Needell, D.—Tropp, J. A.: CoSaMP: Iterative Signal Recovery from Incomplete and Inaccurate Samples. Applied and Computational Harmonic Analysis, Vol. 26, 2009, No. 3, pp. 301–321, doi: 10.1016/j.acha.2008.07.002.

[26] Olshausen, B. A.—Field, D. J.: Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1? Vision Research, Vol. 37, 1997, No. 23, pp. 3311–3325, doi: 10.1016/S0042-6989(97)00169-7.

[27] Du, P.—Xue, Z.—Li, J.—Plaza, A.: Learning Discriminative Sparse Representations for Hyperspectral Image Classification. IEEE Journal of Selected Topics in Signal Processing, Vol. 9, 2015, No. 6, pp. 1089–1104, doi: 10.1109/JSTSP.2015.2423260.

[28] Du, Q.—Fowler, J. E.: Hyperspectral Image Compression Using JPEG2000 and Principal Component Analysis. IEEE Geoscience and Remote Sensing Letters, Vol. 4, 2007, No. 2, pp. 201–205, doi: 10.1109/LGRS.2006.888109.

[29] Rajwade, A.—Kittle, D.—Tsai, T.-H.—Brady, D.—Carin, L.: Coded Hyperspectral Imaging and Blind Compressive Sensing. SIAM Journal on Imaging Sciences, 2013, pp. 782–812, doi: 10.1137/120875302.

[30] Reed, S. I.—Yu, X.: Adaptive Multiple-Band CFAR Detection of an Optical Pattern with Unknown Spectral Distribution. IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 38, 1990, No. 10, pp. 1760–1770, doi: 10.1109/29.60107.

[31] Ricci, M.—Magli, E.: Predictor Analysis for Onboard Lossy Predictive Compression of Multispectral and Hyperspectral Images. Journal of Applied Remote Sensing, Vol. 7, 2013, No. 1, Art. No. 074591, doi: 10.1117/1.JRS.7.074591.

[32] Ryan, M. J.—Arnold, J. F.: The Lossless Compression of AVIRIS Images by Vector Quantization. IEEE Transactions on Geoscience and Remote Sensing, Vol. 35, 1997, No. 5, pp. 546–550, doi: 10.1109/36.581964.

[33] Song, H.—Wang, G.: Sparse Signal Recovery via ECME Tresholding Pursuits. Mathematical Problems in Engineering, Vol. 2012, 2012, Art. No. 478931, 22 pp., doi: 10.1155/2012/478931.

[34] Tipping, M. E.: Sparse Bayesian Learning and the Relevance Vector Machine. Journal of Machine Learning Research, Vol. 1, 2001, pp. 211–244.

[35] Tropp, J. A.—Gilbert, A. C.: Signal Recovery from Random Measurements via Orthogonal Matching Pursuit. IEEE Transactions on Information Theory, Vol. 53, 2007, No. 12, pp. 4655–4666, doi: 10.1109/tit.2007.909108.

[36] Ülkü, İ.—Kizgut, E.: Hyperspectral Compressive Sensing Based on Online Dictionary Learning. Imaging and Applied Optics 2017, Optical Society of America, 2017, Art. No. ITh4E.1, doi: 10.1364/ISA.2017.ITh4E.1.

[37] Ülkü, İ.—Kizgut, E.: Large-Scale Hyperspectral Image Compression via Sparse Representations Based on Online Learning. International Journal of Applied Mathematics and Computer Science, Vol. 28, 2018, No. 1, pp. 197–207, doi: 10.2478/amcs-2018-0015.

[38] Ülkü, İ.—Töreyin, B. U.: Sparse Representations for Online-Learning-Based Hyperspectral Image Compression. Applied Optics, Vol. 54, 2015, No. 29, pp. 8625–8631, doi: 10.1364/AO.54.008625.

[39] Wang, J.—Kwon, S.—Shim, B.: Generalized Orthogonal Matching Pursuit. IEEE Transactions on Signal Processing, Vol. 60, 2012, No. 12, pp. 6202–6216, doi: 10.1109/TSP.2012.2218810.

[40] Wang, Z.—Nasrabadi, N. M.—Huang, T. S.: Spatial-Spectral Classification of Hyperspectral Images Using Discriminative Dictionary Designed by Learning Vector Quantization. IEEE Transactions on Geoscience and Remote Sensing, Vol. 52, 2014, No. 8, pp. 4808–4822, doi: 10.1109/tgrs.2013.2285049.

[41] Wang, H.—Celik, T.: Sparse Representation-Based Hyperspectral Image Classification. Signal, Image and Video Processing, Vol. 12, 2018, No. 5, pp. 1009–1017, doi: 10.1007/s11760-018-1249-1.

[42] Wright, J.—Yang, A. Y.—Ganesh, A.—Sastry, S. S.: Robust Face Recognition via Sparse Representation. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 31, 2009, No. 2, pp. 210–227, doi: 10.1109/tpami.2008.79.

[43] Yang, A. Y.—Sastry, S. S.—Ganesh, A.—Ma, Y.: Fast ℓ_1-Minimization Algorithms and an Application in Robust Face Recognition: A Review. 2010 IEEE International Conference on Image Processing (ICIP), 2010, pp. 1849–1852, doi: 10.1109/ICIP.2010.5651522.

[44] Yang, J.—Peng, Y.—Xu, W.—Dai, Q.: Ways to Sparse Representation: An Overview. Science in China Series F: Information Sciences, Vol. 52, 2009, No. 4, pp. 695–703, doi: 10.1007/s11432-009-0045-5.

[45] Zayyani, H.—Babaie-Zadeh, M.—Jutten, C: An Iterative Bayesian Algorithm for Sparse Component Analysis in Presence of Noise. IEEE Transactions on Signal Processing, Vol. 57, 2009, No. 11, pp. 4378–4390, doi: 10.1109/tsp.2009.2025154.

[46] Zayyani, H.—Korki, M.—Marvasti, F: Dictionary Learning for Blind One Bit Compressed Sensing. IEEE Signal Processing Letters, Vol. 23, 2016, No. 2, pp. 187–191, doi: 10.1109/lsp.2015.2503804.

[47] Zhang, L.—Wei, W.—Tian, C.—Li, F.—Zhang, Y.: Exploring Structured Sparsity by a Reweighted Laplace Prior for Hyperspectral Compressive Sensing. IEEE Transactions on Image Processing, Vol. 25, 2016, No. 10, pp. 4974–4988, doi: 10.1109/tip.2016.2598652.

[48] Zhang, Z.—Xu, Y.—Yang, J.—Li, X.—Zhang, D.: A Survey of Sparse Representation: Algorithms and Applications. IEEE Access, Vol. 3, 2015, pp. 490–530, doi: 10.1109/ACCESS.2015.2430359.

İrem Ülkü received her M.Sc. degree in electrical and electronics engineering from Middle East Technical University in 2013, and Ph.D. degree in electronics and communication engineering from Çankaya University in 2017. Her research interests include hyperspectral image processing, dictionary learning, compressive sensing and sparse coding.

Ersin Kizgut received his Ph.D. degree in mathematics from Middle East Techical University in 2016. He is recently a postdoctoral researcher at Polytechnic University of Valencia. His research interests include applied mathematics, computer science, operator theory, complex analysis, and topological vector spaces.

Computing and Informatics, Vol. 38, 2019, 173–196, doi: 10.31577/cai_2019_1_173

CL-VIS: VISUALIZATION PLATFORM FOR UNDERSTANDING AND CHECKING THE OPENCL PROGRAMS

SeongKi Kim

Division of Computer Science Engineering
Keimyung University, 1095, Dalgubeol-daero, Dalseo-Gu
Daegu, Republic of Korea
e-mail: skkim9226@gmail.com

HyukSoo Han

Department of Computer Science
Sangmyung University, 20, Hongjimun 2-gil, Jongno-Gu
Seoul, Republic of Korea
e-mail: hshan@smu.ac.kr

Abstract. Due to GPU's improved hardware performance, many researchers have tried to utilize the GPU for computer vision, image processing, cryptography, and artificial intelligence. As results, the GPU could successfully speed up algorithms from tens to hundreds of times in many cases. However, GPU programming is still known to be difficult because of its different characteristics from the traditional CPU programming. Also, it is hard to find the root causes of software failures because the failures are irreproducible in many cases. Our goal is to simplify the process of verifying intended actions when debugging GPGPU programs. To achieve this goal, we use the visualization method of executed codes because it can increase the human's understanding through seeing and analyzing the real actions by each thread. We developed a platform that can visualize the running OpenCL codes and algorithms that can identify data race, barrier divergence, and infinite loop in the GPU. To the best of our knowledge, this is the first study on the visualizations of OpenCL operations and detection of infinite loops in the programs. We also suggest an algorithm for detecting data race with GPU-specific lock-step execution and barrier function.

Keywords: Visualization, GPGPU, debug, data race, barrier divergence, infinite loop

1 INTRODUCTION

To meet the user's increasing demands for realistic software and the increasing display resolution (e.g. Full HD (1 920 × 1 080), Ultra HD (3 840 × 2 160)), the sizes of visual data (e.g. Texture, Vertex, Color, Normal) have become larger, and the GPU has improved. Also with the improvements, GPGPU platforms, such as Compute Unified Device Architecture (CUDA) [13] and Open Computing Language (OpenCL) [20], enabled the programming of the GPU for general purposes. Many researchers have tried to use them for performance improvements in different fields. As a result, the GPU could successfully accelerate algorithms for computer vision [17], image processing [7], cryptography [29], and artificial intelligence [19] fields. Furthermore, many researchers have built GPU-based supercomputers that are ranked highly in top 500 websites [22] and used for complex calculations.

Although the GPGPU has been widespread, most of the previous studies concentrated on performance improvement and correctness of results, without much focus on debugging GPGPU programs. Recently, GPGPU has started to become widespread in safety-critical systems, and its software faults can become important issues, shortly. For example, computer vision algorithms, deep learning within a car [14] or medical imaging [21] can exploit the GPGPU, but the failure of these systems can lead to a severe accident.

Regardless of the importance of error-free GPGPU programs, GPGPU is hard to program, and the possibilities of mistakes are high because of the following reasons. First, GPU's characteristics are different from the CPU. Because of it, we cannot simply convert the codes for CPU to the ones for GPU. The simple conversion can lead to a minor performance benefit in many cases, or the conversion can be impossible in many cases because of different characteristics. Second, the results of GPU codes may be different if a different number of threads or thread groups is employed. The developers can alter the number of threads for optimization cases, and the change may cause unexpected results from the inter-thread intervention. Third, the number of GPU threads can be huge (e.g. 10 000 000), and some failures can be irreproducible because their causes are from the thread scheduling or status.

In addition to the programming/debugging difficulties attributed to different characteristics, and thread intervention, data race, barrier divergence, and infinite loop also make the GPGPU programming error-prone. The data race can be a significant problem particularly in the GPU because a vast number of threads may access the same memory at the same time. These simultaneous accesses can cause an unexpected problem depending on the access orders of many threads. The barrier divergence is a specific problem to the GPU and can cause unexpected results in many cases. It happens when all of the GPU threads within a group do not reach the same barrier point. Differently from the traditional CPU case, the infinite loop can happen by GPU-specific lock-step execution.

To minimize these difficulties in GPGPU programming and help in checking the GPGPU programs, some tools such as Nsight systems [24], Allinea DDT [25], Profiler [27] and mem-check [26] have been developed for the CUDA platform. However,

the Nsight does not support the visualization of GPU codes but the visualization of CPU codes only. In addition, Allinea DDT, profiler, and cuda-memcheck are developed only for the CUDA platform. To make up for these limitations, we developed the visualization platform for the OpenCL case. The contribution of this paper is as follows. First, we suggest a visualization method of running OpenCL codes. Second, we propose a heuristic for finding the GPU-specific infinite loop. Third, we suggest a method for finding data races with the GPU-specific lockstep execution and barrier function.

The rest of this paper is organized as follows. Section 2 describes the OpenCL, Oclgrind as well as Data Race/Barrier Divergence/Infinite Loops as backgrounds and related works. Section 3 describes the GPU-specific characteristics and issues in more detail. Section 4 describes our visualization methods and algorithms for finding data races, barrier divergences, and infinite loops at the GPU and their rationales. Section 5 describes the results through our implementations, and the conclusion is drawn in Section 6.

2 BACKGROUND AND RELATED WORKS

In this paper, we used the OpenCL as a GPGPU language because it is an open standard regardless of vendor and platforms. For background knowledge, this section describes OpenCL and Oclgrind. Also for related works, this section introduces the previous works for data race, barrier divergence and infinite loop at the GPU.

2.1 OpenCL

OpenCL is an open standard for heterogeneous computing and allows the programming of the CPU or GPU. Apple Inc. [1] originally developed the OpenCL, and the Khronos Group [15] maintains it presently. The Khronos Group defines the specification [10] of OpenCL so that all vendors should support for compatibility. The specification defines four different models in terms of platform, execution, memory, and programming aspects.

Platform Model: The platform model describes a *host* and a *device*. The host is a central unit that executes the main program, divides works to each device, and collects the results from each device. A device is a target unit that runs the parallel parts of a program. A device can have many compute units, and each compute unit can have many processing elements. Figure 1 illustrates these relations.

In Figure 1, a host creates a context to compute device 1 and enqueues commands to the device through the established context. The device schedules the commands, delivers them to the compute units and the processing elements according to its policy, and each processing element runs the commands. The CPU or GPU can be one of the devices.

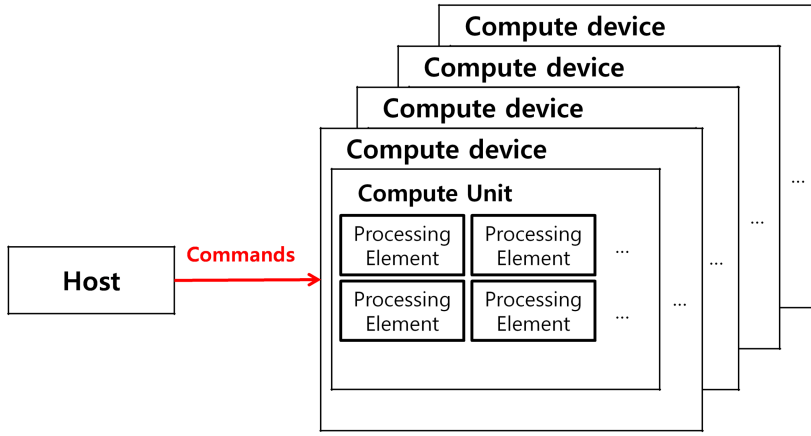


Figure 1. OpenCL platform model

Execution Model: The execution model defines a *host program* and a *kernel*. The host program is a program that runs on the host. A kernel is a function executed on the compute device. When the host program sends the commands (kernel execution, memory read, memory write), it also sends the number of work-groups and work-items. Work-group is a group of threads that run the same operation and runs on a compute unit. Work-item is a thread and executes on a processing element. Figure 2 illustrates the decompositions of work-groups and work-items handled by OpenCL.

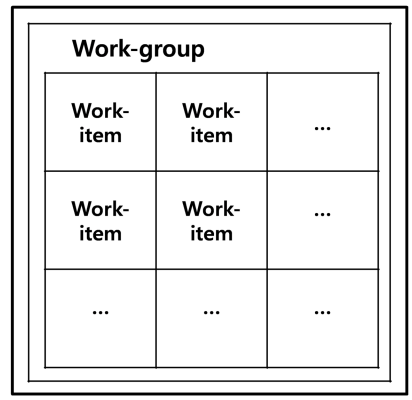


Figure 2. The decomposition of work-groups and work-items

Figure 2 shows the relationship among work-groups and work-items in two dimensions. Work-groups are independent of each other and simultaneously run, and work-items within the same work-group execute the same operation in the

lock-step method. A compute unit runs a work-group, and a processing element runs a work-item. If the number of work-groups and work-items are more than those of the physical compute units or processing elements, the mapping of work-group or work-item is scheduled according to the vendor's policy. Work-items within a single work-group can communicate through a shared buffer, and any accesses to the buffer should be synchronized.

2.2 Oclgrind

Oclgrind [18] is an open-source simulator based on the OpenCL's Standard Portable Intermediate Representation (SPIR) [16]. It enables to create a development tool for the OpenCL program and implements the OpenCL specification on the CPU. As inputs, Oclgrind receives an OpenCL application or a kernel and runs it on the CPU. Oclgrind can check barrier divergences and data races with additional options. It also enables to debug the OpenCL application or kernel interactively and exports some plugin interfaces so that any other applications can be developed.

Oclgrind can check the data race, but it can have some false positives because it only checks whether the other work-items access the same memory address or not and does not check the lock-step executions, which is described in detail in Section 3.

2.3 Data Race, Barrier Divergence, Infinite Loop

The data race happens when two or more threads access the same memory location and one of the accesses is a write operation. If many threads access the same memory, then the results are dependent on the threads' read/write orders and are unpredictable in some cases. This scenario has been a large problem to the CPU-based multithreaded system; therefore, many researchers have tried to detect it automatically with minimum false positives. FastTrack gave some idea to improve the slow but precise vector-clock race detector [8]. Relay was a static race detector based on the locksets that could be scaled to millions of lines of C code [28], and Pacer was low-overhead sampling-based data race detector based on the FastTrack [6]. However, most of these approaches are inapplicable to the GPU because the GPU programs only support the barrier function for synchronization [32].

The problem of a data race can be worse in the GPU case because thousands of work-items (threads) can run simultaneously the same instruction, and simultaneously make accesses to the same memory. When these simultaneous accesses occur, the different execution orders of the memory load or store can cause different results or hard-to-reproduce errors depending on the situation. To detect the data race at the GPU, many researchers have suggested static, dynamic, hybrid algorithms or special hardware. GPUVerify is a static verifier [5], which translates a kernel into a sequential Boogie [2] program, and proves the correctness of the sequential program. LDetector is a static detector and uses a two-pass approach to detect write-write/read-write races [11]. LD statically detects a race with a two-pass detection algorithm that is atomic free and a memory-adaptive solution [12]. Oclgrind

is a dynamic simulator, which can identify the race through monitoring the memory accesses [18]. GRace [32] and GMRace [31] are hybrid mechanisms that combine the static and the dynamic ways, reduces the number of statements, and monitors only the survived memory accesses. HAccRG [9] and Hydra [30] designed special kinds of hardware for this problem. However, these implementations have their limitations for real use. If we want to use the GPUVerify, LD, LDetector, GRace or GMRace, the compilers should be modified. If we want to use hardware-based approaches such as HAccRG and Hydra, special hardware should be added. Furthermore, GPUVerify and LDetector do not support the atomic operations. Oclgrind just includes a simple access monitor and does not consider the GPU characteristics that all work-items within the same work-group run the same instruction. Section 3 will describe this limitation through an example in more detail.

Barrier divergence is one of the specific problems of the GPU, and can also cause unexpected results. The results can be different from an architecture to an architecture or a vendor to a vendor because the OpenCL specification does not clarify the results. The OpenCL 2.1 Reference Pages [3] and the OpenCL 2.2 Reference Guide [4] only mentions that "All work-items in a work-group executing the kernel on a processor must execute this function before any are allowed to continue execution beyond the work_group_barrier." "Work-items in a work-group must execute this before any can continue." and does not mention anything about the results. Therefore, an unexpected result can happen according to the implementation. GPUVerify and Oclgrind can detect this problem.

An infinite loop can occur at both the CPU and the GPU, but its causes can be different from each other. One of the GPU-specific reasons of infinite loop is that all work-items within the same work-group run the same operation in the GPU case. Section 3 will also describe the barrier divergence and the infinite loop through an example. To the best of our knowledge, no tools or algorithms can detect this problem at this time.

3 SPECIFIC CHARACTERISTICS AND ISSUES OF OPENCL PROGRAMMING

OpenCL programming is similar to the case of CPU programming in many aspects and uses a subset of C or C++ language. However, it has different characteristics and issues that this section describes.

3.1 GPU-Specific Programming Model

The GPU threads within a group run code simultaneously in lock-step method, which highlights its difference from the CPU threads. Given that the goal of a GPU is to maximize the parallelism, it internally has thousands of processing elements. Given this characteristic, it is hard for each processing element to run the entirely separate parts of codes. Therefore, all work-items within the same work-group execute the same instruction.

If a kernel includes a conditional statement, such as *if*, then the condition can be evaluated as *true* at some work-items, but *false* at the other work-items. For this different branch case within a single work-group, the GPU uses the predicated executions. A processing element executes the predicated instructions only if the condition is *true*. Otherwise, the processing element discards the instruction or does not commit the result according to the GPU's internal architecture. Figure 3 illustrates the example of a conditional statement, and Figure 4 shows the generated instructions with a predicated form after compiling Figure 3.

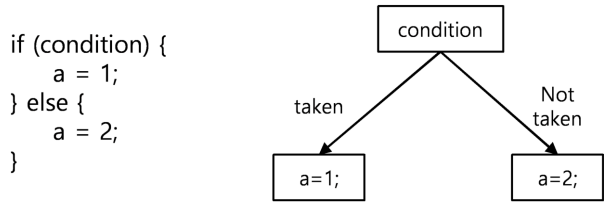


Figure 3. An example of a conditional statement

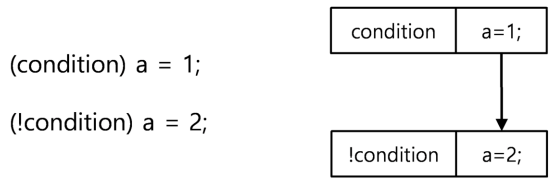


Figure 4. An example of predicated execution

Although the exact instructions can be different from architecture to architecture or vendor to vendor, Figure 4 illustrates the fundamental idea of predicated executions. In Figure 4, if the condition is *true*, the processing element runs $a = 1$. Otherwise, the processing element runs $a = 2$. With these predicated executions, all work-items within the same work-group can execute the same instruction at the same time. The loops, such as *for*, *do*, and *while*, also use these predicated executions. In this case, some processing elements can finish the loop earlier than other elements, but they cannot exit the loop and are still in the loop while ignoring the results of executed statements through this predicated executions.

3.2 Data Race

Listing 1 shows an example of a data race in the GPU case. If we run Listing 1 after setting all values in the array, g, to 1, and the number of work-items and workgroups to 128 and 8, respectively, then the work-item 15 runs the *int gid = get_global_id(0)* and *gid* becomes 15 because the *get_global_id(0)* function returns the number of work-item. The work-item 15 runs the $g[15] = g[16] + g[17]$, and

the work-item 16 runs the $g[16] = g[17] + g[18]$. Because the work-items 15 and 16 are respectively included in work-groups 0 and 1, and the memory coherency is not guaranteed between work-groups, $g[15]$ can be two if $g[16]$ and $g[17]$ are one. $g[15]$ can also be three if $g[16]$ is one and $g[17]$ is two. $g[15]$ can also be four. The results are unpredictable depending on the GPU's internal implementation, the number of work-items, and the work-groups or the memory policy. Listing 2 illustrates the example that can resolve the data race in the Listing 1.

```
1 __kernel void data_race(__global int *g)
2 {
3     int gid = get_global_id(0);
4     g[gid] = g[gid + 1] + g[gid + 2];
5 }
```

Listing 1. An example of data race

```
1  __kernel void no_data_race_1(__global int * g)
2  {
3      int gid = get_global_id(0);
4
5      int temp1 = g[gid + 1];
6      int temp2 = g[gid + 2];
7
8      barrier(CLK_GLOBAL_MEM_FENCE);
9
10     g[gid] = temp1 + temp2;
11 }
```

Listing 2. An example of resolved data race

If a work-item meets the barrier function in Listing 2, the work-item waits until the other work-items within the same work-group also reach the barrier function. Therefore, work-item 15 stores the values of $g[16]$ and $g[17]$ into the private variables $temp1$ and $temp2$, respectively. The work-item 16 has the values of $g[17]$ and $g[18]$ in the variables $temp1$ and $temp2$. The variables $temp1$ and $temp2$ are private to each work-item. After the barrier function, the work-item 15 runs $g[15] = temp1 + temp2$, and the result will be two without any data race. The work-item 16 also runs $g[16] = temp1 + temp2$, and the result will be also two.

Data race can happen or not when a GPGPU kernel is executed with the different numbers of work-groups and work-items. Listing 3 shows an example.

If we run Listing 3 with four work-items and one work-group, all of the four work-items are included in the same work-group. Therefore, all of the four work-items perform the same instruction in the lock-step method, and the data race cannot

```
1 __kernel void no_data_race_2(__global int * g)
2 {
3     int gid = get_global_id(0);
4
5     int temp1 = g[gid + 1];
6     int temp2 = g[gid + 2];
7
8     g[gid] = temp1 + temp2;
9 }
```

Listing 3. An example of the data race according to the number of work-groups and work-items

happen because the private variables of each work-item, *temp*1 and *temp*2, are used to store the values of $g[gid+1]$ and $g[gid+2]$, differently from the Listing 1. However, if we run it with four work-items and four work-groups, all of the four work-items runs separately. Therefore, the data race can occur according to the scheduling, and we should avoid the four work-groups or use the barrier function. Oclgrind does not consider this GPU-specific lock-step execution and report the data race even in one work-group case.

3.3 Barrier Divergence

Listing 4 illustrates the barrier divergence at the GPU. If we run Listing 4 with 128 work-items and 8 work-groups, work-item 0 evaluates the condition as *true* at line 5. Therefore, the $g[0]$ will be zero by line 7, and work-item 0 executes the first barrier function. Work-item 1 evaluates the condition as *false*; therefore, it performs the second barrier function at line 12. If a work-item meets a barrier function, the work-item waits until the other work-items also reach the location; therefore, work-items 0 and 1 wait for each other at different places. In this case, the results are unpredictable according to the implementation.

3.4 Infinite Loop

Listing 5 illustrates the infinite loop caused by the lock-step execution. When we run the kernel *infloop* in Listing 5 with 128 work-items and 8 work-groups after setting g to zero, work-item 0 calls the *lock* function at line 11. In the *lock* function, the *atom_xchg* function changes the global variable g into one, and returns the old value, zero, of g. Therefore, the local variable o will be zero. Then, work-item 0 can exit the loop. Meanwhile, work-item 1 also calls the *lock* function, and the *atom_xchg* function changes the global variable, g, into one again, and returns the old value, one, of g because it was already modified by work-item 0. Therefore, the local variable o will be one. Then, work-item 1 continues the loop again because it

```
1  __kernel void barrier_divergence(__global int *g)
2  {
3      int gid = get_global_id(0);
4
5      if (gid % 2 == 0)
6      {
7          g[gid] = gid;
8          barrier(CLK_GLOBAL_MEM_FENCE);
9      }
10     else
11     {
12         barrier(CLK_GLOBAL_MEM_FENCE);
13         g[gid] = gid + g[gid - 1];
14     }
15 }
```

Listing 4. An example of the barrier divergence

does not satisfy the exit condition $o \leq 0$. Besides work-item 0, all work-items in the workgroup continue the loop. Work-item 0 satisfies the exit condition, but it cannot also exit the loop because it should run the same instruction in the lock-step similar to other work-items. As a result, all work-items will loop forever.

```
1  void lock(__global int* g)
2  {
3      int o;
4
5      do {
6          o = atom_xchg(g, 1);
7      } while (o > 0);
8  }
9
10 __kernel void infloop(__global int* g)
11 {
12     lock(g);
13 }
```

Listing 5. An example of infinite loop

We checked that Listing 5 caused the infinite loop, and the system sometimes halts at Intel's OpenCL implementation (HD Graphics 530, Driver version 20.19.15.4 531) and NVIDIA's implementation (GTX 960M, Driver version 378.78). This problem is quite common because this kind of codes, such as Listing 5, works well in the CPU case as the locking codes. In the CPU case, the thread 0 (work-item) does not

have to loop again; therefore, the CPU does not meet the infinite loop. However, the GPU is frozen by the lock-step execution.

4 CL-VIS: VISUALIZATION PLATFORM

This section describes our platforms for the visualization of running GPGPU codes (CL-Vis) and algorithms to find the issues described in Section 3.

4.1 Overview

When programming the GPU, it is hard to understand what each work-item is doing. Without knowing what went on, it is hard to fix the problems or the bugs within the software. If a developer can see what the work-item does, he/she can easily understand why the problem happens in many cases and can fix the problems. To help understand the internal operations of each work-item, we developed a visualization platform based on Oclgrind. To implement it, we used the architecture in Figure 5.

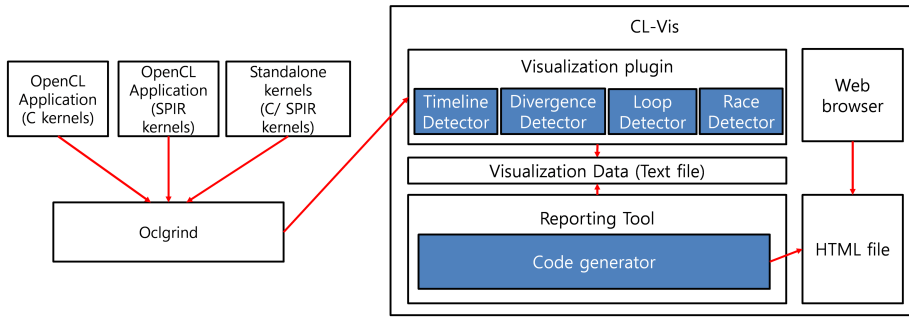


Figure 5. The overall architecture of CL-Vis

Our visualization platform, CL-Vis, largely consists of the following components: a visualization plugin and a reporting tool. As an input, Oclgrind receives the information file that includes the running kernel/file, the number of work-items and work-groups, and the parameter values, and runs the kernel. During the execution, it calls our visualization plugin that implements the callback functions. Oclgrind supports the following basic callback functions in Table 2 so that any user can develop additional tools based on Oclgrind. Besides the essential functions in Table 2, we extended Oclgrind so that we can obtain more information and added more functions in Table 3.

Through these callback functions in Tables 2 and 3, the visualization plugin records the information on the executed instructions and their times to thread-specific files. The timeline of each operation is detected through Algorithm 1.

Through Algorithm 1, the visualization plugin records the global id (the thread id), the time difference between the current time and the kernel's starting time, the C-based statement, the assembly-based instructions to a buffer, and writes the buffer to a file that includes the execution information of a work-item. It also detects the infinite loop, the barrier, and the data race through the algorithms described in more detail in Subsection 4.2. The reporting tool uses the file of each work-item for the visualization. The reporting tool receives the generated files by the visualization plugin as inputs and generates an HTML file as outputs. After these procedures, the HTML file can be viewed through a web browser. To summarize, our platform can visualize the codes after the execution completes. Figure 6 presents the result of our visualization.

ALGORITHM 1: Timeline detector

```
1: for each work-item w do
2:     for each start and end of an instruction do
3:         Record the w's id, time difference between the current time and the w's starting
           time, the statement and the executed instruction to a buffer
4:     end for
5:     Write the recorded buffer to a file
6: end for
```

In Figure 6, the horizontal axis shows the time, and the vertical axis lists the number of work-item and workgroup. Each blue rectangle represents an operation that a work-item performed, and the text in the box is the executed action at the time. If we scroll down the page, we can see other threads. We can also see the other operations if we move the page to the left or right. We can also zoom in/out the box through a mouse wheel. In many cases, a single line of the C-based statement consists of many lines of assembly-based statements, and many boxes can include the same text. Therefore, our platform generates the C-based statement first, and the assembly-based statement within a parenthesis next to avoid confusion.

Figure 6 shows that a single work-group includes 16 work-items, and the work-items run independently. We can also see the executed action if we move the mouse cursor on the box or horizontally increase the box. Through this visualization, we can check that the program operates as intended. For example, we can see that the work-item 0 runs the *true* case, and the other work-items run the *false* case at the *if* statement. We can also check the number of loops that the work-items run. Besides the checking, this visualization can also be used for the education purpose, and we can use this platform to describe the concepts of GPGPU programming.

4.2 Detecting Algorithms

This subsection describes the algorithms used to detect the data race, the barrier divergence, and the infinite loop included in the CL-Vis. We designed them as

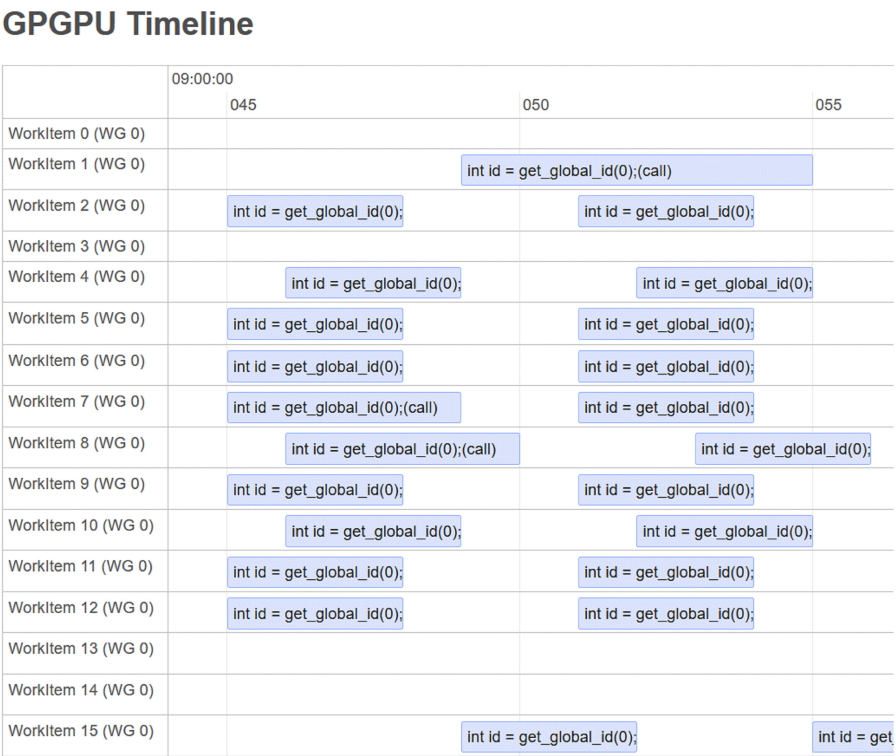


Figure 6. Visualization of the GPGPU program

a plugin of Oclgrind, and it is an event-based program. Therefore, we described each algorithm using events.

4.2.1 Race Detector

The visualization plugin includes the race detector that detects a data race through Algorithms 2 and 3. Algorithm 2 monitors all memory accesses of work-item w and reports the data race if the below conditions 1, 2, 3 are evaluated as *true*, and condition 4 or 5 is evaluated as *true*. Algorithm 3 changes the states of memory accesses into inactive if a work-item meets a barrier.

1. The first condition is that the address space of access is *global* or *local*. Work-items share only the global and local memories so that these types can have a data race.
2. The second condition is that the access to memory is active. If a work-item meets a barrier, all of the previous memory accesses are guaranteed to be committed. Therefore, the memory accesses become inactive in this case.

3. The third condition is that another thread accesses the same memory, and one of the access is the write. The data race can happen only in this case.
4. The fourth condition is that the accessing statement is the same in the case that the current work-item and the work-item of the previously access are in the same group. The work-items in the same work-group run an operation in lock-step; therefore, the data race will happen only if the accessing statements are the same.
5. The fifth condition is that the memory type is global if the work-items are in different groups. The work-items in different groups can be simultaneously executed based on the GPU's internal policy; therefore, the data race can happen.

ALGORITHM 2: Race detector

```
1: for each work-item w do
2:   for each memory access m do
3:     if address space of m is global or local then
4:       for each access entry e ∈ access table do
5:         if e is active, another thread also accesses m's address, and one of the
           accesses is the write operation then
6:           if w and the work-item of e are in the same group then
7:             if the accessing statement is the same then
8:               Mark the access m as a data race
9:             end if
10:          else
11:            if address space of m is global then
12:              Mark the access m as a data race
13:            end if
14:          end if
15:        end if
16:      end for
17:    end if
18:    Record m's address, size, statement, memory type (global/local), access type
       (read/write) and group number into the access table
19:  end for
20: end for
```

ALGORITHM 3: Access clearing

```
1: for each work-item w do
2:   if w meets a barrier then
3:     Change the all access states of w in the access table to inactive
4:   end if
5: end for
```

Most of the previous dynamic algorithms for detecting the data race do not include conditions 4 and 5. They just check conditions 1, 2 and 3, so have more false positives. To decrease the false positives, we added condition 4 utilizing that the work-items in a work-group run the code in lock-step. We also added condition 5 because the data race can happen in the global memory case even if the accessing statements are different.

4.2.2 Divergence Detector

The visualization plugin includes the divergence detector that detects a barrier divergence through Algorithms 4 and 5. Whenever a work-item meets a barrier function, Algorithm 4 records the global id (the thread id), the finishing time and statement. We record the statement because a kernel can have many barrier functions and barrier divergence, and we need to identify the barrier function. We also record the finishing time because our visualization platform should find the most recently cleared barrier function. Then, when Oclgrind clears the barrier function, Algorithm 4 marks the latest time among the met work-items as a barrier time because Oclgrind is a simulator based on the CPU, which sequentially executes a kernel differently from the GPU. Algorithm 5 uses the most recent time because all work-items should wait for the last finished work-item.

ALGORITHM 4: Barrier recording at the divergence detector

```
1: for each work-item w do
2:     if w meets a barrier then
3:         Record the w's id, the finishing time and the statement
4:     end if
5: end for
```

ALGORITHM 5: Barrier clearing at the divergence detector

```
1: for each barrier b do
2:     Mark the last finished work-item's time among the met work-items as a barrier
       time
3: end for
```

Algorithms 4 and 5 can find all of the barrier divergences without a false positive because Algorithm 4 records all of the work-items that meet a barrier, and Algorithm 5 marks to all of the met work-items when a barrier is cleared.

4.2.3 Loop Detector

The visualization plugin includes the loop detector that detects a possible infinite loop through Algorithm 6. In Algorithm 6, whenever a work-item w meets a loop l, our visualization plugin records the used variables if they are global and modified

within the loop. Our plugin also records the affected variable by a global variable. Then, our plugin checks that the changed or affected variables are used at the exit condition of the loop.

ALGORITHM 6: Loop detector

```
1: for each work-item w do
2:   for each loop l do
3:     Record the used variables if they are global and modified within the loop l
4:     Record the affected variables by a global variable if they are private or local,
       and modified within the loop l
5:     if the global or affected variables are used as an exit condition then
6:       Mark the exit condition as an infinite loop
7:     end if
8:   end for
9: end for
```

If a work-item changes the global variable and uses the variable at the exit condition, then other work-items can also be affected because the exit condition can also be modified. The count of a loop can be shorter, longer or even infinite. In Listing 5, the global variable, g, is modified within the loop and affects the private variable o. The other work-items are also affected by the modification of g, and the exit condition is also affected by the change of o. The infinite loop at the GPU happens in this case; therefore, our plugin detects the changes of an exit condition.

Algorithm 6 may have a false positive if a global variable is modified or a local/private variable is modified by a global variable within a loop, and the variable is used as an exit condition. But, we have never met such case in many kernels until now.

4.3 Detailed Implementation

To verify our architecture and algorithms, we implement the visualization plugin and the reporting tool. Besides the function additions in Table 3, we also disabled the optimization of a GPGPU kernel code because we should obtain the statement information in our algorithms.

We implemented the algorithms in Section 4 at our visualization plugin and the reporting tool in the Linux environment (Ubuntu 14.04), and verified the results using Firefox 52.0. To visualize the results, we used a timeline within the *vis.js* library [23] that supports powerful display functions for the web browser. To determine whether two statements are the same or not in Algorithm 2, we used their line numbers as statement information.

5 RESULTS OF DETECTING ALGORITHMS

This section presents the results of suggested architecture and algorithms.

5.1 Data Race

We run Listing 1 with 128 work-items and 8 work-groups. Figure 7 presents the results. In Figure 7, our Algorithm 2 detects the data race at the $g[gid] = g[gid + 1] + g[gid + 2]$ statement in Listing 1, and marks them with yellow color (Red box). Therefore, any researcher or developer can easily notice it. We also run Listing 2 with 128 work-items and 8 work-groups. Our Algorithms 2 and 3 did not detect the race because the barrier function cleared all of the previous memory accesses. However, Oclgrind reports some data races.

Figure 7. The result of detecting data race at Listing 1. Algorithm 2 detects the data race at the $g[gid] = g[gid + 1] + g[gid + 2]$ statement in Listing 1, and marks them with yellow color (red box).

Figure 8 illustrates the results of no race and barrier.

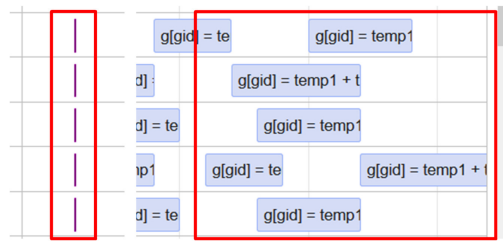


Figure 8. The result of detecting data race at Listing 2. Algorithms 2 and 3 did not detect the race at the statement that Oclgrind reports some data races but they cannot happen in the real scenario (red box in the right side).

In Figure 8, our CL-Vis platform does not report any data races with yellow color (the right red box) because the barrier function cleared all of the previous memory accesses and correctly marks the barrier function with magenta color (the left red box).

We also run Listing 3 with one work-group and four work-groups. Our Algorithms 2 and 3 can correctly detect the race from the lock-step execution only in the four work-group, as shown in Figures 7 and 8.

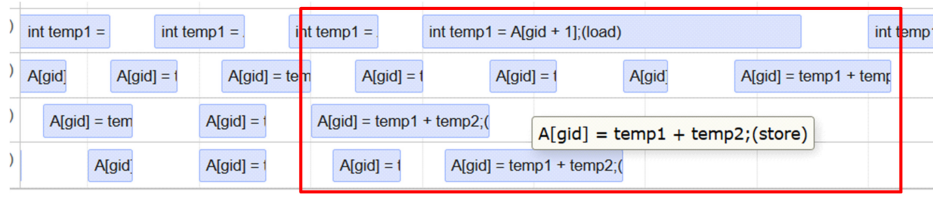


Figure 9. The result of detecting data race with one work-group at Listing 3. Given that the accessing statements are at different locations, Algorithm 2 does not report it as a data race (red box).

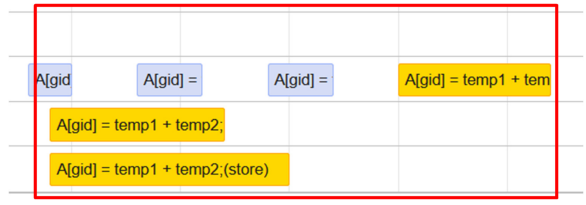


Figure 10. The result of detecting data race with four work-groups at Listing 3. The work-item can run the separate instruction in the different work-group case; therefore, our algorithm reports the data race (red box).

In the one work-group case, such as Figure 9, all work-items in the same work-group run the same instruction, and our Algorithm 2 checks that the accessing statement is the same. Given that the accessing statements are at different locations, our Algorithm 2 does not report it as a data race. However, in the different work-group case, such as Figure 10, the work-item can run the separate instruction; therefore, our algorithm reports the data race (red box).

5.2 Barrier Divergence

We run Listing 4 with four work-items and four work-groups, detect the barrier divergence through Algorithms 4 and 5, and mark it with magenta as shown in Figure 11 (red box). In Figure 11, the bar with magenta color shows that work-items 0 and 2 run the first barrier in Listing 4, but work-items 1 and 3 execute the second barrier in Listing 4. If all of the work-items meet the same barrier function, then only one bar with a magenta color exists. Therefore, any researcher or developer can see that the barrier divergence happened.

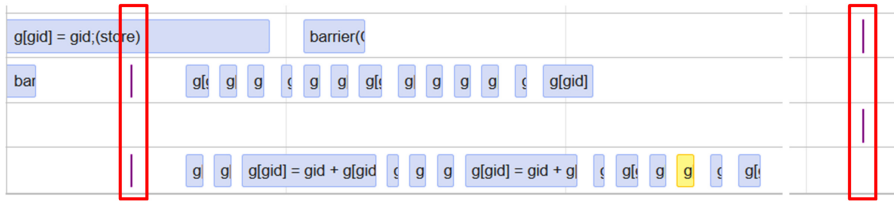


Figure 11. The result of detecting barrier divergence at Listing 4. Algorithms 4 and 5 mark the barrier divergence with magenta in the case of four work-items and four work-groups.

5.3 Infinite Loop

We run Listing 5 with 128 work-items and 8 work-groups as shown in Figure 12. In Figure 12, our Algorithm 6 detects the infinite loop and marks it with red color (red box). It records the global or the affected variables within a loop, then checks that the exit condition uses the variables. Through this algorithm, our platform detects the infinite loop, and shows it at the statement $while(o > 0)$ in Listing 5.

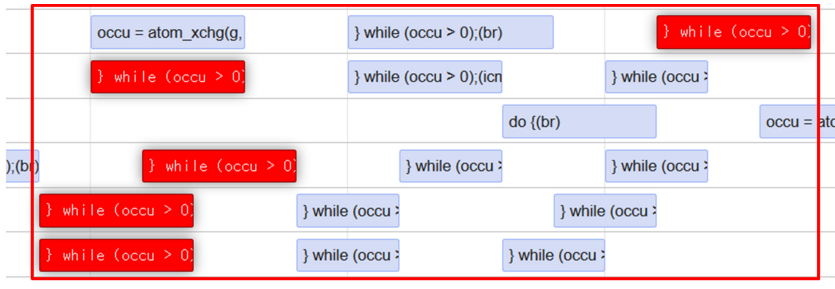


Figure 12. The result of detecting infinite loop at Listing 5. Algorithm 6 records the global or the affected variables within a loop, and checks that the exit condition uses the variables. It also marks the infinite loop with red color.

5.4 Others

Besides these primary results through the kernels in this paper, we also verify that our architecture and algorithms can successfully visualize 15 kernels within Oclgrind. Table 1 summarizes the tested kernels and their sizes of global/local works. Among the kernels in Table 1, Oclgrind wrongly reports the data races in the *global_only_fence*, *intragroup_hidden_race*, *local_read_write_race* cases because it does not check the lock-step execution at the GPU. Furthermore, Oclgrind reports the data races in the *local_only_fence* because it does not clear the memory access

before the barrier function. However, our algorithm correctly finds no data race in all of those cases.

Group	File	Global Size	Local Size
Barrier	barrier_different_instructions	4, 1, 1	4, 1, 1
Divergence	barrier_divergence	4, 1, 1	4, 1, 1
Data Race	broadcast	4, 1, 1	1, 1, 1
	global_fence	16, 1, 1	4, 1, 1
	global_only_fence	4, 1, 1	4, 1, 1
	global_read_write_race	4, 1, 1	4, 1, 1
	global_write_write_race	4, 1, 1	1, 1, 1
	increment	4, 1, 1	1, 1, 1
	intergroup_hidden_race	2, 1, 1	1, 1, 1
	intergroup_race	8, 1, 1	4, 1, 1
	intragroup_hidden_race	2, 1, 1	2, 1, 1
	local_only_fence	16, 1, 1	4, 1, 1
	local_read_write_race	4, 1, 1	4, 1, 1
	local_write_write_race	4, 1, 1	4, 1, 1
	uniform_write_race	4, 1, 1	4, 1, 1

Table 1. Tested kernels

6 CONCLUSIONS

GPU programming is known to be difficult due to several reasons: difficulty in understanding the GPU characteristics, which are different from the CPU, and few of debugging tools compared to the CPU. To reduce these problems and help in checking the GPGPU programs, we developed the CL-Vis based on Oclgrind. We also suggest the algorithms for automatic detection of data race with lock-step execution and barrier function, barrier divergence, and infinite loop. These algorithms are included in the CL-Vis and verified through various kernels. To the best of our knowledge, our paper is the first research that visualizes the executed functions of GPU and suggests an algorithm for detecting infinite loops automatically. Furthermore, our paper suggests the algorithm for detecting data races at the GPU-specific executions.

However, our CL-Vis and algorithms can be improved more in the following aspects: our plugin is based on Oclgrind, which is an OpenCL simulator run on the CPU. However, the real working environment with the GPU can be different. For example, the actual execution time can be different. If the GPU vendor develops its simulator that is more similar to its internal operations, then we can visualize the running codes based on it and see the instructions more correctly. Furthermore, our race detection algorithm is not aware of the barrier function with a local memory option. If a work-item meets a barrier function, only the local memory accesses within the same group should be cleared. The current implementation makes all

of the global/local memory accesses inactive. Also, our visualization platform has a limitation in the number of threads due to the restricted memory. Finally, if we can see the result of each action by a work-item, it can also be helpful to check the OpenCL program. We have plans to improve these aspects and find other ways to relieve the burden of GPU users.

Acknowledgements

This research was supported by the MSIT (Ministry of Science, ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-0-0445) supervised by the IITP(Institute for Information and communications Technology Promotion). SeongKi Kim was supported by NRF in Korea (NRF-2017R1A1A1A05069806).

REFERENCES

[1] Apple Coporation. Availaible at: `https://www.apple.com/`.

[2] Barnett, M.—Chang, B.-Y. E.—DeLine, R.—Jacobs, B.—Leino, K. R. M.: Boogie: A Modular Reusable Verifier for Object-Oriented Programs. In: de Boer, F. S., Bonsangue, M. M., Graf, S., de Roever, W. P. (Eds.): Formal Methods for Components and Objects (FMCO 2005). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4111, 2006, pp. 364–387, doi: 10.1007/11804192_17.

[3] OpenCL 2.1 Reference Pages. Availaible at: `https://www.khronos.org/registry/OpenCL/sdk/2.1/docs/man/xhtml/`.

[4] OpenCL 2.2 Reference Guide. Availaible at: `https://www.khronos.org/files/opencl22-reference-guide.pdf`.

[5] Betts, A.—Chong, N.—Donaldson, A.—Qadeer, S.—Thomson, P.: GPUVerify: A Verifier for GPU Kernels. ACM SIGPLAN Notices – OOPSLA '12, Vol. 47, 2012, No. 10, pp. 113–132, doi: 10.1145/2398857.2384625.

[6] Bond, M. D.—Coons, K. E.—McKinley, K. S.: PACER: Proportional Detection of Data Races. ACM SIGPLAN Notices – PLDI '10, Vol. 45, 2010, No. 6, pp. 255–268, doi: 10.1145/1809028.1806626.

[7] Fernando, R.: GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics. Pearson Higher Education, 2004.

[8] Flanagan, C.—Freund, S. N.: FastTrack: Efficient and Precise Dynamic Race Detection. ACM SIGPLAN Notices – PLDI '09, Vol. 44, 2009, No. 6, pp. 121–133, doi: 10.1145/1543135.1542490.

[9] Holey, A.—Mekkat, V.—Zhai, A.: HAccRG:: Hardware-Accelerated Data Race Detection in GPUs. Proceedings of the 2013 42nd International Conference on Parallel Processing (ICPP '13), 2013, pp. 60–69, doi: 10.1109/icpp.2013.15.

[10] Howes, L. (Ed.): The OpenCL Specification Version: 2.1. Document Revision: 24, 2018. `https://www.khronos.org/registry/OpenCL/specs/opencl-2.1.pdf`.

[11] Li, P.—Ding, C.—Hu, X.—Soyata, T.: LDetector: A Low Overhead Race Detector for GPU Programs. 5^{th} Workshop on Determinism and Correctness in Parallel Programming (WoDET 2014), Salt Lake City, UT, March 2014.

[12] Li, P.—Hu, X.—Chen, D.—Brock, J.—Luo, H.—Zhang, E. Z.—Ding, C.: LD: Low-Overhead GPU Race Detection Without Access Monitoring. ACM Transactions on Architecture and Code Optimization, Vol. 14, 2017, No. 1, Art. No. 9, 25 pp., doi: 10.1145/3046678.

[13] Nickolls, J.—Buck, I.—Garland, M.—Skadron, K.: Scalable Parallel Programming with CUDA. Queue – GPU Computing, Vol. 6, 2008, No. 2, pp. 40–53, doi: 10.1145/1365490.1365500.

[14] NVIDIA Corporation. Driving Innovation. Availaible at: `http://www.nvidia.com/object/drive-automotive-technology.html`.

[15] Khronos Group. Availaible at: `https://www.khronos.org/`.

[16] Ouril, B. (Ed.): The SPIR Specification Version 2.0 – Provision. 2014.

[17] Pharr, M.—Fernando, R.: GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (GPU Gems). Addison-Wesley Professional, 2005.

[18] Price, J.—McIntosh-Smith, S.: Oclgrind: An Extensible OpenCL Device Simulator. Proceedings of the 3^{rd} International Workshop on OpenCL (IWOCL '15), New York, NY, USA, 2015, Art. No. 12, doi: 10.1145/2791321.2791333.

[19] Silver, D.—Huang, A.—Maddison, C. J.—Guez, A.—Sifre, L.—van den Driessche, G.—Schrittwieser, J.—Antonoglou, I.—Panneershelvam, V.—Lanctot, M.—Dieleman, S.—Grewe, D.—Nham, J.—Kalchbrenner, N.—Sutskever, I.—Lillicrap, T.—Leach, M.—Kavukcuoglu, K.—Graepel, T.—Hassabis, D.: Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature, Vol. 529, 2016, No. 7587, pp. 484–489, doi: 10.1038/nature16961.

[20] Stone, J. E.—Gohara, D.—Shi, G.: OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. Computing in Science and Engineering, Vol. 12, 2010, No. 3, pp. 66–73, doi: 10.1109/MCSE.2010.69.

[21] Stone, S. S.—Haldar, J. P.—Tsao, S. C.—Hwu, W.-M. W.—Sutton, B. P.—Liang, Z.-P.: Accelerating Advanced MRI Reconstructions on GPUs. Journal of Parallel and Distributed Computing, Vol. 68, 2008, No. 10, pp. 1307–1318, doi: 10.1016/j.jpdc.2008.05.013.

[22] TOP 500 The List. Availaible at: `https://www.top500.org/`.

[23] vis.js. Availaible at: `http://http://visjs.org/`.

[24] NVIDIA Nsight Visual Studio Edition. `https://developer.nvidia.com/nsight-visual-studio-edition`.

[25] Allinea DDT. `https://developer.nvidia.com/allinea-ddt`.

[26] CUDA-MEMCHECK. `https://docs.nvidia.com/cuda/cuda-memcheck/index.html`.

[27] Profiler. `https://docs.nvidia.com/cuda/profiler-users-guide/index.html`.

[28] Voung, J. W.—Jhala, R.—Lerner, S.: RELAY: Static Race Detection on Millions of Lines of Code. Proceedings of the the 6^{th} Joint Meeting of the Euro-

pean Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE '07), New York, NY, USA, 2007, pp. 205–214, doi: 10.1145/1287624.1287654.

[29] Yang, J.—Goodman, J.: Symmetric Key Cryptography on Modern Graphics Hardware. In: Kurosawa, K. (Ed.): Advances in Cryptology – ASIACRYPT 2007. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4833, 2007, pp. 249–264, doi: 10.1007/978-3-540-76900-2_15.

[30] Zhang, W.—Yu, S.—Wang, H.—Dai, Z.—Chen, H.: Hardware Support for Concurrent Detection of Multiple Concurrency Bugs on Fused CPU-GPU Architectures. IEEE Transactions on Computers, Vol. 65, 2016, No. 10, pp. 3083–3095, doi: 10.1109/TC.2015.2512860.

[31] Zheng, M.—Ravi, V. T.—Qin, F.—Agrawal, G.: GMRace: Detecting Data Races in GPU Programs via a Low-Overhead Scheme. IEEE Transactions on Parallel and Distributed Systems, Vol. 25, 2014, No. 1, pp. 104–115, doi: 10.1109/TPDS.2013.44.

[32] Zheng, M.—Ravi, V. T.—Qin, F.—Agrawal, G.: GRace: A Low-Overhead Mechanism for Detecting Data Races in GPU Programs. ACM SIGPLAN Notices – PPoPP '11, Vol. 46, 2011, No. 8, pp. 135–146, doi: 10.1145/2038037.1941574.

SeongKi Kim is Assistant Professor at Keimyung University. He received his Ph.D. degree in computer science and engineering from Seoul National University in 2009. He researched and developed software for the GPU, the GPGPU and dynamic voltage and frequency scaling (DVFS) at the Samsung Electronics from 2009 to 2014. He also worked at the Ewha Womans University and SangMyung University from 2014 to 2017. His current research interests include the areas of graphics/game algorithms, an algorithm optimization through the GPU, and high-performance computing with CPU and GPU.

HyukSoo Han is Professor of computer science at SangMyung University. He received his M.Sc. degree in computer science and statistics from Seoul National University in 1987 and his Ph.D. degree in computer engineering from South Florida University in 1992. His current research interests include software process, software quality, software safety, software education and human computer interaction.

Appendices

Name	Description
hostMemoryLoad	Called when a host memory is loaded
hostMemoryStore	Called when a host memory is stored
instructionExecuted	Called when an instruction is executed
kernelBegin	Called when a kernel starts
kernelEnd	Called when a kernel ends
log	Called when a log message is outputted
memoryAllocated	Called when a memory is allocated
memoryAtomicLoad	Called when an atomic memory is loaded
memoryAtomicStore	Called when an atomic memory is stored
memoryDeallocated	Called when a memory is deallocated
memoryLoad	Called when a memory is loaded
memoryMap	Called when a memory is mapped
memoryStore	Called when a memory is stored
memoryUnmap	Called when a memory is unmapped
workGroupBarrier	Called when a barrier is cleared
workGroupBegin	Called when a work-group begins
workGroupComplete	Called when a work-group completes
workItemBegin	Called when a work-item begins
workItemComplete	Called when a work-item completes

Table 2. Callback functions

Name	Description
instructionBeforeExecuted	Called before an instruction is executed
workGroupBarrierBeforeClear	Called before a barrier is cleared
workItemBarrier	Called when a work-item clears a barrier

Table 3. Added Callback functions

Computing and Informatics, Vol. 38, 2019, 197–222, doi: 10.31577/cai_2019_1_197

ASSESSMENT OF TWO TASK FRAMEWORKS WITH DEPENDENCIES FOR MATRIX FACTORIZATIONS ON A MULTICORE ARCHITECTURE

Jarosław Bylina

Marie Curie-Sklodowska University
Institute of Mathematics
Pl. M. Curie-Sklodowskiej 5
20-031 Lublin, Poland
e-mail: jaroslaw.bylina@umcs.pl

Abstract. In this study, we evaluate two task frameworks with dependencies for important application kernels coming from the numerical linear algebra. In this approach, the algorithms of the matrix factorization are considered, namely the tiled LU and the WZ factorizations both without pivoting. In tiled algorithms, the operations are represented as a sequence of small tasks which operate on square blocks (tiles) of the data. The dependencies among tasks are expressed as a direct acyclic graph and the runtime system runs the graph on a multicore architecture. The performance of applications based on the task dependencies is related to efficient compilers and the runtime systems. We report the performance and the scalability of two task frameworks with dependencies on the multicore architecture for the matrix factorizations. Namely, we compare OpenMP and Intel Thread Building Blocks. Our results show that the number of tiles in both factorizations always have an impact on the performance and the speedup. Both the frameworks show their suitability for efficient parallelization of such applications, although both have their own merits and flaws.

Keywords: Task parallelism, task dependencies, parallel programming model, runtime system, OpenMP, Intel TBB

Mathematics Subject Classification 2010: 65Y05

1 INTRODUCTION

In recent years task-based parallel programming paradigms became an alternative to classical thread-based paradigms on the shared memory multicore architectures. Such task-based implementations of parallel applications are suitable for multicore architectures. The use of tasks causes higher concurrency and scalability of the implementations.

However, the task parallelism requires appropriate compilers and execution systems to perform efficiently. Such systems must decide about load-balancing, overheads, and task scheduling. It is difficult to evaluate the efficiency of such run-time systems because, for various applications, various criteria will be important. Contemporary architectures which employ shared-memory parallelism produce appearance of a lot of frameworks exploiting them efficiently – such as OpenMP [6], Intel Threading Building Blocks (TBB for short) [14], Cilk Plus [24], OpenCL [25] and others. Thus, choosing a proper framework for a specific problem is not easy.

Different techniques are provided by task-based programming frameworks to the programmer for writing programs. In some approaches, the algorithms are represented as graphs of tasks and the runtime system runs the graph on the target architecture. In the graphs, the nodes are computational tasks performed in kernel subroutines and edges represent the dependencies among them. In particular, in the application connected to the numerical linear algebra, direct acyclic graphs (DAGs or dags) are utilized. A dag is a finite directed graph without cycles. A dag contains a finite number of vertices and edges. Each edge is directed from one vertex to another.

Matrix factorization algorithms are dense linear algebra algorithms used often in many scientific applications. This paper addresses two matrix factorizations. In addition to the well-known LU factorization, we test another form of factorization, namely the WZ factorization. The WZ factorization was introduced in [8, 19]. It was a novel method for solving linear systems in parallel. They both have $O(n^3)$ time complexity using $O(n^2)$ data space. The tiled LU and WZ factorization algorithms use the standard set of Basic Linear Algebra Subprograms (BLAS) [7] and a block array layout for better cache performance. Moreover, the computations on array tiles (square blocks) fit the task-based parallel model well. The tiled LU and WZ algorithms can be represented as a dag where nodes are the executed BLAS routines.

The first contribution of this paper is providing details of implementations of the tiled LU and WZ factorizations in OpenMP and TBB; the second contribution is an analysis of the experimental results of these factorization implementations for two frameworks that support task parallelism with dependencies. In this article, we investigate two task frameworks, namely OpenMP and Intel TBB. We chose these frameworks because they are different in the way of implementation development. OpenMP is an extension to the C/C++ languages and TBB is a C++ library. These frameworks have also been chosen because they are quite popular, work for different architectures and CPUs, they can perform differently on different hardware

architectures and they differ in their approaches to tasks. To compare OpenMP and TBB we study the tiled LU factorization without pivoting and the WZ factorization (also without pivoting).

The rest of this paper is organized as follows: Section 2 shows some related works. Section 3 presents the tiled LU factorization without pivoting and tiled WZ factorization without pivoting and shows dags (direct acyclic graphs) for each algorithm. Section 4 describes the details of parallel implementations of the tiled LU and tiled WZ algorithms on multicore, shared-memory machines. One of them relies on the use of the OpenMP `task` directive with the `depend` clause. The second one uses TBB. Section 5 is devoted to the results of numerical experiments carried out on shared memory multicore architectures and to the comparisons of the two task-based frameworks, namely OpenMP standard and TBB. Section 6 shows the conclusions of our research and presents future plans.

2 RELATED WORKS

In this paper, we evaluate two task frameworks with dependencies on the multicore architecture. Similarly, the issue of the comparison of the task parallel frameworks in the multicore environments is considered in the works [16, 20, 21, 22].

In the work [16], the tasks without dependencies are considered. The authors compare OpenMP 3.0 runtimes on unbalanced task graphs against Cilk and Intel TBB. The conclusion of these studies is the fact that the OpenMP task management mechanisms are less optimized than those of the other threading approaches, namely Cilk and Intel TBB.

The evaluation of OpenMP 4.0 tasks with dependencies with the benchmark called KASTORS consisting of small kernels ported to the OpenMP dependent task model is described in the paper [22]. KASTORS uses the OpenMP 4.0 task dependency constructs to extend different applications. One with these kernels is the LU decomposition from the PLASMA library (Parallel Linear Algebra for Scalable Multicore Architectures) framework [1, 13]. The performance of OpenMP applications expressing task dependencies is closely related to how efficiently compilers and runtime systems implement this new feature. The FLAME (Formal Linear Algebra Method) project [11, 15, 18] is another set of high performance libraries. Moreover, it is not only software but rather a formal approach to creating correct, fast and efficient linear algebra algorithms and their implementations.

The author of the work [21] evaluates Intel's C++ Concurrent Collections (CnC) and Threading Building Blocks (TBB) libraries for application coming from numerical linear algebra, namely tiled Gauss–Jordan algorithm. The conclusion of these studies is the fact that CnC is almost as fast as TBB.

The paper [20] aims to evaluate OpenMP, TBB and other ways of parallelization and optimization of computational problems that need task parallelism as well as data parallelism. The examples used there are adaptive Simpson's integration and Belman-Ford algorithm.

3 MATRIX DECOMPOSITION

The matrix decomposition is a factorization of a matrix into a product of matrices. We assume that the decomposed matrix is nonsingular, square, and diagonally dominant (thus, we can use factorization without pivoting). In this section, we describe two tiled matrix decompositions, namely the well-known tiled LU decomposition without pivoting and the tiled WZ decomposition (also without pivoting). Each of the algorithms is expressed in terms of the elementary operations and the graphs.

3.1 Block LU Factorization

Let the dense square $(n \times n)$ diagonally dominant matrix $\mathbf{A}$ be partitioned into $q \times q$ tiles of size $t \times t$ ($n = qt$ and $1 \leq t \leq n$) and $\mathbf{A}_{ij}$ is a square tile on row i and column j. The tiled LU factorization algorithm performs the majority of its floating-point operations (flop) using the level 3 BLAS operations.

The tiled algorithm for the LU factorization may base on the following set of elementary operations.

- `DTRSM(u/nonu, up/lo, l/r, A, X, B)`. This BLAS subroutine is used to compute $\mathbf{X} = \mathbf{A}^{-1} \cdot \mathbf{B}$ (denoted by `l`), or $\mathbf{X} = \mathbf{B} \cdot \mathbf{A}^{-1}$ (denoted by `r`), where $\mathbf{X}$ and $\mathbf{B}$ are $s \times s$ matrices, $\mathbf{A}$ is a unit (`u`) or non-unit (`nonu`), upper (`up`) or lower (`lo`) triangular matrix.
- `DGEMM(A, B, C)`. This BLAS subroutine is used to compute $\mathbf{A} = -\mathbf{B} \cdot \mathbf{C} + \mathbf{A}$, where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are $s \times s$ matrices.

Algorithm 1 presents the tiled LU factorization algorithm expressed in terms of elementary operations. The circled numbers shown in Algorithm 1 emphasize the correspondence between the operations and the tasks in Figure 1.

The scheduler executes tasks in any order that respects the dependencies shown in the dag. This approach is presented in [3] for tiled linear algebra algorithms. Figure 1 presents a directed acyclic graph for parallel tile LU factorization of a 4×4 tile matrix. Arrows show dependencies between tasks. The tasks are denoted by circles. The red circles (with the number 1) represent line 2 in Algorithm 1; the magenta circles (with the number 2) – line 4; the green circles (with the number 3) – line 7 and the blue ones (with the number 4) correspond to line 11.

3.2 Block WZ Factorization

The WZ factorization is described in [8, 19, 23]. Let us assume that $\mathbf{A}$ is a square, nonsingular and diagonally dominant matrix of the size $n \times n$ (we consider only even n, for simplicity's sake).

We are to find matrices $\mathbf{W}$ and $\mathbf{Z}$ that fulfill $\mathbf{WZ} = \mathbf{A}$. The main diagonal of the matrix $\mathbf{W}$ consists only of ones. The second diagonal consists of zeros. These

Algorithm 1 Tiled LU factorization

Require: $\mathbf{A}$, q
Ensure: $\mathbf{L}$, $\mathbf{U}$
1: **for** $k \leftarrow 1, q$ **do**
2: `LU`$(\mathbf{A}_{kk}, \mathbf{L}_{kk}, \mathbf{U}_{kk})$ ①
3: **for** $i \leftarrow k+1, q$ **do**
4: `DTRSM(nonu, up, q,` $\mathbf{U}_{kk}$, $\mathbf{L}_{ik}$, $\mathbf{A}_{ik}$`)` ②
5: **end for**
6: **for** $j \leftarrow k+1, q$ **do**
7: `DTRSM(u, lo, l,` $\mathbf{L}_{kk}$, $\mathbf{U}_{kj}$, $\mathbf{A}_{kj}$`)` ③
8: **end for**
9: **for** $i \leftarrow k+1, q$ **do**
10: **for** $j \leftarrow k+1, q$ **do**
11: `DGEMM(`$\mathbf{A}_{ij}$, $\mathbf{L}_{ik}$, $\mathbf{U}_{kj}$`)` ④
12: **end for**
13: **end for**
14: **end for**

diagonals divide the matrix into four triangles. The left and right triangles contain non-zeros, and the top and bottom ones contain only zeros. The matrix $\mathbf{Z}$ has non-zeros where the matrix $\mathbf{W}$ has zeros or ones – and vice versa. The first part of the WZ factorization algorithm consists of setting successive parts of columns of the matrix $\mathbf{A}$ to zeros. In the first step, we do that with the elements in the 1^{st} and n^{th} columns – from the 2^{nd} row to the $(n-1)^{\text{th}}$ row. Next, we update the inner submatrix of $\mathbf{A}$ of the size $(n-2) \times (n-2)$ and for $k = 2, \ldots, \frac{n}{2}$ we zero elements in the k^{th} and $(n-k+1)^{\text{st}}$ columns – from the $(k+1)^{\text{st}}$ row to the $(n-k)^{\text{th}}$ row and we update the inner submatrix.

The tiled WZ factorization algorithm [5] performs the majority of its floating-point operations (flop) using the level 3 BLAS operations. We assume that $\mathbf{A}$ is a square nonsingular matrix of an even size n and it is partitioned on $r \times r$ (r is also even) parts (r of each side – rows and columns). The tiled WZ algorithm consists of four repeating stages $r/2$ times. Stage 1 (line 3 in Algorithm 2) comprises the WZ factorization of a matrix built from four corner blocks of the input matrix. Stage 2 (lines 4–11 in Algorithm 2) computes $2s$ (where $s = \frac{n}{r}$) columns of the matrix $\mathbf{W}$ – s right columns and s left columns. Stage 3 (lines 12–19 in Algorithm 2) computes $2s$ rows of the matrix $\mathbf{Z}$ – s bottom rows and s top rows. Stage 4 (lines 20–25 in Algorithm 2) updates the inner submatrix of $\mathbf{A}$ – that is, $\mathbf{A}$ without outer $2s$ columns and $2s$ rows. In the next step, the algorithm is repeated for this inner matrix. The tiled algorithm for the WZ factorization will be based on the following set of elementary operations.

- `DTRSM(u/nonu, up/lo, l/r, A, X, B)`. This BLAS subroutine is used to compute $\mathbf{X} = \mathbf{A}^{-1} \cdot \mathbf{B}$ (denoted by `l`), or $\mathbf{X} = \mathbf{B} \cdot \mathbf{A}^{-1}$ (denoted by `r`), where $\mathbf{X}$

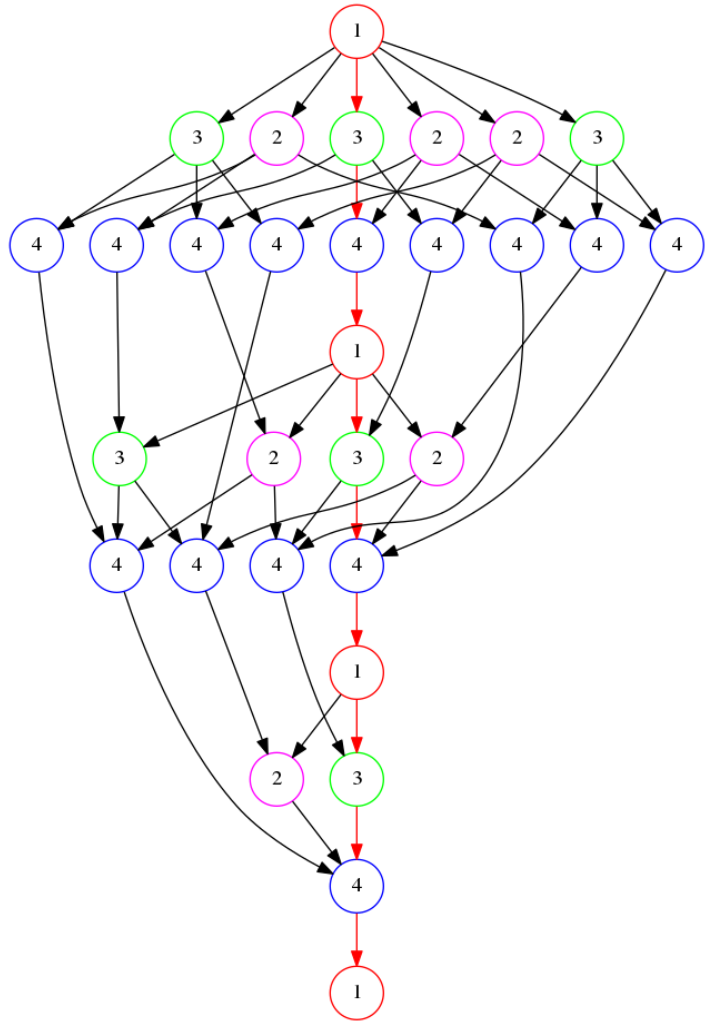


Figure 1. A directed acyclic graph for the tile LU factorization of a 4×4 tiled matrix

and $\mathbf{B}$ are $s \times s$ matrices, $\mathbf{A}$ is a unit (`u`) or non-unit (`nonu`), upper (`up`) or lower (`lo`) triangular matrix.

- `DGEMM(A, B, C)`. This BLAS subroutine is used to compute $\mathbf{A} = -\mathbf{B} \cdot \mathbf{C} + \mathbf{A}$, where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are $s \times s$ matrices.
- `DGEMM_copy(A, B, C, D)`. This BLAS subroutine is used to compute $\mathbf{A} = -\mathbf{B} \cdot \mathbf{C} + \mathbf{D}$, where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are $s \times s$ matrices.

Algorithm 2 presents the tiled WZ factorization algorithm expressed with the above-mentioned operations (`DTRSM`, `DGEMM`, `DGEMM_copy`) for a nonsingular matrix $\mathbf{A}$ partitioned into $r \times r$ blocks. The matrices $\mathbf{W}$ and $\mathbf{Z}$ are the results of this algorithm. Again, the circled numbers in Algorithm 2 show which operations belong to respective tasks in Figure 2.

Algorithm 2 Tiled WZ factorization

Require: $\mathbf{A}$, r
Ensure: $\mathbf{W}$, $\mathbf{Z}$

1: **for** $k \leftarrow 1, r/2$ **do**
2: $k_2 \leftarrow r - k + 1$
3: WZ($\begin{bmatrix} \mathbf{A}_{kk} & \mathbf{A}_{kk_2} \\ \mathbf{A}_{k_2k} & \mathbf{A}_{k_2k_2} \end{bmatrix}, \begin{bmatrix} \mathbf{W}_{kk} & \mathbf{W}_{kk_2} \\ \mathbf{W}_{k_2k} & \mathbf{W}_{k_2k_2} \end{bmatrix}, \begin{bmatrix} \mathbf{Z}_{kk} & \mathbf{Z}_{kk_2} \\ \mathbf{Z}_{k_2k} & \mathbf{Z}_{k_2k_2} \end{bmatrix}$) ①
4: DTRSM(nonu, up, l, $\mathbf{Z}_{kk}$, $\mathbf{D_1}$, $\mathbf{Z}_{kk_2}$) ①
5: DGEMM_copy($\mathbf{E_1}$, $\mathbf{Z}_{k_2k}$, $\mathbf{D_1}$, $\mathbf{Z}_{k_2k_2}$) ①
6: DTRSM(u, lo, r, $\mathbf{W}_{kk}$, $\mathbf{D_2}$, $\mathbf{W}_{k_2k}$) ①
7: DGEMM_copy($\mathbf{E_2}$, $\mathbf{D_2}$, $\mathbf{W}_{kk_2}$, $\mathbf{W}_{k_2k_2}$) ①
8: **for** $i \leftarrow k+1, k_2-1$ **do**
9: DGEMM($\mathbf{A}_{ik_2}$, $\mathbf{A}_{ik}$, $\mathbf{D_1}$) ②
10: DTRSM(nonu, lo, r, $\mathbf{E_1}$, $\mathbf{W}_{ik_2}$, $\mathbf{A}_{ik_2}$) ②
11: DGEMM($\mathbf{A}_{ik}$, $\mathbf{W}_{ik_2}$, $\mathbf{Z}_{k_2k}$) ②
12: DTRSM(nonu, up, r, $\mathbf{Z}_{kk}$, $\mathbf{W}_{ik}$, $\mathbf{A}_{ik}$) ②
13: **end for**
14: **for** $i \leftarrow k+1, k_2-1$ **do**
15: DGEMM($\mathbf{A}_{k_2i}$, $\mathbf{D_2}$, $\mathbf{A}_{ki}$) ③
16: DTRSM(u, up, l, $\mathbf{E_2}$, $\mathbf{Z}_{k_2i}$, $\mathbf{A}_{k_2i}$) ③
17: DGEMM($\mathbf{A}_{ki}$, $\mathbf{W}_{kk_2}$, $\mathbf{Z}_{k_2i}$) ③
18: DTRSM(u, lo, l, $\mathbf{W}_{kk}$, $\mathbf{Z}_{ki}$, $\mathbf{A}_{ki}$) ③
19: **end for**
20: **for** $j \leftarrow k+1, k_2-1$ **do**
21: **for** $i \leftarrow k+1, k_2-1$ **do**
22: DGEMM($\mathbf{A}_{ij}$, $\mathbf{W}_{ik}$, $\mathbf{Z}_{kj}$) ④
23: DGEMM($\mathbf{A}_{ij}$, $\mathbf{W}_{ik_2}$, $\mathbf{Z}_{k_2j}$) ④
24: **end for**
25: **end for**
26: **end for**

Algorithm 2 can be represented as a dag. Figure 2 shows such a dag for the tiled WZ factorization when Algorithm 2 is executed for a 4×4 tiled matrix. This figure also corresponds to the lines in Algorithm 2. The tasks are denoted by circles and the red circles (with the number 1) represent lines 3–7; the magenta circles (with the number 2) – lines 9–12; the green circle (with the number 3) – lines 15–18 and the blue ones (with the number 4) correspond to lines 22–23.

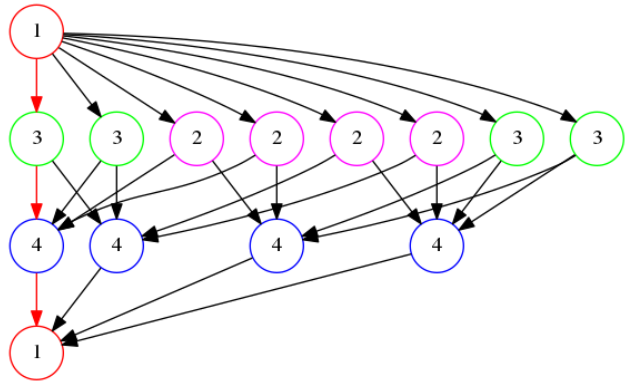


Figure 2. A dag for the tiled WZ factorization of a 4×4 tiled matrix

3.3 Theoretical Speedup – Amdahl's Law

We compute a maximal theoretical speedup of our algorithms from Amdahl's law, using the cost of the sequential traditional versions, that is:

$$C_{LU}(n) = \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n,$$
$$C_{WZ}(n) = \frac{2}{3}n^3 - \frac{7}{3}n - 3.$$

Let us compute the cost of Algorithm 2, namely $C_{WZ}(n, s)$ (as it depends not only on n but also on the size s of the tile). To achieve this, we are to compute costs of the particular stages 1–4 which we denote C_{WZ1}, C_{WZ2}, C_{WZ3} and C_{WZ4}, respectively ($n = r \cdot s$):

$$C_{WZ1}(s) = C_{WZ}(2s) = \frac{16}{3}s^3 - \frac{7}{3}s - 3,$$
$$C_{WZ2}(k, r, s) = C_{WZ3}(k, r, s) = 3s^3 + s^2 + \sum_{i=k+1}^{r-k} (6s^3 + 2s^2)$$
$$= 3s^3 + s^2 + (6s^3 + 2s^2)(r - 2k),$$
$$C_{WZ4}(k, r, s) = \sum_{i=k+1}^{r-k} \sum_{j=k+1}^{r-k} (4s^3 + 2s^2) = (4s^3 + 2s^2)(r - 2k)^2.$$

Thus, the number of floating-point arithmetic operations for the tiled WZ factorization algorithm (Algorithm 2) is:

$$\begin{aligned} C_{WZ}(n,s) &= \sum_{k=1}^{\frac{r}{2}} (C_{WZ1}(s) + 2C_{WZ2}(k,r,s) + C_{WZ4}(k,r,s)) \\ &= \frac{n^3(4s+2) + 6n^2s^2 + n(6s^3 - 2s^2 - 7s - 9)}{6s}. \end{aligned}$$

Analogously, we can obtain the cost of the tiled LU factorization (here, $n = q \cdot t$ and the size of the block is t; we present only formulas necessary for further considerations), namely:

$$\begin{aligned} C_{LU1}(t) &= \frac{2}{3}t^3 - \frac{1}{2}t^2 - \frac{1}{6}t, \\ C_{LU}(n,t) &= \sum_{k=1}^{q} (C_{LU1}(t) + 2C_{LU2}(k,q,t) + C_{LU4}(k,q,t)) \\ &= \frac{n^3(4t+2) - 3n^2t - n(2t^2+t)}{6t}. \end{aligned}$$

The maximal theoretical speedup for p threads can be estimated from Amdahl's law. To use this law we must determine which part must be executed sequentially, and which part can be executed in parallel. In our algorithms, the only parts which have to be executed sequentially are the first stages (denoted with ①).

Thus, let P_{WZseq} be the relative cost of this sequential part of Algorithm 2. The cost of one execution of stage 1 is C_{WZ1}, but it is executed $\frac{r}{2}$ times. So:

$$P_{WZseq}(n,s) = \frac{\frac{r}{2} \cdot C_{WZ1}(s)}{C_{WZ}(n,s)} = \frac{16s^3 - 7s - 9}{n^2(4s+2) + 6ns^2 + 6s^3 - 2s^2 - 7s - 9}.$$

According to Amdahl's law [10], the best theoretical speedup for the parallel tiled WZ factorization algorithm (for p threads, $n \times n$ matrix and $s \times s$ tile) is:

$$S_{WZ}(p;n,s) = \frac{1}{P_{WZseq}(n,s) + \frac{1 - P_{WZseq}(n,s)}{p}}.$$

Analogously, we can obtain similar formulas for the tiled LU factorization (here, $n = q \cdot t$ and the size of the block is t; we present only formulas necessary for further

considerations), namely:

$$
\begin{aligned}
C_{LU1}(t) &= \frac{2}{3}t^3 - \frac{1}{2}t^2 - \frac{1}{6}t, \\
C_{LU}(n,t) &= \sum_{k=1}^{q}(C_{LU1}(t) + 2C_{LU2}(k,q,t) + C_{LU4}(k,q,t) \\
&= \frac{n^3(4t+2) - 3n^2t - n(2t^2+t)}{6t}, \\
P_{LUseq}(n,s) &= \frac{q \cdot C_{LU1}(t)}{C_{LU}(n,t)} = \frac{4t^3 - 3t^2 - t}{n^2(4t+2) - 3nt - 2t^2 - t}, \\
S_{LU}(p;n,t) &= \frac{1}{P_{LUseq}(n,t) + \frac{1-P_{LUseq}(n,t)}{p}}.
\end{aligned}
$$

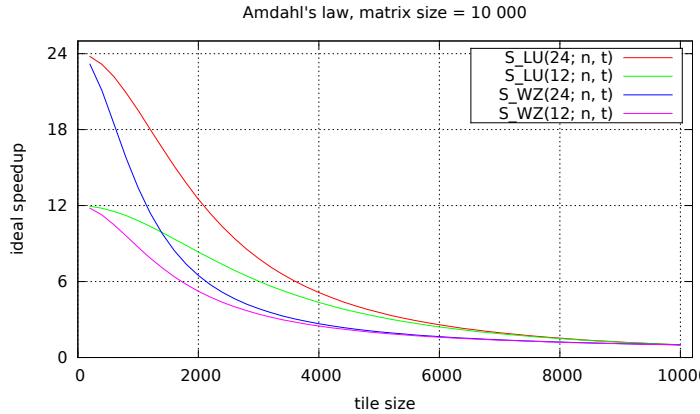


Figure 3. The theoretical maximum speedup for the tiled LU and WZ implementations

Figure 3 shows the theroetical maximum speedup as a function of the size of the block, for fixed $n = 1\,000$ and for selected numbers p of threads ($p \in \{12, 24\}$). We can see from Figure 3 that the speedup should be the best for as small blocks as possible. However, smaller blocks require more communication and synchronization – which is not counted in Amdahl's law. So the size of the block has to be chosen experimentally.

4 IMPLEMENTATIONS

In our work, the matrices are stored as one-dimensional arrays of the tiles and we refer to it as a tiled layout, similarly to [3]. In the tile layout, the matrices are

represented as small square tiles of data contiguous in memory so that each core can operate on an individual tile independently.

4.1 OpenMP

In our first implementations, we employ the OpenMP task directive and the BLAS routines for matrices' operations. We call these implementations TLU(q)-OpenMP and TWZ(r)-OpenMP. The well-known OpenMP standard was extended with the task construct introduced in version 3.0 [16] with support for task dependencies by means of the `depend` clause. The clause allows defining lists of data items that are only inputs, only outputs, or both inputs and outputs. The annotated task will be scheduled for execution only when the dependencies expressed by those data items are satisfied with respect to preceding tasks in the same task region. This task is bound to a thread from the current team of threads. The execution of the new task can be instant or delayed according to the task schedule and availability of threads. The OpenMP runtime provides a dynamic scheduler of the tasks while avoiding data hazards by keeping track of dependencies. The dynamic scheduler means that the tasks are queueing and executed as quickly as possible. Algorithms 3 and 4 present the tiled factorization with the `#pragma omp task` with dependencies (and the color circles representing the content of the particular tasks).

Algorithm 3 Tiled LU factorization – task-based with dependencies

Require: $\mathbf{A}$, q
Ensure: $\mathbf{L}$, $\mathbf{U}$

for $k \leftarrow 1, q-1$ **do**
  `#pragma omp task depend(in:`A_{kk}`) depend(out:` L_{kk}`) depend(out:` U_{kk}`)`
  `LU(`$\mathbf{A}_{kk}, \mathbf{L}_{kk}, \mathbf{U}_{kk}$`)` ①
  for $i \leftarrow k+1, q$ **do**
    `#pragma omp task depend(in:`A_{ik}`) depend(in:` U_{kk}`) depend(out:` L_{ik}`)`
    `DTRSM(nonu, up, q,` $\mathbf{U}_{kk}$`,` $\mathbf{L}_{ik}$`,` $\mathbf{A}_{ik}$`)` ②
  end for
  for $j \leftarrow k+1, q$ **do**
    `#pragma omp task depend(in:`A_{kj}`) depend(in:` L_{kk}`) depend(out:` U_{kj}`)`
    `DTRSM(u, lo, l,` $\mathbf{L}_{kk}$`,` $\mathbf{U}_{kj}$`,` $\mathbf{A}_{kj}$`)` ③
  end for
  for $i \leftarrow k+1, q$ **do**
    for $j \leftarrow k+1, q$ **do**
      `#pragma omp task depend(in:`L_{ik}`) depend(in:` U_{kj}`) depend(inout:` A_{ij}`)`
      `DGEMM(`$\mathbf{A}_{ij}$`,` $\mathbf{L}_{ik}$`,` $\mathbf{U}_{kj}$`)` ④
    end for
  end for
end for

Algorithm 4 Tiled WZ factorization – task-based with dependencies

Require: $\mathbf{A}$, r
Ensure: $\mathbf{W}$, $\mathbf{Z}$

1: **for** $k \leftarrow 1, r/2$ **do**
2: $k_2 \leftarrow r - k + 1$
3: `#pragma omp task depend(in:` A_{kk}, A_{kk_2}, $A_{k_2k_2}$, A_{k_2k}`)`
`depend(out:` W_{kk}, W_{kk_2}, $W_{k_2k_2}$, W_{k_2k}, Z_{kk}, Z_{kk_2}, $Z_{k_2k_2}$, Z_{k_2k}, D_1, E_1, D_2, E_2 `)`
4: `WZ(`$\begin{bmatrix} \mathbf{A}_{kk} & \mathbf{A}_{kk_2} \\ \mathbf{A}_{k_2k} & \mathbf{A}_{k_2k_2} \end{bmatrix}, \begin{bmatrix} \mathbf{W}_{kk} & \mathbf{W}_{kk_2} \\ \mathbf{W}_{k_2k} & \mathbf{W}_{k_2k_2} \end{bmatrix}, \begin{bmatrix} \mathbf{Z}_{kk} & \mathbf{Z}_{kk_2} \\ \mathbf{Z}_{k_2k} & \mathbf{Z}_{k_2k_2} \end{bmatrix}$`)` (1)
5: `DTRSM(nonu, up, l,` $\mathbf{Z}_{kk}$, $\mathbf{D_1}$, $\mathbf{Z}_{kk_2}$`)` (1)
6: `DGEMM_copy(`$\mathbf{E_1}$, $\mathbf{Z}_{k_2k}$, $\mathbf{D_1}$, $\mathbf{Z}_{k_2k_2}$`)` (1)
7: `DTRSM(u, lo, r,` $\mathbf{W}_{kk}$, $\mathbf{D_2}$, $\mathbf{W}_{k_2k}$`)` (1)
8: `DGEMM_copy(`$\mathbf{E_2}$, $\mathbf{D_2}$, $\mathbf{W}_{kk_2}$, $\mathbf{W}_{k_2k_2}$`)` (1)
9: **for** $i \leftarrow k+1, k_2-1$ **do**
10: `#pragma omp task depend(in:` A_{ik}, A_{ik_2}, Z_{kk}, Z_{k_2k}, D_1, E_1`)`
`depend(out:` W_{ik}, W_{ik_2}`)`
11: `DGEMM(`$\mathbf{A}_{ik_2}$, $\mathbf{A}_{ik}$, $\mathbf{D_1}$`)` (2)
12: `DTRSM(nonu, lo, r,` $\mathbf{E_1}$, $\mathbf{W}_{ik_2}$, $\mathbf{A}_{ik_2}$`)` (2)
13: `DGEMM(`$\mathbf{A}_{ik}$, $\mathbf{W}_{ik_2}$, $\mathbf{Z}_{k_2k}$`)` (2)
14: `DTRSM(nonu, up, r,` $\mathbf{Z}_{kk}$, $\mathbf{W}_{ik}$, $\mathbf{A}_{ik}$`)` (2)
15: **end for**
16: **for** $i \leftarrow k+1, k_2-1$ **do**
17: `#pragma omp task depend(in:` A_{k_2i}, A_{ki}, W_{kk_2}, W_{kk}, D_2, E_2`)`
`depend(out:` Z_{ki}, Z_{k_2i}`)`
18: `DGEMM(`$\mathbf{A}_{k_2i}$, $\mathbf{D_2}$, $\mathbf{A}_{ki}$`)` (3)
19: `DTRSM(u, up, l,` $\mathbf{E_2}$, $\mathbf{Z}_{k_2i}$, $\mathbf{A}_{k_2i}$`)` (3)
20: `DGEMM(`$\mathbf{A}_{ki}$, $\mathbf{W}_{kk_2}$, $\mathbf{Z}_{k_2i}$`)` (3)
21: `DTRSM(u, lo, l,` $\mathbf{W}_{kk}$, $\mathbf{Z}_{ki}$, $\mathbf{A}_{ki}$`)` (3)
22: **end for**
23: **for** $j \leftarrow k+1, k_2-1$ **do**
24: **for** $i \leftarrow k+1, k_2-1$ **do**
25: `#pragma omp task depend(in:` W_{ik}, W_{ik_2}, Z_{kj}, Z_{k_2j}`) depend(out:` A_{ij}`)`
26: `DGEMM(`$\mathbf{A}_{ij}$, $\mathbf{W}_{ik}$, $\mathbf{Z}_{kj}$`)` (4)
27: `DGEMM(`$\mathbf{A}_{ij}$, $\mathbf{W}_{ik_2}$, $\mathbf{Z}_{k_2j}$`)` (4)
28: **end for**
29: **end for**
30: **end for**

It would seem that keeping some matrices in cache between tasks would be profitable. However, forcing it is not possible. Moreover, even if it would be possible, it is not desirable – we use tasks in our implementations and the data are in the cache during one task execution (if the block size is not too big), but locking them between tasks would restrict the task scheduler. The task scheduler itself has to decide, which tasks are to be run on which processors, considering the cache content and the dependencies.

4.2 TBB

Our second set of implementations uses Intel Thread Building Blocks (TBB) and similarly, as in our previous implementations, they call the BLAS routines for matrix operations. We denote these implementations TLU(q)-TBB and TWZ(r)-TBB. The Intel Threading Building Blocks (Intel TBB) [12, 17] is a C++ template library for parallel programming on multicore architectures. This library provides parallel constructs like algorithms, containers, and tasks which the programmer can use to implement an algorithm and run it in parallel.

The TBB task interface requires the declaration of a new class extending the task class and the creation of task object instances. A member function executes the work of the task. However, there are also other tools to run a task-based algorithm – and one of them is the flow graph.

The greatest advantage of this approach is the separation of concerns. We can do the following implementation jobs independently:

- describe the algorithm;
- design and implement small independent computational kernels;
- connect them with the graph to schedule them efficiently.

The use of the flow graph (which can be an arbitrary directed graph, not only a dag) in TBB is different from the OpenMP. Here, the programmer has to build a dependency graph on his own – quoting the dependencies is not enough. For building the graph, there are (among others) following elements (all from the `tbb::flow` namespace):

- the class `graph` – this is the main class which provides the graph implementation;
- the class template `continue_node` – this is an auxiliary class which represents a single node of the graph – and a task at the same time. The main job of the node is storing a functor which describes actions to be performed on this node.
- the class `continue_msg` – it is a helper class used as a signal between consecutive nodes;
- the function template `make_edge` connects the nodes and thus, it determines the sequence of the nodes (and the actions, at the same time) and – which can be more important – also the dependencies between nodes. A `continue_node`

can perform any actions if and only if all its previous nodes (connected with it directly by edges) finished their actions.

We should also mention `try_put` (a node method which sends the first `continue_msg` in order to start the computations) and `wait_for_all` (a graph method which waits for all the computations to finish).

Some fragments of the code of TLU(q)-TBB are shown in Listing 1 (it is `LU_Graph` – the main class responsible for building the graph and conducting the computations).

The maps (`nodes_lu`, `nodes_U`, `nodes_L`, `nodes_X`) store (smart) pointers to the nodes and are crucial in building the graph (thanks to them, all the created nodes are easy to reference).

The constructor creates nodes and edges with the use of the functions: `red_node`, `green_node`, `magenta_node` and `blue_node` (names according to colors from Figure 1). Only one (`green_node`) of these functions is shown – the others are similar.

Then, we have some helper functions (`red_action`, `green_action`, `magenta_action` and `blue_action`) which describe the desired computational actions for respective nodes and return computational kernels in the form of lambdas. In them, we use some macros but there are just BLAS routines inside. Again, only one function (`magenta_action` this time) is fully shown here.

Finally, we can see the method `run` which starts the computations and waits for them to finish.

The idea of the TWZ(r)-TBB implementation is the same – although the dependencies are somewhat different (see Figure 2) and the kernels are more complicated (what can be inferred from Algorithm 4).

5 NUMERICAL EXPERIMENTS

We tested the performance of two matrix decompositions, namely the tiled LU factorization and the tiled WZ factorization. We compared four implementations of these matrix decompositions, that is:

- TLU(q)-OpenMP – a parallel implementation of the tiled LU factorization with the use of single-threaded level 3 BLAS routines (`DTRSM` and `DGEMM`) from the MKL library and the OpenMP standard with tasks and the dynamic scheduling;
- TWZ(r)-OpenMP – a parallel implementation of the tiled WZ factorization with the use of single-threaded level 3 BLAS routines (`DTRSM` and `DGEMM`) from the MKL library and the OpenMP standard with tasks and the dynamic scheduling;
- TLU(q)-TBB – a parallel implementation of the tiled LU factorization with the use of single-threaded level 3 BLAS routines (`DTRSM` and `DGEMM`) from the MKL library and a TBB flow graph;
- TWZ(r)-TBB – a parallel implementation of the tiled WZ factorization with the use of single-threaded level 3 BLAS routines (`DTRSM` and `DGEMM`) from the MKL library and a TBB flow graph.

```
using namespace tbb::flow;
class LU_Graph {
private:
    graph g;
    /* other class members */
    std::map<std::vector<int>,
          std::shared_ptr<continue_node<continue_msg>>>
                nodes_lu, nodes_U, nodes_L, nodes_X;
public:
    LU_Graph(int q, /* other parameters */) { /*...*/ }

    void red_node(int k) { /*...*/ }
    void green_node(int k, int i) {
        nodes_U[ {k, i} ] =
            std::make_shared<continue_node<continue_msg>>
            (g,
             green_action(k, i));
        make_edge(*nodes_lu.at({k}),
                  *nodes_U.at({k, i}));
    }
    void magenta_node(int k, int j) { /*...*/ }
    void blue_node(int k, int i, int j) { /*...*/ }

    auto red_action(int k) { /*...*/ }
    auto green_action(int k, int i) { /*...*/ }
    auto magenta_action(int k, int j) {
        return [=](const continue_msg &) {
            TLU_DTRSM_no_copy(TLU_u, TLU_lo, TLU_l,
                              TILE_ADDR_X(L, k, k),
                              TILE_ADDR_X(A, k, j));
        };
    }
    auto blue_action(int k, int i, int j) { /*...*/ }

    void run() {
        nodes_lu.at({0})->try_put(continue_msg());
        g.wait_for_all();
    }
};
```

Listing 1. Fragments of the main class responsible for building the graph and conducting the computations in the TLU(q)-TBB implementation

Table 1 shows details of the specification of the hardware and software used in the numerical experiment. The flags used in compilation and linking were: `-mkl=sequential -fopenmp -O3 -ip -no-prec-div -fp-model fast=2 -std=c++14 -ltbb`. The theoretical peak performance (in Gflops) can be computed from the specification, with the use of the formula:

$$\#\text{ of cores} \times \text{clock frequency in GHz} \times \text{flops per cycle} = 24 \times 2.3 \times 16 = 883.2\,[\text{Gflops}]$$

CPU	2 × Intel Xeon E5-2670 v.3 (Haswell)
# of cores	24 (12 per socket)
# of threads	48 (2 per core)
clock	2.30 GHz
level 1 data cache	32 kB per core
level 2 cache	256 kB per core
compiler	Intel ICC 16.0.0
BLAS/LAPACK libraries	MKL 2016.0.109

Table 1. Hardware and software used in the experiments

The input matrices were generated by the authors. They were random, square, dense matrices, with a dominant diagonal of even sizes (1 024, 2 018, ... 14 336). Various numbers of tiles were tested, namely, each matrix was divided into 16, 32, 64, and 128 tiles for each side (both for the rows and the columns). The matrices are stored (from the beginning) in a tiled format [9] – as shown in Figure 4.

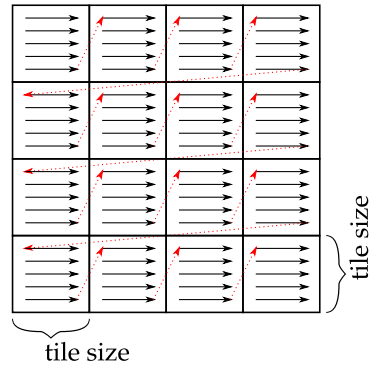


Figure 4. Memory layout of the test matrices. Arrows show data sequence in memory (black: within a tile; red: between tiles).

The performance times were measured with the use of a standard function, namely `(omp_get_wtime())`. The measured performance time does not include the

time needed for the matrix generation and for storing it in the aforementioned tiled format. However, it was quite short ($O(n^2)$), relative to the time of the factorizations ($O(n^3)$).

We set the number of OpenMP threads using the `omp_set_num_threads` function and the number of TBB threads with the use of `tbb::task_scheduler_init`. All the experiments reported below were performed with the use of the double-precision arithmetic.

5.1 Performance

In our experiments, as a metric, we use the number of floating-point operations per second (flops). The number of floating point operations for both the LU factorization and the WZ factorization of the matrix of the size $n \times n$ is $\frac{2}{3}n^3 + O(n^2)$, so it approximately equals $\frac{2}{3}n^3$.

Thus, to obtain the metric in Gflops ($= 10^9$ flops) we use the following formula

$$\frac{2n^3}{3 \cdot T \cdot 10^9},$$

where T is the execution time of a measured implementation. This metric allows comparing all implementations with the same measure.

Figure 5 presents the performance (in Gflops) of the TWZ(r)-OpenMP, TWZ(r)-TBB, TLU(q)-OpenMP and TLU(q)-TBB for the number of threads 24 for four different number of tiles (16, 32, 64, 128) as a function of the matrix size. We tested matrices of the sizes being multiples of 1 024, thus we were limited to the numbers dividing 1 024, that is, powers of 2. Thus, we chose above-mentioned numbers. However, some other tests (conducted on the matrix of the size 15 120 which has many more divisors) showed that the best results are obtained for q between 32 and 64 (LU) and r between 64 and 128 (WZ).

We can observe that the number of tiles has a great impact on the performance. For a wrongly chosen number of tiles (especially in the WZ factorization), the performance can drop drastically (e.g., even to about 100 Gflops for WZ with $r = 16$ in both frameworks). For the LU factorization and $q = 128$, the performance is also poor. Having analyzed all the experiments for TLU-OpenMP, we can see that the values $q = 16$ and $q = 128$ can be dismissed. On the other hand, for smaller matrices (up to the size 8 192), $q = 32$ is the best and for bigger ones (12 288 and more), we should choose $q = 64$. Between 8 192 and 12 288, the choice is ambiguous – $q = 32$ or $q = 64$ is better, but they are similar. For TLU-TBB, we can ignore $q = 128$ and $q = 64$ (never being the best choices). For small matrices (up to 6 144), the better is smaller of the remaining ones (that is, $q = 16$) and for bigger matrices (7 168 and more), the better is $q = 32$. For the tiled LU factorization, both frameworks are very close – usually, OpenMP prevails, but the differences are very minute. However, we can see that in TLU, OpenMP needs q to be twice as big as for TBB.

After the analysis for TWZ-OpenMP, we can see that this implementation behaves the best for $r = 64$ (up to the size 10 240) and for $r = 128$ (for the matrix

size 11 264 and more). The parameter $r = 16$ is never the best and for very small matrices, $r = 32$ gives good results. The TWZ-TBB implementation gives the best results for $r = 32$ (but only for the size 4 096 and less) and for $r = 64$ (from 5 120). The other values ($r = 16$ and $r = 128$) do not perform well. Again, the best results for both tiled WZ implementations are very close and we cannot assess which is the best. Moreover, in TWZ, we can also see that OpenMP needs r to be twice as big as for TBB.

Both the algorithms perform better in TBB if the parameter (q and r) is two times smaller than in OpenMP. In other words, the TBB versions work better for bigger portions of the data. Precisely: four times bigger – because the optimal linear size of the tile is twice bigger in TBB than in OpenMP; so the amount of data is four times bigger in TBB. That leaves an open question: why is this so?

To sum up, the performance depends strongly on the size of the matrix (what is quite obvious) and on the number of tiles (that is q or r) – thus, indirectly on the sizes of a single tile. The framework itself (OpenMP or TBB) has only a slight impact.

Implementation	Time [s]	Performance [Gflops]	% of Peak Performance
MKL LU	3.15	623.67	70.61 %
TLU(64)-OpenMP	2.98	658.28	74.53 %
TLU(32)-TBB	3.24	606.66	68.69 %
TWZ(128)-OpenMP	3.85	509.62	57.70 %
TWZ(64)-TBB	3.68	533.40	60.39 %

Table 2. The comparison of the best times and performances for four presented implementations for 24 threads and the matrix size of 14 336

In Table 2, we can see the best times and performances (with its values of r or q) and peak performance percentages, chosen experimentally for a matrix of size 14 336 and 24 threads – for each considered implementation and for a vendor MKL LU factorization. For the largest matrix size (14 336), the TLU algorithm achieves the best performance in the OpenMP implementation, although for the TWZ algorithm, the TBB implementation is better. Our implementations gave comparable results to the results of a vendor implementation (namely, the LU factorization without pivoting from the MKL library, that is `dgetrfnpi`). The same tests were also conducted for similar sizes (like 14 208, 14 464 and others; to exclude problems with cache associativity) and the general performance is very similar. However, not always the TLU(64)-OpenMP implementation was the best.

5.2 Speedup

In our proposed implementations only Stage 1 is not parallelized. In this section, we investigate the influence of this sequential part on the speedup possibilities.

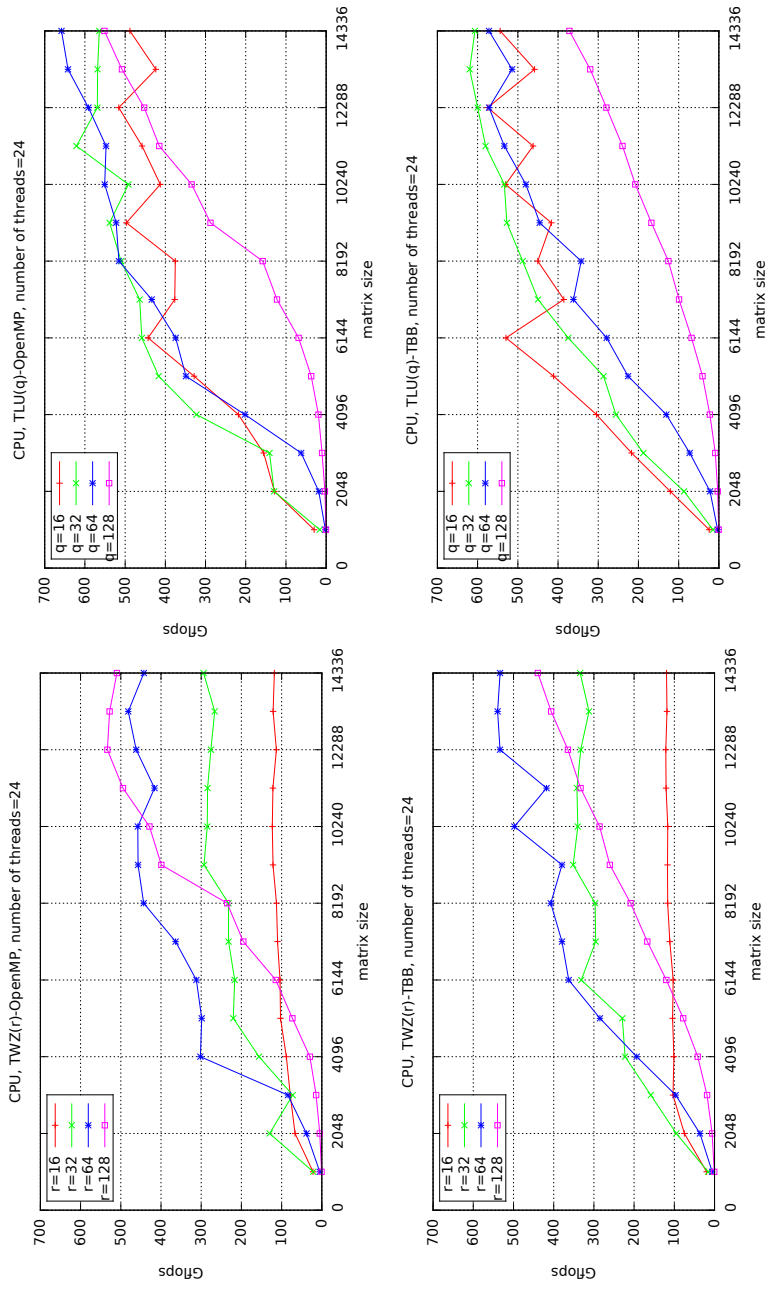


Figure 5. The performance in Gflops of the parallel tiled LU and WZ factorization algorithms for the number of threads 24 for four different number of tiles (16, 32, 64, 128) as a function of the matrix size

Let T_p be the time to perform the computation using p threads. Speedup for p threads is defined as:

$$S_p = \frac{T_1}{T_p}.$$

Figure 6 shows the experimental speedup (relative to the same algorithm run with the use of one thread – the times are shown in Table 3) as a function of the number of threads (1–27 threads) for different values of r and q for a matrix of the size 14 336.

Considering the best choices of q and r, the OpenMP implementations give a significantly better speedup (for 24 threads, it is more than 20). The best what the TBB implementations gain is speedup of 20. However, Table 3 shows that the TBB implementations have better performance for one thread, and thus they achieve poorer results in terms of relative speedup (Section 5.1 shows similar performance for OpenMP and TBB implementations of the same algorithm).

Implementation	Time [s]		
	1 Thread	12 Threads	24 Threads
TLU(64)-OpenMP	68.93	5.84	2.98
TLU(32)-TBB	62.99	5.48	3.23
TWZ(128)-OpenMP	75.05	6.98	3.85
TWZ(64)-TBB	70.02	6.25	3.65

Table 3. The performance time for selected numbers of threads and the matrix size of 14 336

We can see that the OpenMP implementations scale better – up to 24 threads. For more threads, the hyperthreading turns on and it does not improve the performance – aggravating the results sometimes. In the case of TBB, the scalability collapses somewhat earlier. For both frameworks, we should choose the maximum number of physical cores as the number of threads, that is, 24 in our environment.

The speedup is sensitive to the values r and q. However, both implementations are scalable (up to the number of physical cores, that is 24) for well-chosen q and r.

5.3 Scalability

Figure 7 shows the weak scalability of the algorithms. The tests here are run for various numbers of processors, however, the amount of the work is chosen to be proportional to the number of employed cores. To achieve a nice weak scalability [10], we expect the plots (of the execution time versus the number of processors employed) to be horizontal. We can see that both frameworks (that is OpenMP and TBB) and both methods (LU and WZ) achieve similar, very good, weak scalability.

Figure 8 shows the strong scalability of the algorithms. This time, the tests were run for various numbers of processors, but the amount of the work was always the same (the size of the matrix was 14 336). The plot (of the execution time versus the number of processors employed) was done in log-log scales [10]. In such a plot,

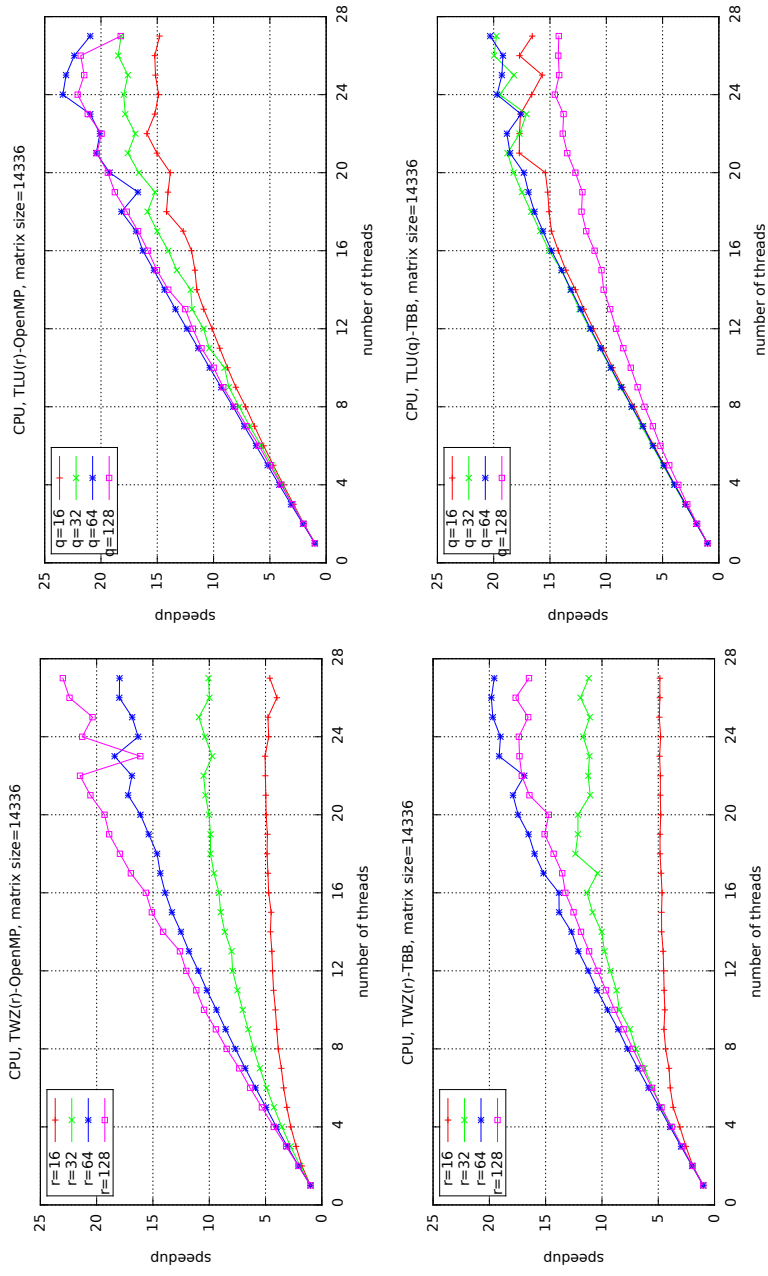


Figure 6. The speedup of the parallel tiled LU and WZ factorization algorithms as a function of the number of threads for different values of r and q for matrix size equals 14 336

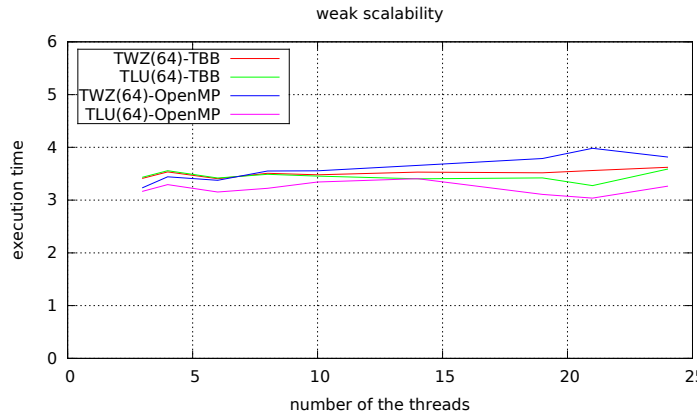


Figure 7. Weak scalability of the tested algorithms for the selected parameters

a good strong scalability should give a straight line with the slope -1. We can see that the scalability is not bad for all cases. However, close to the maximal number of processors, something spoils (what can be also seen in Figure 6). It can be explained by an automatic constraint on the energy used by the processor.

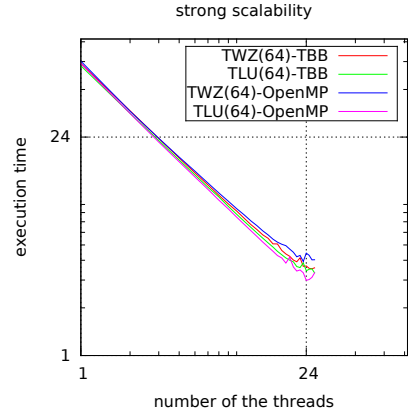


Figure 8. Strong scalability of the tested algorithms for the selected parameters

6 CONCLUSION

In this work, we reported numerical experiments aimed to compare two parallel task frameworks, namely OpenMP and TBB for two matrix factorizations. We focused on two matrix factorizations which use BLAS functions from MKL library and are

computationally intensive. We implemented these matrix factorizations using tasks with dependencies in OpenMP and TBB. We chose these frameworks because they differ significantly in their approach to defining dags and dependencies. Moreover, OpenMP is a language extension, but TBB is an ordinary library.

The TBB library seems to be more flexible – as it is just a library seamlessly fitting into the C++ language and its other libraries. Arbitrary C++ types can be used with TBB, whereas OpenMP have troubles with some more complicated types (like classes, templates, lambdas) and they cannot be used directly. High-level abstraction (provided by these types) does not port well into OpenMP. Conversely, in TBB they are treated quite transparently, because TBB is not an overlay onto the language – as OpenMP is. As a library, TBB synergizes with C++ standard library as well as with external libraries. We can also easily use templates and lambdas which facilitate the creation of flexible and reusable code. On the other hand, in OpenMP, pragmas do not accept many C++ constructs (sometimes not even macros) and we must employ some tricks to achieve our goals.

The TBB library also offers more tools to better control the execution. With the use of OpenMP, we are not building the graph – this is done by the compiler and run-time system. We can only give the dependencies and trust that the graph will be correct. However, sometimes (especially when the graph is explicit – as in our case) it is easier to build the graph than to write complex (and sometimes artificial) dependencies. Moreover, the dependency graph built by a programmer in TBB can be an arbitrary graph (contrary to OpenMP, where graphs are not arbitrary and they must be given implicitly, by dependencies – as we mentioned above). Each graph node represents a task and its edges describe arbitrary dependencies between them. The task scheduling is a very important part of TBB. It automatically allocates tasks to workers (threads) to maintain the best load balancing. But the main advantage of the TBB is that it is completely compatible with the C++ language and can be freely used with other libraries – which is priceless in advanced applications.

On the other hand, OpenMP is very popular and quite simple to use. It is also a portable and (de facto) standard approach. However, OpenMP has some limitations. It causes problems when dependencies are more complicated, does not allow using some C/C++ constructs (even some simple expressions or data members) in pragmas and clauses, forcing a programmer to use unnatural notations (as illegible casts). Specifically to dependencies, if array sections appear in them, they must be either the same or disjoint. It is also a C-based standard so it does not treat well some C++ elements (like references). Thus, OpenMP is a common and quite efficient tool, although TBB is more programmer-friendly and sometimes TBB's features and flexibility make TBB the only option.

There was not a clear performance relation among the considered frameworks, and the differences between them were small in most cases. In fact, the average performance difference between the slowest and the fastest implementation in our tests was about 19 %. However, the OpenMP implementations relative speedup is clearly better than that for TBB.

There is also a programmer's experience issue. For simpler projects, OpenMP seems easier – especially when we have a working sequential implementation. On the other hand, for more complicated problems, notably ones needing some code reuse (as, for example, refinement techniques [2, 4] where the same algorithm is used with different precision types), TBB is better for an experienced developer, although, TBB is also more demanding. However, the merits of TBB (its flexibility, generality, code readability) prevails over the OpenMP (its limitations and error proneness).

Our corollaries can be generalized to a wide class of algorithms. Namely, all the linear algebra algorithms – as, for example, various factorizations (Cholesky, QR, etc.), matrix-matrix multiplication (GEMM) and iterative methods (Jacobi, Gauss-Seidel, GMRES) – which can be designed as a tiled version (that is, with the use of square blocks and the special storing format mentioned in Section 5) can be also implemented with tasks (using both OpenMP and TBB) similarly.

Further work is needed to determine other ways in which OpenMP and TBB frameworks could potentially be improved and whether additional information could be provided to enable better performance. Also, there is a plenty other computing areas where we can use the task approach – as, for example, in sparse computations, machine learning, etc.

REFERENCES

[1] Agullo, E.—Demmel, J.—Dongarra, J.—Hadri, B.—Kurzak, J.—Langou, J.—Ltaief, H.—Luszczek, P.—Tomov, S.: Numerical Linear Algebra on Emerging Architectures: The PLASMA and MAGMA Projects. Journal of Physics: Conference Series, Vol. 180, 2009, No. 1, Art. No. 012037, doi: 10.1088/1742-6596/180/1/012037.

[2] Baboulin, M.—Buttari, A.—Dongarra, J.—Kurzak, J.—Langou, J.—Langou, J.—Luszczek, P.—Tomov, S.: Accelerating Scientific Computations with Mixed Precision Algorithms. Computer Physics Communications, Vol. 180, 2009, No. 12, pp. 2526–2533, doi: 10.1016/j.cpc.2008.11.005.

[3] Buttari, A.—Langou, J.—Kurzak, J.—Dongarra, J.: A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures. Parallel Computing, Vol. 35, 2009, No. 1, pp. 38–53, doi: 10.1016/j.parco.2008.10.002.

[4] Bylina, B.—Bylina, J.: Mixed Precision Iterative Refinement Techniques for the WZ Factorization. Proceedings of the 2013 Federated Conference on Computer Science and Information Systems, 2013, pp. 425–431.

[5] Bylina, B.—Bylina, J.: OpenMP Thread Affinity for Matrix Factorization on Multicore Systems. Proceedings of the 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), Annals of Computer Science and Information Systems, Vol. 11, 2017, pp. 489–492, doi: 10.15439/2017F231.

[6] Chandra, R.—Dagum, L.—Kohr, D.—Maydan, D.—McDonald, J.—Menon, R.: Parallel Programming in OpenMP. Morgan Kaufmann Publishers, San Francisco, 2001, doi: 10.1016/b978-155860671-5/50003-7.

[7] Dongarra, J. J.—Du Croz, J.—Hammarling, S.—Duff, I. S.: A Set of Level-3 Basic Linear Algebra Subprograms. ACM Transactions on Mathematical Software, Vol. 16, 1990, pp. 1–17, doi: 10.1145/77626.79170.

[8] Evans, D. J.—Hatzopoulos, M.: A Parallel Linear System Solver. International Journal of Computer Mathematics, Vol. 7, 1979, No. 3, pp. 227–238, doi: 10.1080/00207167908803174.

[9] Gustavson, F. G.: High-Performance Linear Algebra Algorithms Using New Generalized Data Structures for Matrices. IBM Journal of Research and Development, Vol. 47, 2003, No. 1, pp. 31–55, doi: 10.1147/rd.471.0031.

[10] Heath, M. T.: A Tale of Two Laws. The International Journal of High Performance Computing Applications, Vol. 29, 2015, No. 3, pp. 320–330, doi: 10.1177/1094342015572031.

[11] Igual, F. D.—Chan, E.—Quintana-Ortí, E. S.—Quintana-Ortí, G.—van de Geijn, R. A.—Van Zee, F. G.: The FLAME Approach: From Dense Linear Algebra Algorithms to High-Performance Multi-Accelerator Implementations. Journal of Parallel and Distributed Computing, Vol. 72, 2012, No. 9, pp. 1134–1143, doi: 10.1016/j.jpdc.2011.10.014.

[12] Kukanov, A.—Voss, M. J.: The Foundations for Scalable Multi-Core Software in Intel Threading Building Blocks. Intel Technology Journal, Vol. 11, 2007, No. 4, pp. 309–322, doi: 10.1535/itj.1104.05.

[13] Kurzak, J.—Luszczek, P.—YarKhan, A.—Faverge, M.—Langou, J.—Bouwmeester, H.—Dongarra, J.: Multithreading in the PLASMA Library. In: Rajasekaran, S., Fiondella, L., Ahmed, M., Ammar, R. A. (Eds.): Multicore Computing: Algorithms, Architectures, and Applications. Chapter 5. Chapman and Hall/CRC, 2013, p. 119–142, doi: 10.1201/b16293-11.

[14] Marowka, A.: TBBench: A Micro-Benchmark Suite for Intel Threading Building Blocks. Journal of Information Processing Systems, Vol. 8, 2012, No. 2, pp. 331–346, doi: 10.3745/jips.2012.8.2.331.

[15] Marqués, M.—Quintana-Ortí, G.—Quintana-Ortí, E. S.—van de Geijn, R. A.: Using Desktop Computers to Solve Large-Scale Dense Linear Algebra Problems. The Journal of Supercomputing, Vol. 58, 2011, No. 2, pp. 145–150, doi: 10.1007/s11227-010-0394-2.

[16] Olivier, S. L.—Prins, J. F.: Comparison of OpenMP 3.0 and Other Task Parallel Frameworks on Unbalanced Task Graphs. International Journal of Parallel Programming, Vol. 38, 2010, No. 5-6, pp. 341–36, doi: 10.1007/s10766-010-0140-7.

[17] Pheatt, C.: Intel Threading Building Blocks. Journal of Computing Sciences in Colleges, Vol. 23, 2008, No. 4, pp. 298–298.

[18] Quintana-Ortí, G.—Quintana-Ortí, E. S.—van de Geijn, R. A.—Van Zee, F. G.—Chan, E.: Programming Matrix Algorithms-by-Blocks for Thread-Level Parallelism. ACM Transactions on Mathematical Software, Vol. 36, 2009, No. 3, pp. 14:1–14:26, doi: 10.1145/1527286.1527288.

[19] Rao, S. C. S.: Existence and Uniqueness of WZ Factorization. Parallel Computing, Vol. 23, 1997, No. 8, pp. 1129–1139, doi: 10.1016/s0167-8191(97)00042-2.

[20] Stpiczyński, P.: Language-Based Vectorization and Parallelisation Using Intrinsics, OpenMP, TBB and Cilk Plus. Journal of Supercomputing, Vol. 74, 2018, pp. 1461–1472, doi: 10.1007/s11227-017-2231-3.

[21] Tang, P.: Measuring the Overhead of Intel C++ Concurrent Collections over Threading Building Blocks for Gauss–Jordan Elimination. Concurrency and Computation: Practice and Experience, Vol. 24, 2012, pp. 2282–2301, doi: 10.1002/cpe.2811.

[22] Virouleau, P.—Brunet, P.—Broquedis, F.—Furmento, N.—Thibault, S.—Aumage, O.—Gautier, T.: Evaluation of OpenMP Dependent Tasks with the KASTORS Benchmark Suite. In: DeRose, L., de Supinski, B. R., Olivier, S. L., Chapman, B. M., Müller, M. S. (Eds.): Using and Improving OpenMP for Devices, Tasks, and More (IWOMP 2014). Springer, Heidelberg, Lecture Notes in Computer Science, Vol. 8766, 2014, pp. 16–29, doi: 10.1007/978-3-319-11454-5_2.

[23] Yalamov, P.—Evans, D. J.: The WZ Matrix Factorization Method. Parallel Computing, Vol. 21, 1995, No. 7, pp. 1111–1120, doi: 10.1016/0167-8191(94)00088-r.

[24] `https://www.cilkplus.org/`.

[25] `https://www.khronos.org/opencl/`.

Jarosław Bylina is a mathematics graduate (1998) with his Ph.D. in computer science (Distributed methods to solve Markovian models of computer networks, done at The Silesian University of Technology in Gliwice, Poland in 2006. He has been working in the Institute of Mathemtaics of Marie Curie-Skłodowska University in Lublin (Poland) since 1998, now as Assistant Professor. He is interested in numerical methods for Markov chains, modelling of teleinformatic systems and parallel and distributed computing and processing.

Computing and Informatics, Vol. 38, 2019, 223–239, doi: 10.31577/cai_2019_1_223

STUDY ON UNSUPERVISED FEATURE SELECTION METHOD BASED ON EXTENDED ENTROPY

Zhanquan Sun

Engineering Research Center of Optical Instrument and System
Ministry of Education, Shanghai Key Lab of Modern Optical System
University of Shanghai for Science and Technology, Shanghai, 200093, China
e-mail: sunzhq@sdas.org

Feng Li

Department of History, College of Liberal Arts, Shanghai University
Shanghai, 200436, China
e-mail: namelf@126.com

Huifen Huang

Shandong Yingcai University
Shandong, China
e-mail: shouyu1976@163.com

Abstract. Feature selection techniques are designed to find the relevant feature subset of the original features that can facilitate clustering, classification and retrieval. It is an important research topic in pattern recognition and machine learning. Feature selection is mainly partitioned into two classes, i.e. supervised and unsupervised methods. Currently research mostly concentrates on supervised ones. Few efficient unsupervised feature selection methods have been developed because no label information is available. On the other hand, it is difficult to evaluate the selected features. An unsupervised feature selection method based on extended entropy is proposed here. The information loss based on extended entropy is used to measure the correlation between features. The method assures that the selected features have both big individual information and little redundancy information with

the selected features. At last, the efficiency of the proposed method is illustrated with some practical datasets.

Keywords: Unsupervised feature selection, extended entropy, information loss, correlation value

1 INTRODUCTION

In recent years, data has become increasingly larger in number of features in many applications such as genome projects, text categorization, image retrieval and customer relationship management, etc. [1, 2]. It may cause serious problems to many machine learning algorithms with respect to scalability and learning performance. How to select the most informative variable combination is a crucial problem. Feature selection techniques are designed to find the relevant feature subset of the original features that can facilitate clustering, classification and retrieval [3, 4]. Feature selection is an important research issue in machine learning and pattern recognition. Lots of research work has been done on the topic. Based on whether the label information is available, feature selection is mainly partitioned into two types, i.e. supervised and unsupervised methods [5]. The former method is based on labeled samples for classification problems. The latter method is mainly used to analyze unlabeled data for clustering problems. Many supervised feature selection methods have been proposed and applied to many application areas. Typical supervised feature selection methods include correlation coefficient method, information gain, logistical regression, regularized method, etc. [6, 7, 8, 9]. In general, supervised feature selection is better in performance than unsupervised methods. But in practice, data samples are usually unlabeled. How to improve the performance of unsupervised feature selection is still a difficult problem to be resolved.

Supervised feature selection methods usually evaluate the importance of a feature by the correlation value between features and class variable. However, in practice, it is expensive or impossible to label large-scale samples in many applications. Hence, it is great significance to develop unsupervised feature selection algorithms that take full use of the unlabeled samples to select the most informative features. Some unsupervised feature selection methods have been proposed, such as maximum variance method, Laplacian score method, clustering based method, etc. [10, 11, 12]. For dealing with multi cluster feature selection problem, spectral regression and sparse space learning based method was proposed [13]. Feature selection is the process of selecting the most informative feature combination. The raw dataset contains many features that are either redundant or irrelevant. They can be removed without incurring much loss of information. Correlation metric is used to measure the relationship between features. Feature selection results based on different correlation metrics are different. Many kinds of correlation metrics have been proposed, such as Pearson correlation coefficient, mutual information and

so on. Mutual information can measure arbitrary statistical dependences between variables [14]. But the computational cost of mutual information between continuous variables and mutual information between discrete and continuous variables is expensive. Information bottleneck theory based information loss is an efficient correlation metric [15, 16]. It has been applied to many complicated clustering problems. But the information loss based on probability cannot process continuous variables. In this paper, information bottleneck theory based information loss is adopted to measure the correlation between features. For improving the general adapt capability, extended entropy is proposed and information loss is calculated based on it. The proposed feature selection method takes both the feature's individual entropy and the redundancy information with the selected features into consideration. It assures that the selected feature combination has the maximum information. For determining the number of selected features, an objective rule is proposed. The method combines the change ratio and the gradient ratio of information increase. At last, the efficiency of the proposed method is illustrated with some practical dataset.

The rest of this paper is organized as follows. Section 2 presents the definition of extended entropy. Section 3 introduces information bottleneck theory and information loss. Section 4 proposes the calculation method of information loss based on extended entropy. Feature selection procedure based on extended entropy is presented in Section 5. Some practical datasets are analyzed with the proposed method in Section 6. Concluding remarks are described in Section 7.

2 EXTENDED ENTROPY

Entropy is a way to measure the amount of information in a signal based on probabilities. Classic entropy is based on probability. Data has no statistical characteristics in many applications. A novel entropy definition, i.e. extended entropy, is proposed here. Extended entropy is not based on probability but on ratio. The definition is as follows.

2.1 Shannon Entropy

Let feature variables be denoted by vector $X = (X_1, X_2, \ldots, X_m)^T$, where $X_i = (x_{ij})$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, q$, denotes the i^{th} feature variable with q difference values, and class variable be denoted by Y, $Y = (y_i)$, $i = 1, 2, \ldots, k$. It means that all features are projected to k different classes. $p(X_i)$ denotes the probability distribution of feature variableX_i, p_Y denotes the probability distribution of class variable Y, and $p(X_i, Y)$ denotes the joint probability distribution of X_i and Y. All probability distributions are calculated according to sample statistics. The Shannon entropy H of feature variable X_i can be described as

$$H(X_i) = -\sum_{j=1}^{q} p(x_{ij}) \operatorname{logp}(x_{ij}). \tag{1}$$

Shannon entropy of class variable Y can be described as

$$H(Y) = -\sum_{i=1}^{k} p(y_i) \operatorname{logp}(y_i). \tag{2}$$

Joint entropy between feature variables and class variable is

$$H(X_i, Y) = -\sum_{j=1}^{q} \sum_{l=1}^{k} p(x_{ij}, y_l) \operatorname{logp}(x_{ij}, y_l) \tag{3}$$

where X_i can be substituted by subset of feature vector S, i.e., the joint entropy can be generalized to p variables.

2.2 Extended Entropy

Let N data vectors be denoted by y_i, $i = 1, 2, \ldots, N$, each vector has n positive number $y_{i1}, y_{i2}, \ldots, y_{in}$, $i = 1, 2, \ldots, N$ and the ratio between each positive number and the sum of all the positive number is

$$r(y_{ij}) = y_{ij}/(y_{i1} + y_{i2} + \ldots + y_{in}). \tag{4}$$

$r(y_{ij})$ is similar to the probability that satisfies $\sum_{j=1}^{n} r(y_{ij}) = 1$, and $r(y_{ij}) \geq 0$, $i = 1, 2, \ldots, n$. Extended entropy based on ratio is defined as

$$S(y_i) = -\sum_{j=1}^{n} r(y_{ij}) \ln r(y_{ij}). \tag{5}$$

3 INFORMATION BOTTLENECK THEORY

Information bottleneck (IB) theory is proposed to operate clustering problem. The theory is based on mutual information. The joint distribution of the object space X and the feature space Y is denoted by $p(x, y)$. According to the IB principle a clustering X that minimizes the information loss $I(X; \hat{X}) = I(X; Y) - I(\hat{X}; Y)$ is optimized. $I(X; \hat{X})$ is the mutual information between X and $\hat{X}$

$$I(X; \hat{X}) = \sum_{x;\hat{x}} p(x) p(\hat{x} \mid x) \log(p(\hat{x} \mid x)/(p(\hat{x}). \tag{6}$$

The IB principle is motivated from Shannon's rate-distortion theory which provides lower bounds on the number of classes. Given a random variable X and a distortion $d(x_1, x_2)$ measure, the symbols of X are represented with no more than R bits. The rate-distortion function is given

$$D(R) = \min_{p(\hat{x}|x) I(X;\hat{X}) \leq R} Ed(x, \hat{x}) \tag{7}$$

where $Ed(x,\hat{x}) = \sum_{x,\hat{x}} p(x)p(\hat{x} \mid x)d(x,\hat{x})$. The loss of the mutual information between X and Y caused by the clustering $\hat{X}$ can be viewed as the average of this distortion measure

$$\begin{aligned} d(x,\hat{x}) &= I(X;Y) - I(\hat{X};Y) \\ &= \sum_{x,\hat{x},y} p(x,\hat{x},y)\log(p(y \mid x))p(y) - \sum_{x,\hat{x},y} p(x,\hat{x},y)\log(p(y \mid \hat{x}))/p(y) \\ &= ED(p(x,\hat{x})||p(y,\hat{x})) \end{aligned} \tag{8}$$

where $D(f\|g) = E_f \log(f/g)$ is the Kullback-Lerbler divergence. The rate distortion function is

$$D(R) = \min_{p(\hat{x}|x)I(X;\hat{X})\leq R} (I(X;Y) - I(\hat{X};Y)) \tag{9}$$

which is exactly the minimization criterion proposed by the IB principle, i.e., finding a clustering that minimizes the loss of mutual information between the objects and the features. Let c_1 and c_2 be two clusters of symbols, the information loss due to the merging is

$$d(c_1,c_2) = I(c_1;Y) + I(c_2;Y) - I(c_1,c_2;Y), \tag{10}$$

information theory operation reveals

$$d(c_1,c_2) = \sum_{y,i=1,2} p(c_i)p(y \mid c_i)\log(p(y \mid c_i))/(p(y \mid c_1 \cup c_2)) \tag{11}$$

where $p(c_i) = |c_i| / |X|$, $|c_i|$ denotes the cardinality of c_i, $|X|$ denotes the cardinality of object space X, $p(c_1 \cup c_2) = |c_1 \cup c_2| / |X|$. It assumes that the two clusters are independent when the probability distribution is combined. The combined probability of the two clusters is

$$p(y \mid c_1 \cup c_2) = \sum_{i=1,2} |c_i| / (c_1 \cup c_2)p(y \mid c_i). \tag{12}$$

4 INFORMATION LOSS BASED ON EXTENDED ENTROPY

According to Equation (12), information loss based on probability can only process discrete variables. Therefore, the classic information loss definition is not suitable in many situations. Extended entropy can deal with any kind of positive dataset. We introduce extended entropy into information bottleneck theory. In the method, each element of the dataset y is taken as a different value probability of which is the ratio between each element's value and the sum of all the element's values. Let n samples and each sample include m features. Calculate the correlations between features according to the values in each sample. Each feature can be taken as an n dimension vector, i.e. $y_i = y_{i1}, y_{i2}, \ldots, y_{in}, i = 1, 2, \ldots, m$. Each sample is taken as a value of the feature variable. n samples means each feature has n values.

The extended probability of feature y_i is calculated according to the ratio between the feature value and the sum, i.e.

$$r(y_{ij}) = y_{ij}/(y_{i1} + y_{i2} + \cdots + y_{in}). \tag{13}$$

It can satisfy the conditions requirements, i.e. $\sum_{j=1}^{n} r(y_{ij}) = 1$ and $r(y_{ij}) \geq 0$, $j = 1, 2, \ldots, n$. The extended entropy based on extended probability is defined as

$$S(y_i) = -\sum_{j=1}^{n} r(y_{ij}) \ln r(y_{ij}). \tag{14}$$

The information loss due to the merging of two clusters is coherent to that of IB

$$d(c_1, c_2) = \sum_{i=1,2} \sum_{j=1}^{n} r(y_j \mid c_i) \log(r(y_j \mid c_i))/(r(y_j \mid c_1 \cup c_2)). \tag{15}$$

According to the calculation equation of information loss, after $p, q \in \{1, 2, \ldots, n\}$ being combined into a variable c, the extended probability of combine variable c can be denoted

$$r(y_{cj}) = \frac{|y_p|}{|y_p \cup y_q|} r(y_{pj}) + \frac{|y_p|}{|y_p \cup y_q|} r(y_{qj}). \tag{16}$$

5 FEATURE SELECTION BASED ON EXTENDED ENTROPY (FSBEE)

Unsupervised feature selection method FSBEE is as follows. A novel correlation definition is introduced. The correlation between feature variable X and feature variable Y is defined as

$$\rho(X, Y) = 1/d(X, Y). \tag{17}$$

The information loss value is inverse proportion to the correlation value. The features of Y are combined into one variable according to Equation (16) firstly when it is a feature set. Then the information loss $d(X, Y)$ is calculated according to Equation (15).

5.1 First Feature Selection

The first feature is selected according to the correlation value of each feature. The initial feature variable set is denoted by $X = \{X_1, X_2, \ldots, X_m\}$. U is used to denote unselected feature set and S is used to denote the selected feature set. At first, U is set the initial feature set and S is set null, i.e. $U = X$ and $S = \Phi$. The feature that has the biggest correlation value with the other feature subset is selected as the first one, i.e.

$$x_l = arg \max_{1 \leq i \leq m} \rho(X_i, (X \backslash X_i)). \tag{18}$$

The maximum correlation value means that the feature can represent the other features in maximum degree. The selected feature is added to selected feature set S and removed from unselected feature set, i.e. $S = \{X_l\}$ and $U = X \backslash X_l$.

5.2 Feature Selection Procedure

After the first feature has been selected, the other features are selected according to the following procedure. The k^{th} feature X_l is selected according to the increase of correlation value. The calculation of increase value is as follows.

$$X_l = \arg\max_{X_i \in U} \{\rho(X_i, (U \backslash X_i)) * d(X_i, S)\}, \tag{19}$$

$$f_k = \max_{X_i \in U} \{\rho(X_i, (U \backslash X_i)) * d(X_i, S)\}. \tag{20}$$

It means that the selected feature can represent the other unselected features in maximum degree. At the same time, the selected feature should provide the least redundancy information to the selected feature set S. The candidate feature should have the largest distance to the selected features. Then the selected feature X_l is added to the selected feature set S and removed from the unselected feature set U, i.e. $S = \{S, X_l\}$ and $U = U \backslash \{X_l\}$. Through iterating the above procedure, features are selected.

5.3 Determination of the Number of Selected Features

No objective rule is available to determine the number of selected features currently. It is often prescribed previously. In this paper, the number of selected features is determined according to the following rules. In the selection procedure, each step corresponds to an increase value of correlation. In general, the value will be in decreasing trend. The gradient ratio between the current step and the first step is calculated according to the following equation.

$$u = (f_{k-1} - f_k)/(f_1 - f_2). \tag{21}$$

The ratio between the increase value corresponding to the current step and that to the first step is denoted by

$$v = f_k/f_1. \tag{22}$$

Threshold values α for u and β for v are prescribed. When the values are less than prescribed threshold values, i.e. $u < \alpha$ and $v < \beta$, the feature selection procedure is stopped. The threshold values are prescribed according to a practical problem. Less threshold value corresponds to larger selected feature number. In general, the two threshold values are in the interval [0,1]. They are set to a less value when the analysis problem is complicated and to a bigger value to a simple problem. Pseudo-code of the feature selection procedure is summarized as Figure 1.

```
Main class
    Dataset load D. //Read data into matrix.
    Initialize variable vector U = X and S = ∅.
    Initialize α and β.
    For i←o to m
        Calculate correlation increase value f(i)
    End
    Obtain the maximum value f_max and its index k_max .
    Update S = {X_l} and U = X\X_l
      While (true)
          Calculate f(i), i ∈ U;
          Obtain f_max and its index k_max ;
          Update S = {S, X_l} and U = U\{X_l}.
          Calculating u and v
          If (u < α && v < β)
          Break;
      End
End mian class
```

Figure 1. Pseudo code of the feature selection procedure

From the above calculation procedure, the computation complexity of the feature selection procedure is about $O(knm^2)$, where k is the number of selected features, m is the number of total features, and n is the number of instances.

6 EXAMPLES

6.1 Data Source

The datasets are downloaded from the UCI machine learning website [17]. For proving the efficiency of the proposed unsupervised feature selection method, some classification datasets are analyzed. They are breast cancer clinic data, smart phone record, credit card record, mesothelioma data and image segmentation set. The basic information, i.e. number of features, number of instances, number of classes, of each dataset is listed in Table 1.

6.2 Feature Selection with the FSBEE

The label information is ignored during feature selection procedure. The label information is used to analyze the performance of feature selection method through comparing the classification results. For calculating the extended entropy, all features

Data Source	Number of Features	Number of Samples	Number of Classes
Breast cancer diagnostic data	30	569	2
Smart phone record	561	10 299	6
Credit card record	24	30 000	2
Mesothelioma data	34	324	2
Image segmentation data	19	2 210	7

Table 1. Dataset information

are transformed into positive values. In this example, all features are normalized into the interval $[0, 1]$. Then the features are selected according to the proposed method in Section 5. The selection results are as follows.

(1) Breast cancer diagnostic data. The data is provided by University of Wisconsin, Clinical Sciences Center. It is used for breast tumor diagnosis. The dataset includes 569 samples and each sample has 30 real-value features. They are categorized into two classes, i.e. benign or malignant. Firstly, each feature is normalized to the interval $[0, 1]$. Then, the extended probability of each sample can be calculated according to (13). The features are selected according to Section 5. For determining the number of selected features, the threshold value is set $\alpha = 0.1$ and $\beta = 0.1$. The increase of correlation value corresponding to each step is shown in Figure 2. At last, 14 features are selected.

(2) Smartphones dataset. The smartphones data is collected to recognize human activity [18]. 10 299 human activity records are collected and each record has 561 features. The human activity is categorized into 6 classes. Firstly, the features are normalized to the interval $[0, 1]$. Then features are selected according to the FSBEE method. The threshold values are set $\alpha = 0.01$ and $\beta = 0.01$. The correlation increase value of each step is shown in Figure 3. At last, 178 features are selected.

(3) Credit card record. The dataset is collected to identify the customer whether credible or not credible according to his/her personal payment record. There are 30 000 records in the dataset and each record has 24 attributes. Among the attributes, some are binary variables and some are continuous variables. For computing convenience, all features are normalized into interval $[0, 1]$ and taken as continuous variables. Then features are selected according to the FSBEE method. The threshold value is set $\alpha = 0.1$ and $\beta = 0.1$. The increase of correlation value of each step is shown in Figure 4. At last, 10 features are selected.

(4) Mesothelioma data. The dataset is collected by Dicle University from real patient reports. It is used to identify whether the patient's mesothelioma is benign or malignant. There are 324 samples in the dataset and each sample has 34 attributes. Firstly, all attributes are normalized to the interval $[0, 1]$. Then features are selected to according to the FSBEE method. The threshold value is

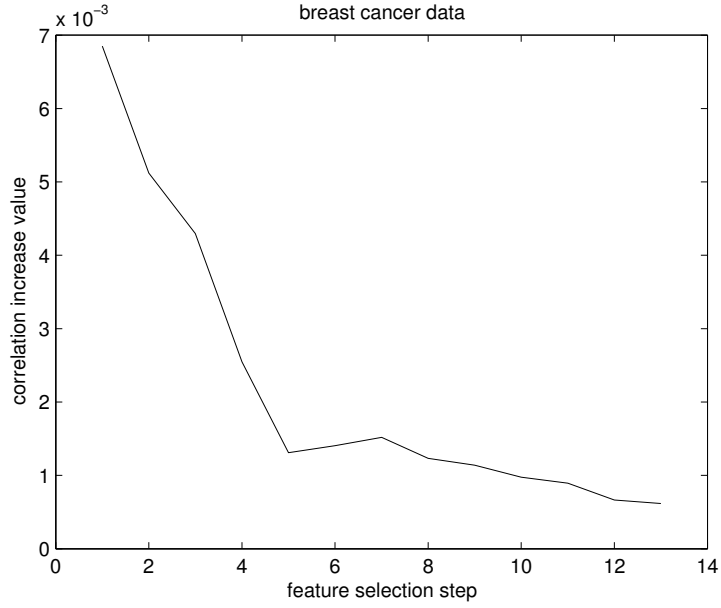


Figure 2. The increase in correlation value of breast cancer data

set $\alpha = 0.1$ and $\beta = 0.1$. The increase of correlation value of each step is shown in Figure 5. At last, 15 features are selected.

(5) Image segmentation. The instances were drawn randomly from a database of 7 outdoor images, i.e. brickface, sky, foliage, cement, window, path, grass. Each instance includes 19 features. The images were hand segmented to create a classification for every pixel. 210 images are taken as training set and 2 000 images are taken as test set. Firstly, all attributes are normalized to the interval $[0, 1]$. Then features are selected according to the FSBEE method. The threshold value is set $\alpha = 0.1$ and $\beta = 0.1$. The correlation increase value of each step is shown in Figure 6. At last, 12 features are selected.

6.3 Result Comparison

For comparison, information gain (IG), correlation coefficient, logistic regression etc. supervised feature selection methods are used to select the features. libSVM [19] is used to classify the dataset according to the selected features obtained with different feature selection methods. The number of selected features is the same as that of FSBEE model. The classification results are listed in Table 2. The computation time of the feature selection corresponding to each dataset are listed in Table 3. K-mean based feature selection, variance based feature selection and mutual infor-

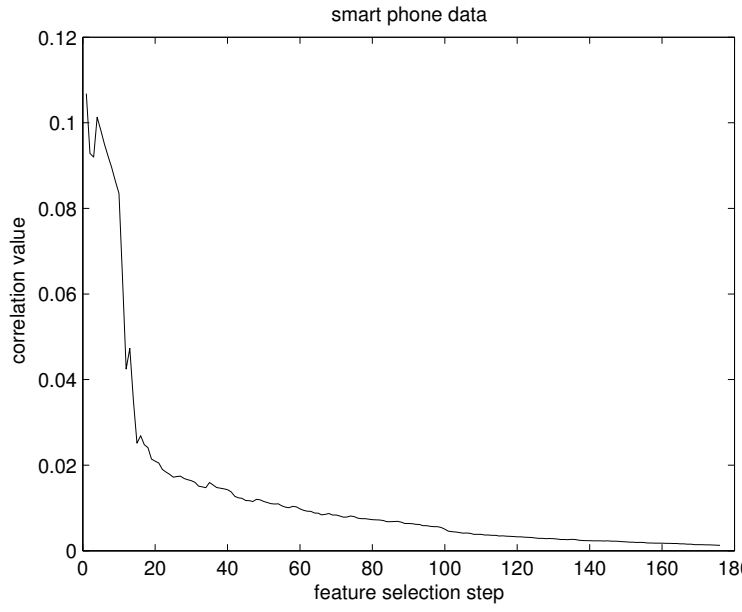


Figure 3. The increase in correlation value of breast cancer data

mation based feature selection (FSBMI) etc. unsupervised methods are used to select the features. The number of selected features is the same as that of FSBEE. The classification results based on different unsupervised feature selection method are listed as in Table 4. The computation time of the unsupervised feature selection corresponding to each dataset are listed in Table 5. From above analysis results we can find that the proposed FSBEE is an efficient unsupervised feature selection method. Through comparing with supervised feature selection methods, the classification accuracy of the proposed FSBEE method is close to that of the supervised method and even better than some supervised methods. Through comparing with other unsupervised methods, the classification accuracy of the proposed method is better. The proposed method is easy to be operated. The computation complexity will increase with the number of feature numbers and the number of selected feature numbers. But it is more efficient than the unsupervised feature selection method FSBMI.

7 CONCLUSIONS

Feature selection for unlabeled samples is a very important task for many pattern recognition problems. For improving the efficiency and performance of unsupervised feature selection, the paper develops a novel unsupervised feature selection method.

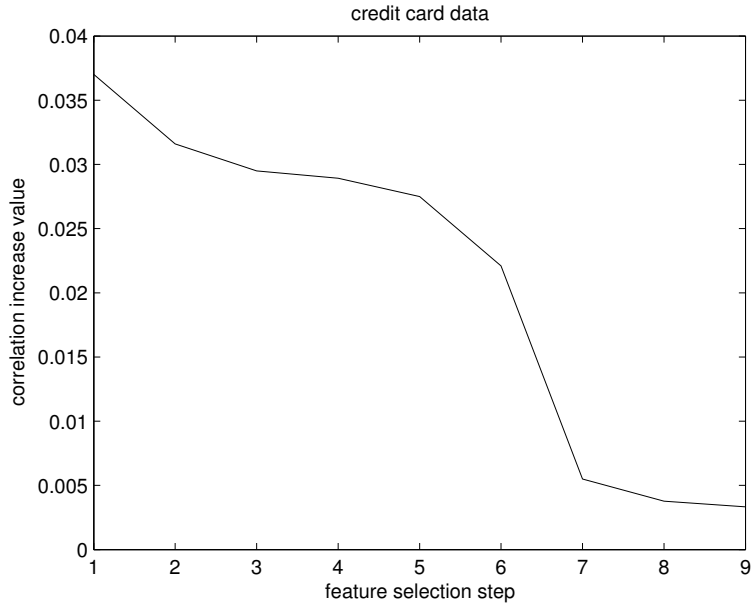


Figure 4. The increase in correlation value of breast cancer data

Data Source	Number of Selected Features	Accuracy			
		FSBEE	IG	Coefficient	Logistic Regression
Breast cancer diagnostic data	14	97.69	98.67	94.05	98.67
Smart phone record	178	89.47	91.32	91.06	90.75
Credit card record	10	77.46	78.68	78.11	77.43
Mesothelioma data	15	86.36	92.68	95.6	69.99
Image segmentation data	12	86.76	87.33	85.21	83.75

Table 2. Classification result comparison with supervised feature selection methods

Data Source	Number of Selected Features	Computation Time			
		FSBEE	IG	Coefficient	Logistic Regression
Breast cancer diagnostic data	14	0.156	0.016	0.015	1.228
Smart phone record	178	7455.375	4.563	4.422	6723.44
Credit card record	10	10.75	8.141	8.125	87.21
Mesothelioma data	15	0.406	0.313	0.297	4.563
Image segmentation data	12	0.047	0.012	0.01	0.53

Table 3. Computation time comparison with supervised feature selection methods

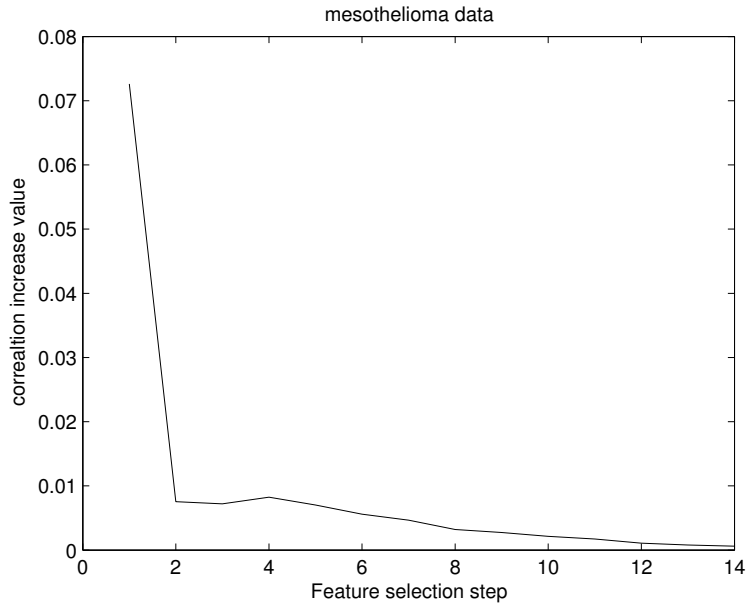


Figure 5. The increase in correlation value of breast cancer data

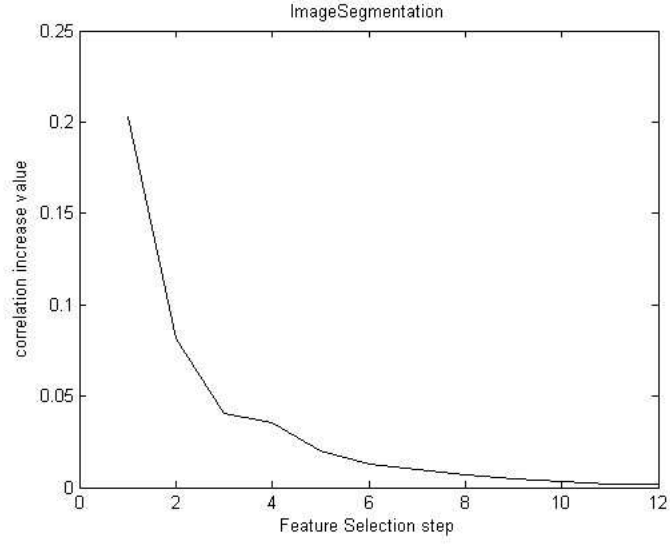


Figure 6. The increase of correlation value of image segmentation data

Data Source	Number of Selected Features	Accuracy			
		FSBEE	k-Mean Clustering Based Feature Selection	Variance Based Feature Selection	FSBMI
Breast cancer diagnostic data	14	97.69	96.84	94.39	100
Smart phone record	178	89.47	88.47	85.48	88.83
Credit card record	10	77.46	76.02	73.44	78.13
Mesothelioma data	15	86.36	89.61	84.65	76.95
Image segmentation data	12	87.76	80.21	78.43	81.98

Table 4. Classification result comparison with commonly unsupervised feature selection methods

Data Source	Number of Selected Features	ComputationTime			
		FSBEE	k-Mean Clustering Based Feature Selection	Variance Based Feature Selection	FSBMI
Breast cancer diagnostic data	14	0.156	0.24	0.031	0.781
Smart phone record	178	7455.375	154.33	4.359	9863.34
Credit card record	10	10.75	11.23	8.156	44.172
Mesothelioma data	15	0.406	0.43	0.297	0.797
Image segmentation data	12	0.047	0.049	0.125	0.14

Table 5. Computation time comparison with supervised feature selection methods

The method uses extended entropy to calculate the information loss between two features. It can measure the distance between features. During the feature selection procedure, the redundant information is considered. It assures that the selected features contain the maximum information. The advantages of the method can be summarized as three aspects. Firstly, extended entropy can simplify the calculation of information loss and improve computation speed markedly. Secondly, it can assure that the selected features contain the most information. Thirdly, it provides an objective rule to determine number of selected features. The experience results show that the proposed unsupervised method is efficient. It can be applied to many types of application areas.

Acknowledgements

This work is partially supported by the National Youth Science Foundation (No. 610-04115), Shandong Science and Technology Development Plan (No. 2016GGC01061), National Science Foundation (No. 61472230), National Youth Science Foundation (No. 61402271), the Natural Science Foundation of Shandong Province (Grant No. ZR2015JL023 and Grant No. ZR2015FL025).

REFERENCES

[1] Yu, L.—Liu, H.: Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution. Twentieth International Conference on Machine Learning (ICML '03), AAAI Press, 2003, pp. 856–863.

[2] Dash, M.—Liu, M.: Dimensionality Reduction. In: Liu, L., Özsu, M. T. (Eds.): Encyclopedia of Database Systems. Springer, 2009, pp. 843–846, doi: 10.1002/9780470050118.ecse112.

[3] Khalid, S.—Khalil, T.—Nasreen, S.: A Survey of Feature Selection and Feature Extraction Techniques in Machine Learning. 2014 Science and Information Conference (SAI), 2014, pp. 372–378, doi: 10.1109/sai.2014.6918213.

[4] Liu, X. M.—Tang, J. S.: Mass Classification in Mammograms Using Selected Geometry and Texture Features, and a New SVM-Based Feature Selection Method. IEEE Systems Journal, Vol. 8, 2014, No. 3, pp. 910–920, doi: 10.1109/jsyst.2013.2286539.

[5] Sarac, F.—Uslan, V.—Seker, H.—Bouridane, A.: Comparison of Unsupervised Feature Selection Methods for High-Dimensional Regression Problems in Prediction of Peptide Binding Affinity. 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2015, pp. 1–4, doi: 10.1109/embc.2015.7320291.

[6] Wang, D.—Nie, F. P.—Huang, H.: Feature Selection via Global Redundancy Minimization. IEEE Transactions on Knowledge and Data Engineering, Vol. 27, 2015, No. 10, pp. 2743–2755, doi: 10.1109/tkde.2015.2426703.

[7] Baccianella, S.—Esuli, A.—Sebastiani, F.: Feature Selection for Ordinal Text Classification. Neural Computation, Vol. 26, 2014, No. 3, pp. 557–591, doi: 10.1162/neco_a_00558.

[8] Sun, Z. Q.—Li, Z.: Data Intensive Parallel Feature Selection Method Study. 2014 International Joint Conference on Neural Networks (IJCNN), 2014, pp. 2256–2262, doi: 10.1109/ijcnn.2014.6889409.

[9] Sun, Z. Q.: Parallel Feature Selection Based on MapReduce. In: Wong, W. E., Zhu, T. (Eds.): Computer Engineering and Network. Springer, Cham, Lecture Notes in Electrical Engineering, Vol. 277, 2013, pp. 299–306, doi: 10.1007/978-3-319-01766-2_35.

[10] Hou, C. P.—Nie, F. P.—Li, X. L.—Yi, D. Y.—Wu, Y.: Joint Embedding Learning and Sparse Regression: A Framework for Unsupervised Feature Selec-

tion. IEEE Transactions on Cybernetics, Vol. 44, 2014, No. 6, pp. 793–804, doi: 10.1109/tcyb.2013.2272642.

[11] Aroquiaraj, I. L.—Thangavel, K.: Mammogram Image Feature Selection Using Unsupervised Tolerance Rough Set Relative Reduct Algorithm. 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering, 2013, pp. 479–484, doi: 10.1109/icprime.2013.6496718.

[12] Xu, J. L.—Zhou, Y. M.—Chen, L.—Xu, B. W.: An Unsupervised Feature Selection Approach Based on Mutual Information. Journal of Computer Research and Development, Vol. 49, 2012, No. 2, pp. 372–382, doi: 10.3724/sp.j.1087.2012.02250.

[13] Cai, D.—Zhang, C. Y.—He, X. F.: Unsupervised Feature Selection for Multi-Cluster Data. Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '10), 2010, pp. 333–342, doi: 10.1145/1835804.1835848.

[14] Witten, I. H.—Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. 2nd ed. Morgan Kaufmann, Amsterdam, 2005.

[15] Battiti, R.: Using Mutual Information for Selecting Features in Supervised Neural Net Learning. IEEE Transactions on Neural Networks, Vol. 5, 1994, No. 4, pp. 537–550, doi: 10.1109/72.298224.

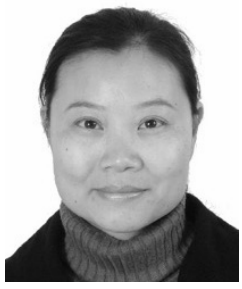
[16] Chiappino, S.—Marcenaro, L.—Regazzoni, C. S.: Information Bottleneck-Based Relevant Knowledge Representation in Large-Scale Video Surveillance Systems. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 4364–4368, doi: 10.1109/icassp.2014.6854426.

[17] UCI Machine Learning Repository. Center for Machine Learning and Intelligent Systems. http://archive.ics.uci.edu/ml/datasets.html.

[18] Anguita, D.—Ghio, A.—Oneto, L.—Parra, X.—Reyes-Ortiz, J. L.: A Public Domain Dataset for Human Activity Recognition Using Smartphones. 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2013, pp. 437–442.

[19] Fan, R.-E.—Chen, P.-H.—Lin, C.-J.: Working Set Selection Using Second Order Information for Training Support Vector Machines. Journal of Machine Learning Research, Vol. 6, 2005, pp. 1889–1918.

Zhanquan Sun, Ph.D., Associated Professor of University of Shanghai for Science and Technology. Major on big data, data mining and artificial intelligent. Has presided and attended 20 research projects and published about 60 academic papers.



Feng Li, Ph.D. candidate, Department of History, College of Liberal Arts, Shanghai University. Major on history data analysis. Attended about 10 research projects.

Huifeng Huang, Ph.D., Professor of Shandong Yingcai University. Major on image steganalysis, information security and data mining. Presided and attended 10 research projects and published about 30 papers.

Computing and Informatics, Vol. 38, 2019, 240–264, doi: 10.31577/cai_2019_1_240

FROM PARSED CORPORA TO SEMANTICALLY RELATED VERBS

Hasan Zafari

Department of Computer Engineering
Malayer Branch, Islamic Azad University
Malayer, Iran
e-mail: Hasan.Zafari@gmail.com

Maryam Hourali

Department of Information and Communication Technology (ICT)
Malek-Ashtar University of Technology
Tehran, Iran
e-mail: mhourali@mut.ac.ir

Abstract. A comprehensive repository of semantic relations between verbs is of great importance in supporting a large area of natural language applications. The aim of this paper is to automatically generate a repository of semantic relations between verb pairs using Distributional Memory (DM), a state-of-the-art framework for distributional semantics. The main idea of our method is to exploit relationships that are expressed through prepositions between a verbal and a nominal event in text to extract semantically related events. Then using these prepositions, we derive relation types including causal, temporal, comparison, and expansion. The result of our study leads to the construction of a resource for semantic relations, which consists of pairs of verbs associated with their probable arguments and significance scores based on our measures. Experimental evaluations show promising results on the task of extracting and categorising semantic relations between verbs.

Keywords: Semantically related verbs, temporal relations, cause-effect relations, related events knowledge base

1 INTRODUCTION

Understanding the semantics relations between events is an important step to capture the meanings of text. This information is crucial in supporting textual inferences, where systems need to automatically infer unknown fact from the currently available facts. For example, consider the following snippet. In Coreference Resolution, it may be useful to know *shot* could result in *killing*.

> An Indiana teenager killed a 73-year-old man.
> Autopsy results show Kim was shot three times.

Knowing that *kill* and *shot* are semantically related, we can easily co-refer *Kim* to *a 73-year-old man* rather *An Indiana teenager* which is not its correct antecedent.

A repository that includes relationship between plausible semantically related events could be helpful in many NLP tasks, including Question Answering [1, 2], Machine Translation, Information Extraction, Coreference Resolution [3], Prediction [4], Summarization [5], Recognizing Textual Entailment [6], etc.

Many semantic resources have been developed to express knowledge of verbs, including FrameNet [7], VerbNet [8], ProBank [9], and WordNet [10]. Despite usefulness in many aspects, unfortunately these resources do not provide (large-scale) semantic knowledge between verb pairs. WordNet [10] provides some types of this knowledge as cause and entailment relations. However this information is just provided for relations that are always true. For instance, it does not include the relation "shot" happens-before "kill" since it is just a plausible sequence of events and is not guaranteed to occur all the time. Such relations hold between a wide number of event pairs but are not accessible easily.

Since a real application demands a wide-coverage resource for related verbs, we develop an automatic method to acquire a broad-coverage repository of semantic relations between verbs based on Distributional Memory (DM) [11]. The main contributions of our research are as follows:

- Providing a broad coverage Knowledge Base (KB) of semantically related verbs.
- Proposing a set of novel metrics to measure the strength of the semantically related verbs.
- Using preposition between verb-noun pairs to infer semantic relations and direction between them.
- Providing plausible common arguments of related verbs that beside other benefits, is helpful in identifying correct sense of verbs that causes their relations.
- Role mapping for the common argument of each related verb pair.

This paper is organized as follows. Section 2 describes previous attempts to discover related verbs. In Section 3 we present Distributional Memory, which is the base of our method. The model for the extraction of semantically related verbs and classification relations types is presented in Section 4, followed by the evaluation and discussion in Section 5 and conclusion in Section 6.

2 RELATED WORK

Due to importance of verb knowledge in natural language processing, many semantic resources have been developed to express knowledge of verbs, including FrameNet [7], VerbNet [8], ProBank [9], and WordNet [10]. Despite usefulness in many aspects, unfortunately these resources do not provide *broad-coverage* semantic knowledge between verb pairs, but provide information about the semantic classes, thematic roles and selectional restrictions of verbs. Among these, WordNet and FrameNet are the only resources which provide information about semantic relation between verbs. However, as these resources are created manually they have a very limited coverage. Researchers have recently shown more interest in the task of automatic recognition of causal-temporal relations between events [12, 13, 14, 15, 16, 17, 18, 19, 20, 21].

In [12] the authors use Naive Bayes classifier to learn the probabilities of semantic relation between event pair from a raw corpus in an unsupervised manner. To evaluate their model, they used two test sets from different domains. Test sets were manually classified with two human annotators. They stated that their best model improved by 7.05 % from the baseline model.

In [22, 23] the authors tried to extract chains of events sharing a common participant. They consider only verbs as events and given an existing chain of events, they predict the next likely event involving the protagonist. They used narrative cloze to evaluate event relatedness, and an order coherence task to evaluate narrative order and reported improvement in both tasks.

In [13], a pattern-based approach is introduced which firstly extracts highly associated verb pairs and their frequency from the web. Then, using co-occurrence data on pairs of verbs, they assessed the strength of the associations by evaluating their mutual information. Finally, using a manually defined threshold they determine whether each association between a verb pair is valid or not. The result is a knowledge base of causal associations of verbs, which contains similarity, strength, antonym, enablement and temporal relations. They did not provide precise evaluation methodology for the obtained results.

Extracting verb-verb, verb-noun and noun-noun event relationship from text [14] concentrated on acquiring causality between events. They used both minimal supervision and unsupervised metrics to learn causal dependencies between two events. They evaluated their model on 20 news articles from CNN. On verbal events, they reported 38.3 % F-score with CEA and 1–2 % improvement using minimally supervised method.

In [21], the authors used Decision Trees for the detection of causations in sentences that contained causal relations. They reported the result of evaluation with precision of 98 % and recall of 84 %, but, their method was not able to detect the causes and the effects.

Using a set of knowledge-rich metrics [17] proposed to learn the likelihood of causal relations between intra- and inter-sentential instances of verb-verb pairs. They relied on the unambiguous discourse markers *because* and *but* to automati-

cally collect training instances of cause and non-cause event pairs, respectively. The result was a knowledge base of causal associations of verbs, which contained three classes of verb pairs: strongly causal, ambiguous and strongly non-causal.

In [18] the authors propose a model for the recognition of causality in intra-sentential verb-noun pairs using Supervised Classifier. They employed linguistic features along with semantic classes of nouns and verbs with high tendency to encode cause or non-cause relations. They generated a test set with instances of form verb-noun phrase and report 46.61 % F-score, and 80.74 accuracy.

In the most recent work, [20] tried to find pairs of verbs linked by a relation explicitly marked by a discourse connector in the corpus, as an indication of a regular semantic relation between the two verbs. The output of this work is the main existing resource that we have compared our results with.

3 DISTRIBUTIONAL MEMORY

Distributional Memory (DM) [11] is a generalized framework for distributional semantics, generalizing different existing typologies of semantic spaces. The aim of Distributional Memory is representing corpus-extracted distributional facts as weighted tuple structures, which are a set of weighted word-link-word tuples $\langle\langle$w1, L, w2$\rangle, \lambda\rangle$. W1 and w2 are a set of strings representing content words, and L is a set of strings representing syntagmatic co-occurrence links between words in a text. Each tuple T has a weight, a real-valued score, assigned by a scoring function $\lambda : W1 \times L \times W2 \rightarrow R$. For example, the tuple $\langle\langle$harvest, before, rain$\rangle$, 66.0141$\rangle$ says *harvest* and *rain* are related through the link *before* with the co-occurrence weight of 66.0141 in the corpus.

DM is built upon the DSM idea. Distributional semantic models (DSMs) are corpus-based models of semantic representation, rely on some version of the distributional hypothesis [24], stating that the degree of semantic similarity between two words (or other linguistic units) can be modelled as a function of the degree of overlap among their linguistic contexts.

Therefore, given a weighted tuple structure, by matricizing the corresponding labelled third-order tensor, four distinct semantic vector spaces can be obtained: $W1 \times LW2$, $W1W2 \times L$, $W1L \times W2$, and $L \times W1W2$. Depending on the tasks, one can choose suitable vector spaces to address it. For instance, one can use $W1 \times LW2$ to tackle attributional similarity tasks such as synonym detection or concept categorization. The $W1W2 \times L$ vectors represent word pairs in a space whose dimensions are links, and can be used to measure relational similarity among different pairs (e.g. $\langle$sergeant, gun$\rangle$ is similar to $\langle$teacher, pen$\rangle$). The $W1L \times W2$ space can be used to capture different verb classes based on the argument alternations they display (e.g. the object slot of kill is more similar to the subject slot of die). The $L \times W1W2$ space displays similarities among links.

Different DM models can be generated based on the selection of the sets W and L and of the scoring function λ. In this paper we used TypeDM, which is

the best performing DM model across the various semantic tasks addressed in [11]. The links of TypeDM include lexico-syntactic shallow patterns and, lexicalized dependency paths. Its tensor contains about 130 M non-zero tuples extracted from a corpus of about 2.83 billion tokens. This corpus has been obtained by concatenation of the Web-derived ukWaC corpus, about 1.915 billion tokens, a mid-2009 dump of the English Wikipedia, about 820 million tokens, the British National Corpus, about 95 million tokens. The resulting concatenated corpus was tokenized, POS-tagged and lemmatized with the TreeTagger and dependency-parsed with the MaltParser [30].

The model contains 30 693 lemmas (20 410 nouns, 5 026 verbs and 5 257 adjectives). These terms were selected based on their frequency in the corpus (they are approximately the top 20 000 most frequent nouns and top 5 000 most frequent verbs and adjectives), augmenting the list with lemmas that could be found in various standard test sets, such as the TOEFL and SAT lists.

4 PROPOSED APPROACH

In this section, we introduce our approach to extract semantically related verbs from TypeDM. Figure 1 depicts the structure of our proposed system. Firstly, candidate tuples are extracted from TypeDM. We assume that a verb-noun pair can be a candidate tuple if they are connected through a preposition. Next, tuples that do not contain event pairs are deleted, including

1. tuples containing phrasal verbs,
2. tuples whose w2 are non-action nominals, and
3. tuples whose w2 distinguished as non-action after disambiguation it based on w1 and *link* as context.

Then, after converting action nominals to their corresponding verbs, and aggregating verb pairs, some metrics of relations strength are introduced. Then using *subject* and *object* links in TypeDM, common arguments of semantically related verbs are extracted, which beside common argument weight (CAW), a measure of relation strength, can help to find mapping of verb pairs thematic roles. Finally, we derive the relation direction and relation types including causal, temporal, comparison, and expansion from links connecting two verbs. The following sections describe each of these steps in more details.

4.1 Extraction of Potential Relations

As explained in previous section, the TypeDM tensor contains about 130 M tuples automatically extracted from corpora of about 2.83 billion tokens. In order to get initial tuples that could denote pairs of related events, we have firstly selected 24 links from 25 336 direct and inverse link types formed by syntactic dependencies and patterns. These links are composed of 22 prepositions plus coordination and its inverse

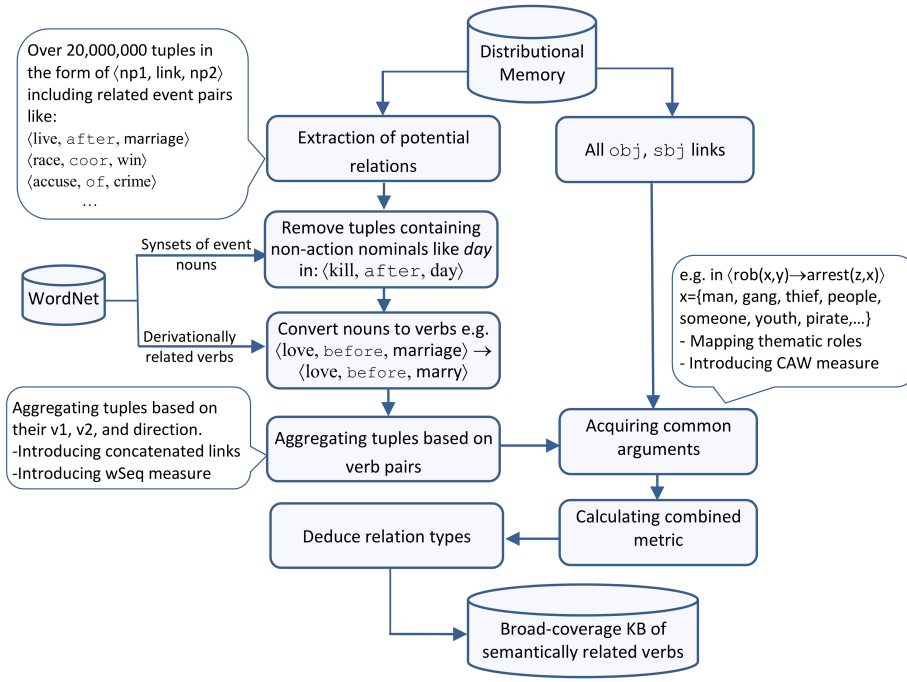


Figure 1. Our proposed system

direction. We extracted all tuples of these 24 links from TypeDM as initial tuples (InitTuples). InitTuples include about 23 M tuples in form of $\langle\langle \mathrm{w1}, \mathrm{L}, \mathrm{w2}\rangle, \lambda\rangle$. In these tuples w1 is mostly a verb (except coordination link) and w2 is always a noun. Table 1 shows these links along with example tuples. For instance, in $\langle \mathrm{accuse}, \mathrm{of}, \mathrm{murder}\rangle$, the preposition *of* is a sign of semantic relation between *accuse* and *murder*.

4.1.1 Relation Direction

Semantic relation is an asymmetric relation, so we have to know temporal direction of the relation. That is, in the tuple $\langle \mathrm{w1}, \mathrm{link}, \mathrm{w2}\rangle$ we have to know if the relation direction is from w1 to w2 (w1 $\rightarrow$ w2) or from w2 to w1 (w2 $\rightarrow$ w1). In the link set we are working on, the direction of most of links is w2 $\rightarrow$ w1, i.e. w2 happens before w1. The direction of some links, however, is w1 $\rightarrow$ w2. Precisely, the direction of *before* and *coord* are always w1 $\rightarrow$ w2. The direction of all other links except three ambiguous-direction links viz. *for, with*, and *without* is always w2 $\rightarrow$ w1.

We plan to give a solution for finding the relation direction of these three ambiguous links in future. However, at the moment, we simplified the problem and

link	Example tuple	link	Example tuple
after	⟨divorce, after, marriage⟩	since	⟨revise, since, publish⟩
at	⟨win, at, match⟩	through	⟨gain, through, study⟩
because	⟨suffer, because, illness⟩	under	⟨purchase, under, agreement⟩
before	⟨defrost, before, cooking⟩	until	⟨teach, until, retirement⟩
by	⟨learn, by, experiment⟩	upon	⟨renew, upon, expiration⟩
despite	⟨fail, despite, effort⟩	via	⟨melt, via, heating⟩
during	⟨kill, during, raid⟩	while	⟨suffocate, while, feeding⟩
for	⟨marry, for, love⟩	whilst	⟨suspend, whilst, investigation⟩
from	⟨absolve, from, blame⟩	with	⟨charge, with, murder⟩
of	⟨accuse, of, murder⟩	without	⟨capture, without, fight⟩
on	⟨attract, on, offer⟩	coordination	⟨fight, coord , die⟩
over	⟨argue, over, deal⟩	coordination!	⟨discount, coord-1, price⟩

Table 1. List of links used to extract potentially related event pairs

adapted the majority of directions as the relation direction for these ambiguous links. According to our experiments, more than 95 % of tuples of *with (without)* links have the direction of w2 → w1, so we supposed all tuples in these links (with-without) have direction from w2 to w1. For the link *for*, about 83 % of tuples have the direction of w1 → w2, so we considered its direction as w1 → w2. It should be noted that since each verb pair is usually connected through multiple links (see Section 4.3), their relation direction is introduced by multiple links. Hence the existing error in the direction of *for* and *with* has negligible effect on the direction of final relation.

4.2 Removing Non-Action Nominals

Having extracted InitTuples from TypeDM, the next step is to remove the tuples from it which do not contain event pairs. In natural language, an event is mostly encoded using a verb or a noun. In all tuples $\langle\langle \mathrm{w1}, \mathrm{link}, \mathrm{w2}\rangle, \lambda\rangle$ extracted in previous subsection, w2 is a noun. Obviously, not all these nouns are events or action nominals. Following [25], action nominals are defined as "nouns derived from verbs (verbal nouns) with the general meaning of an action or a process". Also, according to [26], "an event is a situation that occur or happen, and can be expressed by verbs, nominal or some other linguistic units". So, we have to identify action nominals (event nouns) from non-action ones in InitTuples. We have intended to remove three types of tuples that do not contain event pairs, including:

- Tuples where w1 together with a preposition create phrasal verbs like *account for*.
- Tuples where based on WordNet event denoting synsets (WEDS) w2 is not event at all, like *day*.
- Tuples where w2 becomes non-event after disambiguating them based on w1 and preposition. For example *race* is not an event noun in ⟨discriminate, because, race⟩.

After removing these non-event pairs, the number of tuples in InitTuples reduced from over 23 M to about 3.2 M tuples.

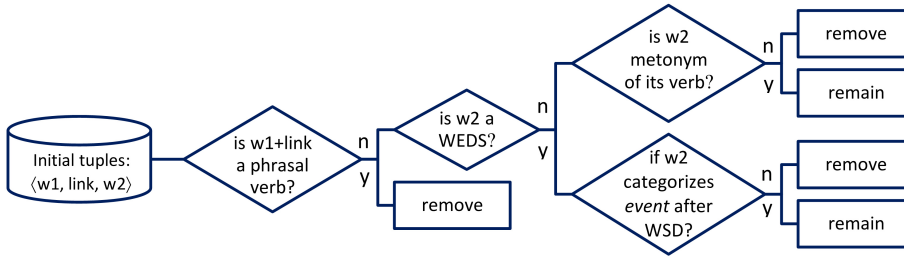


Figure 2. The flowchart of removing tuples containing non-action nominal

4.2.1 Removing Phrasal Verbs

In this article we are interested in prepositions connecting a verb to a noun. On the other hand, in English, prepositions could be combined with verbs producing phrasal verbs like *abide by, accord with, account for, look after*, and so on. This usage of prepositions differs from the one we based our method on. If we let tuples containing phrasal verbs remain in our data, they will produce noise in subsequent steps causing wrong results in relations types' classification and determining relations strength. So, we used a predefined list of phrasal verbs to remove such tuples from InitTuples.

4.2.2 Non-Event Nominals

The term *event* itself has many readings. Some authors use it to refer only to dynamic actions, while others use it to refer also to static situations [42]. In the recent work [38] some definitions of event are provided. The best-known classification of events is one proposed by [41], who distinguishes between *states* (non-dynamic situations persisting over a period of time and without an endpoint, e.g., believe), *activities* (open-ended dynamic processes, e.g., walk), *accomplishments* (processes with a natural endpoint and an intrinsic duration, e.g., build a house) and *achievements* (almost instantaneous events with an endpoint, e.g., find). Moreover, in the linguistic literature, all types of actions, states and processes often fall under the cover term *eventualities*, coined by [32] in his work on the algebra of events. Following Bach's broad notion of event [32], TimeML identifies a wide range of linguistic expressions realizing events, i.e. tensed and untensed verbs (e.g., was captured, to thank), adjectives (e.g., sick), nominals (e.g., strike) and prepositional phrases (e.g., on board).

Investigating various classifications of the event in past literature, we summarize them in Table 2. Respecting these classifications, we can see that three classes exist in almost all classifications viz. state (non-dynamic situations persisting over

Work(authors, year)	Event types	Reference
De Swart, 1998	states, activities, events	[41]
Vendler, 1967	states, activities, event (accomplishments, achievements)	[43]
Bach, 1986	state, process, event (protracted, happenings, culminations)	[34]
Dölling, 2014	point, state, process, event (episode, changes)	[35]
Moens et al, 1988	states, events (culmination, culminated process, point, process)	[36]
de Swart et al, 1999	states, activities, events	[42]
Pustejovsky et al, 2003	state (state, i-state), event (occurrence, aspectual, perception, i-action), reporting	[37]
Pustejovsky, 1991	states, processes, transitions	[38]

Table 2. Event classification in past literature

a period of time and without an endpoint), process (or activity, open-ended dynamic processes), and event (or transition, natural endpoint that are quantized, telic or terminative). Considering the TimeML [35], another class could be added to them i.e. reporting.

On the other hand, like [14, 37] we used WordNet to identify some sub-trees from WordNet synsets so that their hyponym (children) are mostly action nominals. Surprisingly, the top four WordNet synsets which we have gained with highest ratio of event nominals are analogous to the four classes that we acquired from the previous literature. These synsets along with some of their hyponym (children) nouns are shown in Table 3. Indeed, not all nouns under these synsets are action nominals. However, as our first goal is identifying and discarding tuples containing non-event nouns, this method works well at present.

Synsets	Sample event nouns
event	work, revolution, death, service, act, birth, flight, game, development, war, crime
process	pressure, loss, review, implication, access, growth, investigation, consideration, assessment
state	operation, fear, interest, need, love, injury, marriage, press, absence, independence
message	comment, note, notice, agreement, request, advice, statement, claim, regulation, instruction
symptom	pain, suffering, scar, founder, fever, congestion, burn, inflammation, tenderness, rash, cough

Table 3. WordNet event denoting synsets and their equivalent classes in the past literature

Given WEDS, we can determine if a noun have the chance of being event noun or not. For instance, the leaf-to-root paths for the first sense of offense(n) and album(n) are as follow, respectively:

- offense ⇒ behavior ⇒ activity ⇒ act ⇒ **event** ⇒ psychological feature ⇒ abstraction ⇒ entity,
- album ⇒ medium ⇒ instrumentality ⇒ artifact ⇒ whole ⇒ object ⇒ physical entity ⇒ entity.

The *event* synset in leaf-to-root hypernym path of the first sense of *offense* indicates that this noun could be an action nominal. For the word *album*, on the other hand, there is not such synset, neither in its first sense nor any other senses.

This implies that *album* could not denote an event. Using this method, we determine the sense number of the nouns that can denote event together with the synsets name. We call this information semantic-category. In order to determine semantic-category for a noun, the leaf-to-root hypernym path for its all senses is searched. That is, we have gone through all its WordNet senses; have examined their hypernyms (parent) in WordNet hypernym relations one-by-one upward. During the search, for each sense S of the noun, if one of the WEDS synset is found, we assigned the synset name along with the sense number of S, otherwise its value will be non-action.

For example *event/4* for semantic-category of a noun means that hypernym path of its fourth sense contains *event*, and *process/1* means that hypernym path of its first sense contains *process*, and so forth. For example semantic-category for *birth* and *authority* are:

- semantic-Cat (birth) = $\langle$event/2 − process/3$\rangle$,
- semantic-Cat (authority) = $\langle$state/4$\rangle$.

Algorithm 1 shows the details for extracting semantic-cat for a word. Firstly, the semantic cat is set to an empty string. Then, by iterating through all noun senses of the word and checking their hypernym against event denoting synsets, semantic-cat is acquired.

Algorithm 1 Semantic-cat extracting algorithm

```
Input: word
Output: semantic categories
semantCat ← empty
nounSenses ← all noun senses (word)
  for S∈ nounSenses do
    HoS ← all hypernyms of S
      for h∈ HoS do
        If h is in {event, process, state, message, symptom} then
          append(semantCat, h+"/"+sense_number(S))
        end if
      end for
  end for
If semantCat is empty then
  semantCat ← non-event
end if
return semantCat
```

4.2.3 Metonym Nouns

Beside action nominals that can be identified through Algorithm 1, there are other nouns that are of our interest. Although semantically can denote event, these

nouns are categorized as non-event nouns by Algorithm 1. Considering the tuple ⟨escape, from, jail⟩ for example, we can understand that event *jail* can result in event *escape*. In fact, the noun *jail* in this tuple can denote event *jail* (putting in jail). However, *jail* is a noun in this tuple and semantic-Cat (jail) is non-event. The point is that the word *jail* has a verb form as well, which is an event. Actually, here the noun *jail* can be metonym of its verb in our method. Another such example is ⟨receive, after, pay⟩, where noun *pay* denotes event of paying, while it is categorized as non-event nouns by Algorithm 1 as well. There are many such nouns in TypeDM which are categorized as non-event while could denote an event. In order to detect these metonym nouns, we have heuristically chosen nouns having two following criteria:

- the noun has a verb form with the same spelling,
- the noun categorizes as non-event based on Algorithm 1.

We found about 800 such nouns in TypeDM through above-mentioned criteria. Table 4 shows some examples of such nouns extracted from TypeDM.

w1	link	w2
remove	through	filter
cut	with	saw
deduct	under	pay
grow	because	feed
run	on	steam
increase	despite	drop
access	via	join
defrost	before	cook
escape	by	hide
win	after	elect
charge	for	fuel

Table 4. Some examples of tuples where w2 is a noun that can be metonym of its verb

4.2.4 Disambiguate Polysemous Words

Many polysemous words in English can have both event and non-event meanings. For instance the word *spring* can be both non-event noun (springtime, fountain, a metal elastic device, and elasticity) and event noun (leap). We have to identify and separate non-event nouns like *spring* in ⟨occur, during, spring⟩ or *race* in ⟨discriminate, because, race⟩ to prevent probable harmful side-effects in subsequent steps. We tried to convert such triples to a sentence and use state of the art WSD like BabelNet [27] to find correct sense of the ambiguous noun. Regarding this approach there are two points. Firstly, it is not easy to convert every tuple to a well formed English sentence to fit input of WSD like BabelNet. Secondly, the precision of this approach is very low in our data. For example, BabelNet disambiguate *race* in tuple ⟨discriminate, because, race⟩ as *any competition* when we tried it in the sentence *"People should not be discriminated because race"*. So we decided to build a model for disambiguating w2 in our tuples.

To do so, we have to construct a model which, given a tuple ⟨w1, preposition, w2⟩, will correctly predict the sense to which the w2 belongs using w1 and preposition as context. This is a classification problem, which for a w2 with N noun senses, has N different classes. For each of these N senses, we extracted salient words from the WordNet glosses, synonyms, and hypernyms as feature set. Also, in order to convert the context (i.e. the w1 and the preposition) to a set of feature words, we used TypeDM. Specifically, for a tuple ⟨w1, preposition, w2⟩ in which w2 is an ambiguous noun, we extracted all tuples $\langle\langle \text{w1}, \text{L}, \text{w2}\rangle, \lambda\rangle$ having the pattern ⟨w1, preposition, *⟩ from TypeDM. Then, we have chosen top 5 w2 of the extracted tuples having highest λ value. For example, in tuple ⟨occur, during, spring⟩ it is unknown for the system if the *spring* means a season, outflow, a metal, or leap. After extracting top nouns for the pattern ⟨occur, during, *⟩ we came up with following nouns: *season, phase, summer, winter, stage*. Using these nouns as context, and comparing similarity between it and the features of each senses of the noun *spring*, we got spring#1 as most similar sense, which means the season of growth. Applying this idea on tuple ⟨pump, from, spring⟩ gives following nouns as context: *mine, pit, stream, station*, and *source*. After calculating similarities between all *spring-n* senses and the context words, we got the spring#2 as the result which means a natural flow of ground water. We evaluated this WSD method on 200 tuples of InitTuples containing ambiguous nouns as their w2 as test data. The correct senses of ambiguous nouns in these tuples were identified by two human annotators. We have achieved a 0.78 kappa score for the human inter-annotator agreement. Evaluating the WSD on this test data yields the accuracy of 74%.

4.2.5 Mapping Semantic-Cat To a Real Number

In the Subsection 4.2.2, we used semantic-cat to remove tuples containing non-action nominals. However, it can also be used to rank action nominals, based on how likely they can refer to an event. In other words, nouns under some WEDS synsets refer to event more often than some others. For example synsets like *event, process* are more event-denoting than *state*. Additionally, sense numbers of the nouns that belong to these synsets are important as well, e.g., *event/1* is more probable to denote an event than *event/2*, and it is relative to *event/3*, and so on. In order to capture action denoting strength of any action nominals, we converted semantic-cat to a real numbers called *catVal*. To obtain the value of catVal we used two metrics:

1. the ratio of the nouns belong to the synset that denote event,
2. the sense number of the noun that denotes event.

The less the sense number is, the higher is the catVal value. Algorithm 1 (italic lines) shows details of this calculation. CatVal will be used in ranking step in Section 4.5.

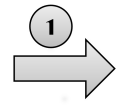
4.3 Aggregating Tuples Based on Verb Pairs and Direction

As explained in Section 3, there is a co-occurrence frequency of the tuple (λ) to characterize its statistical salience; however, it is not accurate enough to determine the real strength of the semantic relations solely. On the other hand, a pair of events may be related by different links in different tuples. So, we decided to aggregate the tuples based on their event pairs and the relation direction. The event pairs in *InitTuples* are now in the form of verb-noun pairs. Although some of these nouns may denote the same event, however, they may have different derivational forms like *graduation, graduating, graduate*, etc. So, we decided to convert w2 nouns to their corresponding verbs to get verb-verb pairs. In addition to solving the problem of tuple aggregation, this conversion is also necessary to find common arguments in the next section. We used derivationally related form API of WordNet to convert nouns to their corresponding verb(s), see Figure 3 b).

Having verb-verb pairs in the tuples, we now can aggregate the tuples based on their verb pairs and the relation direction, summing the co-occurrence frequency and concatenating the links for them. This way, verb pairs that are related through different links (in different tuples) will be connected through concatenation of that links (converted to just one tuple). This grouping process reduces the number of tuples (distinct verb pairs) in our KB to about 1.5 M tuples. Doing so, the weight of each verb pair is now the sum of the weights of all tuples that have been grouped to create that pair. This new sum is more accurate to capture the strength of semantic relationship between two verbs. We call this new metric *wSequence* or *wSeq* for short. Figure 3 shows this process for the verb pair *admit-graduate*.

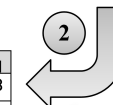
(a) Verb-noun tuples for the pair admit-graduate.

W1	link	W2	dir	λ
admit	upon	graduation	←	24.98
admit	after	graduation	←	21.00
admit	coord	graduation	→	19.49
admit	after	graduating	←	7.01
admit	with	graduating	←	4.88
admit	for	graduate	→	2.15
admit	of	graduate	←	1.41



(b) Verb-verb tuples for the pair admit-graduate.

W1	link	W2	dir	λ
admit	after	graduate	←	28.01
admit	upon	graduate	←	24.98
admit	coord	graduate	→	19.49
admit	with	graduate	←	4.88
admit	for	graduate	→	2.15
admit	of	graduate	←	1.41



(c) Aggregate the tuples based on verb pair and the direction for the pair admit-graduate.

W1	concatenated links	W2	dir	wSeq
admit	after (28.01), upon (24.98), with (4.88), of (1.41)	graduate	←	59.28
admit	coord (19.49), for (2.15)	graduate	→	21.64

Figure 3. a) Initial tuples of pair admit-graduate, b) convert nouns to their corresponding verbs, c) aggregate the tuples based on verb pairs and the direction

4.4 Roles of Common Arguments

Considering the fact that semantically related verbs should have common arguments, we believe that the more two verbs are semantically related, the more words they will have as their common arguments (subject or object). For instance, *plant* and *harvest* which are semantically related have many words that can be their common arguments, but *plant* and *crash*, which are not semantically related, have almost no word as their common arguments:

- Argument (plant) $\cap$ Argument (harvest) = {crop, plant, grape, seed, potato, grain, fruit, corn, wheat},
- Argument (plant) $\cap$ Argument (crash) = $\emptyset$.

We call these words that can be arguments of both verbs *common arguments*. Common arguments can be found in subject and object links in TypeDM. There are more than 10 M such links in TypeDM. We define Common Argument Weight (CAW) as the relative measure of the strength between two verbs. To acquire this value for two given verbs *verb1* and *verb2*, we have firstly chosen subject and object links (tuples) of TypeDM for them, namely V1Links and V2Links, respectively. There are about a few thousands such links for each verb in TypeDM. Then we joined the tuples of V1Links and V2Links based on their common arguments to get joint tuples *jointTuples*. That is, for each tuple of V1Links $\times$ V2Links if V1Links.arg equals V2Links.arg, we keep (join) them, otherwise discard them. Then, we calculate $f(\lambda 1, \lambda 2)$ as a function of $\lambda 1$ and $\lambda 2$ of the joint tuple, where $\lambda 1$ is the weight of verb1 tuple and $\lambda 2$ is the weight of verb2 tuple. Lastly, by sorting the joint tuples based on $f(\lambda 1, \lambda 2)$ and picking the highest one, CAW can be calculated as a function of $\lambda 1, \lambda 1$ of this tuple. See Algorithm 2 for more details.

Algorithm 2 has three outputs, CAW, common arguments and role mapping. Common arguments can be acquired by selecting common arguments of top n tuples from sortedTuples, sorted tuples of jointTuples based on $f(\lambda 1, \lambda 2)$. Role mapping maps the thematic roles of related verbs (e.g., the Agent of kill is mapped to the Patient of arrest). This is very useful information about semantically related verbs that can be used in many NLP applications, like Coreference Resolution. In order to get this mapping, we have heuristically chosen the rel1 and rel2 of the top 1 tuple from sortedTuples. Although this is only a heuristic, but in most cases it works properly. The rationale behind it is that the common argument that comes with both verbs most of the times has a certain role with each verb. Hence, choosing the top 1 tuple of the sortedTuples which has the highest value of λc is a simple and acceptable solution for this problem. The verb pair (escape, arrest), for instance, has nouns like *prisoner, criminal, man* as their common arguments which are usually subject of *escape* and object of *arrest*. So, for this verb pair, we obtain the mapping escape (sbj) = arrest (obj).

Although gathered from big parsed corpora and not necessarily co-occurring in the same document, the acquired common arguments are so accurate. Beside

Algorithm 2 Algorithm for extraction of CAW, Role Mapping, and Common Arguments

Input: verb1, verb2
Outputs: sbj, obj mapping, CAW, common arguments
V1Links ← all the links of TypeDM where w1= verb1 and link ∈ {sbj, obj}
V2Links ← all the links of TypeDM where w1= verb2 and link ∈ {sbj, obj}
    jointTuples ← empty
    for t1∈ V1Links **do**
      for t2∈ V2Links **do**
        if t1.arg=t2.arg **then**
          CA ← t1.arg
          λ1 ← t1.λ
          λ2 ← t2.λ
          rel1 ← t1.rel
          rel2 ← t2.rel
          λc ← f(λ1, λ2)
          jointTuples.add (CA, λ1, λ2, λc, rel1, rel2)
        end if
      end for
    end for
sortedTuples ← jointTuples.sortDescending(λc)
CAW ← $\pi_{\lambda c}(\sigma_{\text{top1}}$ sortedTuples)
RoleMapping ← $\pi_{\text{rel1, rel2}}(\sigma_{\text{top1}}$ sortedTuples)
commonArgs ← $\pi_{\text{CA}}(\sigma_{\text{top n}}$ sortedTuples)

a metric for relations strength measurement, common arguments can act as a mean to disambiguate polysemous verb with respect to another verb. For instance in (install, execute) common arguments are words denoting a program or script, which indicates *execute* means run a program, but in (arrest, execute) common arguments denote a prisoner or criminal, which indicates *execute* means put to death. Table 5 shows some examples of CAW, common arguments and role mapping.

Verb pair	Common arguments	Role mapping	CAW
design- print	page, poster, card, form, leaflet, book, work, logo, character, map, stamp, piece, line, part, cover, section, t-shirt, pattern	obj-obj	685.7961
sow- harvest	crop, seed, field, grain, plant, corn, wheat, bean, barley, onion, variety, vegetable, rice, flower, carrot	obj-obj	651.5708
rob-arrest	man, gang, thief, people, someone, youth, pirate, bandit, government, criminal, robber, soldier, guy, thug, group, woman, burglar, gunman	sbj-obj	258.8246
Try-succeed	government, man, company, party, student, team, child	sbj- sbj	396.6854
own-manage	Business, company, property, site, estate, asset, land, team, farm	obj-obj	1227.6482

Table 5. Some examples of CAW, common arguments and role mapping

4.4.1 Calculating $f(\lambda 1, \lambda 2)$

In above subsection we expressed $f(\lambda 1, \lambda 2)$ as a function that combines $\lambda 1$ and $\lambda 1$ as a single metric which we called λc. We decided to choose the minimum of $\lambda 1$ and $\lambda 2$ as $f(\lambda 1, \lambda 2)$, i.e. $f(\lambda 1, \lambda 2) = \text{minimum}(\lambda 1, \lambda 2)$. One may wonder why we have chosen minimum not maximum or multiplication of $\lambda 1$ and $\lambda 2$, for example. We opted for the minimum for two reasons. First, it is obvious that both of $\lambda 1$ and $\lambda 2$ are important in weighting the joint tuple of v1Link and v2Link, so we have to use a function of both values. Second, if we choose multiplication or maximum or average of $\lambda 1$ and $\lambda 2$, it may cause undesirable results, because the common argument may come with verb1 (verb2) more often than the other, resulting in a big value for $\lambda 1(\lambda 2)$. So if we multiply, sum, or choose the maximum value of $\lambda 1$ and $\lambda 2$, we will get a high value of CAW for a verb pair that may not agree with their common arguments.

4.5 Calculating Combined Metric

Since the beginning of Section 4, we introduced three metrics that can be used in ranking semantically related verbs based on the relations strength, i.e., catVal, wSeq, and CAW. In this section, after introducing a new metric, we plan to combine them to obtain combined metric.

In addition to the metrics introduced in previous subsections, we can use PMI (Pointwise Mutual Information). PMI (Equation (1)) is information-theory approach to measure the statistical association between two words. In our dataset, PMI estimates whether the co-occurrence of two verbs is higher than the a priori probability of them occurring independently. PMI defined as:

$$PMI(v1, v2) = \log\left(\frac{P(v1, v2)}{P(v1)P(v2)}\right). \tag{1}$$

The value of $P(v1, v2)$ can be obtained from the of co-occurrence weight of two words acquired from the corpus (Section 3). For calculating the value of $P(v1)$, the sum of λ in all tuples T where $v1 \in T$ is computed. $P(v2)$ is calculated in a similar way.

Now there are four metrics which can be used to rank tuples of verb pairs based on their relation strength. In order to create the combined ranking formula based on these metrics, we ranked 150 verb pairs manually and used them as train data of a linear regression model.

4.6 Deduce Relation Types

To classify semantic relations, following Penn Discourse Treebank (PDTB) [28], we grouped discourse relations into four classes: causal relations (Contingency), temporal relations (Precedence, Succession), comparison relations (Contrast), and expansion relations (Conjunction).

Contingency is used when the connective indicates that one of the events causally influences the other.

Temporal is used when the connective indicates that the situations described in the arguments are related temporally.

Comparison applies when the connective indicates that the relation highlight prominent differences between the two situations.

Expansion covers those relations which expand the discourse by providing additional information or illustrating alternative situations.

To acquire relation types between verb pairs, we used their connecting links. As explained in previous subsections, there are 24 links connecting verb pairs in our KB (see Table 1). After aggregating tuples based on verb pairs and the relation direction, every verb pair is connected through a subset of these links (Figure 3). Now the problem is to infer relation type(s) from the connecting links. For each relation, there are some linguistic cues to infer it from Table 6 shows these cue links.

Our introduced cue words for each relation are in accordance with the abovementioned definitions of those relations. In temporal relation, for example, the definition says "the connective indicates that the situations described in the arguments are related temporally". Each of our introduced cue words for temporal relation is such indicative without no exception or ambiguity. The *after* and *before* prepositions denote *succession* and *precedence* relations, respectively, which are subtypes of the Asynchronous temporal relation introduced in PDTB. *Until* and *upon* denote succession relation as well. The prepositions *at, on, while, during, whilst*, and *over* denote Synchronous relations which is subtype temporal relation introduced in PDTB. Some of these prepositions could denote other meanings than those of temporal relation. For example, *at* and *on* could denote position or location, but as we removed non-event arguments for these prepositions in Section 4.2, they will just denote temporal relation in existing tuples. For the comparison (contrast) and expansion (Conjunction) relations, the selected cue words in Table 6 are in accordance with PDTB definitions for these relations.

Relation type	Cue word links
causal	through, by, via, with, of, for, from, because, since, under
temporal	after, at, on, over, before, until, upon, while, during, whilst
comparison	despite, without
expansion	coord , coord-l

Table 6. Cue word links for each relation type

For three relations, i.e. *temporal*, *comparison*, and *expansion* we can use a rule, based on their cue words to infer the existence of that relation between each verb pair. That is, these three relations can be identified by this simple rule: for each relation $R \in \{\text{temporal}, \text{comparison}, \text{expansion}\}$ if the words in connecting link be-

tween verbs $v1$ and $v2$ contain a cue word of the relation R, then the relation R holds between $v1$ and $v2$.

For the *cause*, on the other hand, we could not find any rule that works based on its introduced cue words. For instance, in the tuple ⟨charge, with, offend⟩ the preposition *with* is a sign of causal relation between *offend* and *charge*, but in ⟨answer, with, laugh⟩ there is no causal relation between *laugh* and *answer*. The problem is that these connectors are ambiguous in that they are associated with several relations. In addition, the amount of links contribution in relations (i.e. the co-occurrence weight of the tuple in corpus) is not 1, 0 modes (i.e. exist or not exist) but they can take a value ranging from small amount to several hundreds. We believe that the links values are also important in determining the relations. Hence, sometimes three cue words with relatively low co-occurrence value (through (10), by (5), via (2)) could denote the cause relation and sometimes just two cue words with high co-occurrence value (with (200), by (170)) is enough. Sometimes one cue word like *because* can solely be translated to the cause relation. These all indicate that generalizing the links to get a rule to translate links set to the *cause* relation is not easy. So we decided to use a learning method for this task.

In order to acquire training data, we manually collected instances of cause and non-cause event pairs from the KB. We labeled 500 cause and 500 non-cause verb pairs. We used the value of connecting links as input features to train a supervised model for classifying relation between verb pairs as cause or non-cause. We chose the Random Forest classifier implemented in Weka [29] to train the model which yielded the highest performance.

5 EVALUATION AND DISCUSSION

In this section, we present the evaluation of our semantically related verbs KB. Specifically we performed experiments to evaluate

(1) the direction between verbs pairs,
(2) the ranking of verb pairs based on their strength of association,
(3) the quality of the four categories relations between verb pairs in KB (i.e., causal, temporal, comparison, and expansion),
(4) the mapping between thematic roles of verb pairs.

For each experiment we created a test data from our KB and asked human annotators to annotate them. Also, for cause relation of case (3) which is most important semantic relation in our task, we compared the performance of our approach with knowledge bases that are extracted in similar way. This experiment is done against available data sets of causal relations that are explained in following subsection. We consulted the freely available resources VerbOcean [13] and V2R [20]. VerbOcean data contains 98 362 tuples including 58 330 distinct verb pairs. V2R contains over 8 000 000 tuples including 3 803 294 distinct verb pairs. It should be noted that the tuples in V2R contain some prefixes or affixes that could affect normal comparison.

For example many tuples contain a [not] or [state verb] prefix. We removed these affixes before comparison. We also removed its tuples which contained non-word tokens (e.g. numbers, quotation, exclamation mark).

5.1 Data Sets

5.1.1 Available Data

This section presents details of freely available data for cause-effect relations. The details of this data set are explained below.

SemEval-2: SemEval-2 Task 8 focuses on multi-way classification of semantic relations between pairs of nominals. One of these relations is Cause-Effect (CE). In the original dataset in each sentence one causal pair has been annotated. We have extracted these pairs. Because the events in VerbOcean, V2R and our KB are expressed through verb pairs, we converted event pairs extracted from SemEval to their corresponding verbs (if possible). This way we obtained 451 Cause-Effect verb pairs.

WordNet cause relations: WordNet contains causal relations between verb pairs. Extracting these relations from WordNet we obtained 743 cause verb pairs.

ECED: in [14] the authors have annotated some causal relations from news documents and used the data in developing and evaluation of their method. We extracted causal event pairs from these annotated data and converted event nouns to their corresponding verbs. This way we get 400 cause-effect verb pairs. We called these data ECED.

As the first experiment, we compared our method with V2R and VerbOcean against above data. This experiment tests the coverage of causal relation along with the direction of relation in our KB. Figure 4 and Table 7 show the results.

The better coverage of our result in comparison with V2R becomes more valuable when taking this point into account that the total number of **distinct** verb pairs in our KB is far lesser that of in V2R (0.9 M vs. 3.8 M). This means that besides the coverage, the precision of our method is much higher.

Data	Methods	Total relations	Included verb pairs	recall
WordNet cause relation	Our method	743	**256**	**35.45%**
	V2R	743	218	29.34%
	VerbOcean	743	16	2.15%
Do et al	Our method	400	**278**	**69.50%**
	V2R	400	232	58.00%
	VerbOcean	400	29	7.25%
SemEval-2010	Our method	451	**114**	**25.27%**
	V2R	451	82	18.18%
	VerbOcean	451	6	1.33%

Table 7. Coverage of cause relation in our KB, V2R, and VerbOcean with respect to ECED, WordNet cause relation and SemEval-2010

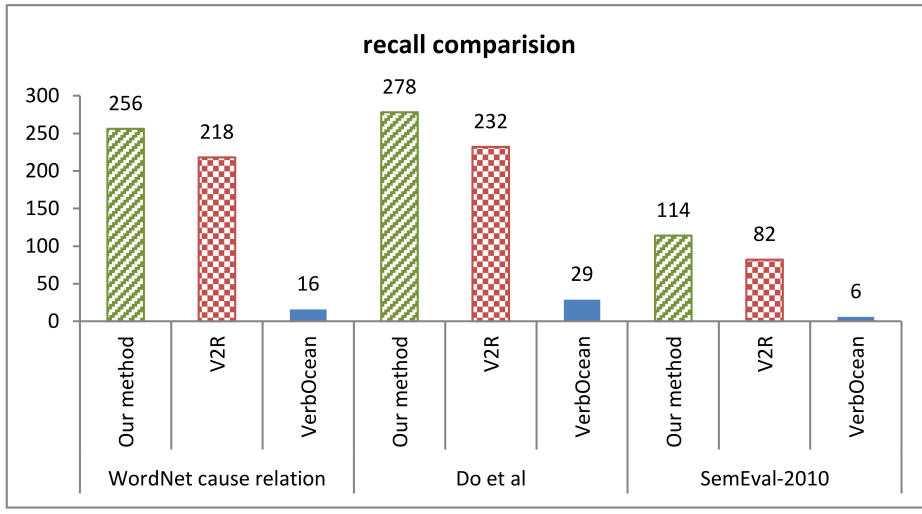


Figure 4. Recall comparison of our method against V2R and VerbOcean

5.1.2 Annotated Data from Our KB

We collected test data from our KB for experiments (1) to (4). These test sets were selected randomly with equal proportion of weak and strong relation strength. We asked two annotators to annotate these data. Then we tested our KB against these test data. The process is explained in more details below.

We selected 100 verb pairs from our KB randomly to create two test sets for experiments (1) and (4), respectively. Then we asked two annotators to identify the direction of relations between each pair (experiment 1) and find the mapping between thematic roles of the verb pairs (experiment 4). The kappa score for the human inter-annotator agreement achieved on Test-set1 (Test-set4) is 0.94 (0.51), respectively. Then we compared the direction of these tuples with that of our KB. The precision for relation direction and thematic role mapping was 91 % and 46 %, respectively.

For experiment (3), i.e. evaluating the quality of the four categories relations between verb pairs in KB, we created a test data from our KB. For this purpose, we selected 100 verb pairs for each of four categories randomly. These data were annotated by two human annotators to determine if the semantic relation holds between the verb pairs of each test set or not. They were provided with annotation guidelines where it was needed. For instance, the cause relation is hard to identify, so we adopted the annotation guidelines from [31, 18] which are as follows: *"Assign cause label to a pair* (a, b)*, if the following two conditions are satisfied: (1) a temporally precedes/ overlap in time, (2) while keeping as many state of affairs constant*

as possible, modifying a must entail predictably modifying b. Otherwise assign non-cause label". The kappa scores for the human inter-annotator agreement achieved on causal, temporal, comparison, and expansion Test-sets are 0.51, 0.91, 0.74, and 0.62, respectively. We compared our KB against these data. Table 8 shows the results.

Relation type	precision
causal	37.42%
temporal	75.69%
comparison	62.33%
Expansion	50.26%

Table 8. Precision of our relations

For experiment (4), i.e. the task of ranking verb pairs based on the strength of relations, we randomly selected 10 verbs. Two annotators were asked to sort related verbs of each 10 verbs based on the strength of their association. We employed Spearman's rank correlation co-efficient (Equation (2)) to compare the ranked list of verb pairs based on the scores of our metrics and the rank given by the human annotators.

$$P = \frac{n(\sum x_i y_i) - (\sum x_i)(\sum y_i)}{\sqrt{n(\sum x_i^2) - (\sum x_i)^2}\sqrt{n(\sum y_i^2) - (\sum y_i)^2}}. \tag{2}$$

Here, n is the total number of verb pairs in the test set, x_i is the human annotation rank and y_i is the metric's rank of verb pairs of the test set. Spearman's rank correlation coefficient has a range of $[-1, 1]$. A coefficient of -1 corresponds to the two lists being perfectly negatively correlated (one is the reverse sort of the other), a coefficient of 1 corresponds to perfect correlation, and a coefficient of 0 for rankings being completely independent.

Table 9 shows the results of evaluating introduced metrics for relation strength. The A, B, C, D, and E schemas are λ, wSeq, wSeq + CAW, wSeq + CAW + catVal, and combined metrics, respectively.

Metric schema	A	B	C	D	E
P score	0.2561	0.34993	0.57852	0.61254	0.74125

Table 9. The Spearman's rank correlation coefficient for the metrics

6 CONCLUSION

Providing a repository of semantic relation between verbs is of great importance in various NLP applications including Question Answering [1, 2], Machine Translation, Information Extraction, Coreference Resolution [3], Prediction [4], Summarization [5], Recognizing Textual Entailment [6], etc.

In this paper, we discussed how parsed data of a big corpus could have a significant impact on creating semantic relations repository between verbs. We used both verb and action nominals as event triggers. Incorporating connecting links between event pairs, we tried to classify relations types to categories such as causal relations, temporal relations, comparison relations, and expansion relations. In order to determine the strength of association between verb pairs, we introduced some numerical measures including wSeq, CAW, catVal, and PMI. We evaluated our work against two freely available resources of semantic verbs. As reported in the evaluation section, the result was promising.

On the other hand, one limitation in our work is that there is no phrasal verb in our repository. The reason is that the parser used in parsing source corpora did not distinguish between particle and preposition. In other word, it treats *put on shoulder* and *put on clothes* the same, while in the former *on* is a preposition and in the latter it is a particle. This way, for a sentence like *put on clothes* we wrongly have *put* as the verb. This has a negative effect on the quality of our results.

REFERENCES

[1] Girju, R.: Automatic Detection of Causal Relations for Question Answering. Proceedings of the ACL 2003 Workshop on Multilingual Summarization and Question Answering (MultiSumQA '03), Vol. 12, 2003, pp. 76–83, doi: 10.3115/1119312.1119322.

[2] Higashinaka, R.—Isozaki, H.: Automatically Acquiring Causal Expression Patterns from Relation-Annotated Corpora to Improve Question Answering for Why-Questions. ACM Transactions on Asian Language Information Processing, Vol. 7, 2008, No. 2, Art. No. 6, doi: 10.1145/1362782.1362785.

[3] Bean, D.—Riloff, E.: Unsupervised Learning of Contextual Role Knowledge for Coreference Resolution. Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004), 2004, pp. 297–304.

[4] Radinsky, K.—Horvitz, E.: Mining the Web to Predict Future Events. Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM '13), 2013, pp. 255–264, doi: 10.1145/2433396.2433431.

[5] Marcu, D.: From Discourse Structures to Text Summaries. Proceedings of the ACL, 1997, pp. 82–88.

[6] Pazienza, M. T.—Pennacchiotti, M.—Zanzotto, F. M.: Mixing Wordnet, Verbnet and Propbank for Studying Verb Relations. Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC '06), 2006, pp. 1372–1377.

[7] Baker, C. F.—Fillmore, C. J.—Lowe, J. B.: The Berkeley FrameNet Project. Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics –

Volume 1 (ACL '98/COLING '98), Montreal, Canada, 1998, pp. 86–90, doi: 10.3115/980845.980860.

[8] Kipper, K.—Dang, H. T.—Palmer, M.: Class-Based Construction of a Verb Lexicon. Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI), 2000, pp. 691–696.

[9] Kingsbury, P.—Palmer, M.—Marcus, M.: Adding Semantic Annotation to the Penn Treebank. Proceedings of the Human Language Technology Conference, 2002, pp. 252–256.

[10] Miller, G. A.: WordNet: A Lexical Database for English. Communications of the ACM, Vol. 38, 1995, No. 11, pp. 39–41, doi: 10.1145/219717.219748.

[11] Baroni, M.—Lenci, A.: Distributional Memory: A General Framework for Corpus-Based Semantics. Computational Linguistics, Vol. 36, 2010, No. 4, pp. 673–721, doi: 10.1162/coli_a_00016.

[12] Chang, D.-S.—Choi, K.-S.: Causal Relation Extraction Using Cue Phrase and Lexical Pair Probabilities. In: Su, K. Y., Tsujii, J., Lee, J. H., Kwong, O. Y. (Eds.): Natural Language Processing – IJCNLP 2004. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3248, 2004, pp. 61–70, doi: 10.1007/978-3-540-30211-7_7.

[13] Chklovski, T.—Pante, P.: VerbOcean: Mining the Web for Fine-Grained Semantic Verb Relations. Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP), Vol. 4, 2004, pp. 33–40.

[14] Do, Q. X.—Chan, Y. S.—Roth, D.: Minimally Supervised Event Causality Identification. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11), 2011, pp. 294–303.

[15] Kozareva, Z.: Cause-Effect Relation Learning. Workshop Proceedings of TextGraphs-7 on Graph-Based Methods for Natural Language Processing (TextGraphs-7 '12), 2012, pp. 39–43.

[16] Mirza, P.—Tonelli, S.: An Analysis of Causality Between Events and Its Relation to Temporal Information. Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, 2014, pp. 2097–2106.

[17] Riaz, M.—Girju, R.: Toward a Better Understanding of Causality Between Verbal Events: Extraction and Analysis of the Causal Power of Verb-Verb Associations. Proceedings of the Annual SIGDIAL Meeting on Discourse and Dialogue (SIGDIAL 2013), 2013, pp. 21–30.

[18] Riaz, M.—Girju, R.: Recognizing Causality in Verb-Noun Pairs via Noun and Verb Semantics. Proceedings of the EACL 2014 Workshop on Computational Approaches to Causality in Language (CAtoCL), 2014, pp. 48–57, doi: 10.3115/v1/W14-0707.

[19] Riaz, M.—Girju, R.: In-Depth Exploitation of Noun and Verb Semantics to Identify Causation in Verb-Noun Pairs. In 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL), 2014, pp. 161–170, doi: 10.3115/v1/w14-4322.

[20] Conrath, J.—Afantenos, S.—Asher, N.—Muller, P.: Unsupervised Extraction of Semantic Relations Using Discourse Cues. Proceedings of COLING 2014, the

25th International Conference on Computational Linguistics: Technical Papers, 2014, pp. 2184–2194.

[21] Sorgente, A.—Vettigli, G.—Mele, F.: Automatic Extraction of Cause-Effect Relations in Natural Language Text. In: Lai, C., Semeraro, G., Giuliani, A. (Eds.): Proceedings of the 7th International Workshop on Information Filtering and Retrieval. CEUR Workshop Proceedings, 2013, Vol. 1109, pp. 37–48.

[22] Chambers, N.—Jurafsky, D.: Unsupervised Learning of Narrative Schemas and Their Participants. Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL '09), 2009, pp. 602–610, doi: 10.3115/1690219.1690231.

[23] Chambers, N.—Jurafsky, D.: Unsupervised Learning of Narrative Event Chains. Proceedings of ACL-08: HLT, 2008, pp. 789–797.

[24] Harris, Z. S.: Distributional Structure. Word, Vol. 10, 1954, No. 2-3, pp. 146–162, doi: 10.1080/00437956.1954.11659520.

[25] Comrie, B.: The Syntax of Action Nominals: A Cross-Language Study. Lingua, Vol. 40, 1976, No. 2–3, pp. 177–201, doi: 10.1016/0024-3841(76)90093-0.

[26] Pustejovsky, J.—Hanks, P.—Sauri, R. et al.: The TIMEBANK Corpus. Proceedings of Corpus Linguistics, 2003, pp. 647–656.

[27] Moro, A.—Cecconi, F.—Navigli, R.: Multilingual Word Sense Disambiguation and Entity Linking for Everybody. Proceedings of the 2014 International Semantic Web Conference – Posters and Demonstrations Track (ISWC-PD '14). CEUR Workshop Proceedings, Vol. 1272, 2014, pp. 25–28.

[28] Prasad, A. J.—Miltsakaki, E.—Dinesh, N.—Lee, A. et al.: The Penn Discourse Treebank 2.0 Annotation Manual. The PDTB Research Group, 2007.

[29] Hall, M.—Frank, E.—Holmes, G.—Pfahringer, B.—Reutemann, P.—Witten, I. H.: The WEKA Data Mining Software. ACM SIGKDD Explorations Newsletter, Vol. 11, 2009, No. 1, pp. 10–18, doi: 10.1145/1656274.1656278.

[30] Nivre, J.—Hall, J.—Nilsson, J.—Chanev, A.—Eryigit, G.—Kübler, S.—Marinov, S.—Marsi, E.: MaltParser: A Language-Independent System for Data-Driven Dependency Parsing. Natural Language Engineering, Vol. 13, 2007, No. 2, pp. 95–135, doi: 10.1017/s1351324906004505.

[31] Riaz, M.—Girju, R.: Another Look at Causality: Discovering Scenario-Specific Contingency Relationships with No Supervision. In 2010 IEEE Fourth International Conference on Semantic Computing (ICSC), 2010, pp. 361–368, doi: 10.1109/icsc.2010.19.

[32] Bach, E.: The Algebra of Events. In: Portner, P. H., Partee, B. H. (Eds.): Formal Semantics: The Essential Readings. Chapter 13. 2002, pp. 324–333, doi: 10.1002/9780470758335.ch13.

[33] Dölling, J.: Aspectual Coercion and Eventuality Structure. In: Robering, K. (Ed.): Events, Arguments, and Aspects: Topics in the Semantics of the Verbs. Studies in Language Companion Series, Vol. 152, 2014, pp. 189–226, doi: 10.1075/slcs.152.05dol.

[34] Moens, M.—Steedman, M.: Temporal Ontology and Temporal Reference. Computational Linguistics, Vol. 14, 1988, No. 2, pp. 15–28.

[35] Pustejovsky, J.—Castaño, J. M.—Ingria, R.—Saurí, R.—Gaizauskas, R. J.—Setzer, A.—Katz, G.—Radev, D. R.: TimeML: Robust Specification of Event and Temporal Expressions in Text. New Directions in Question Answering, Vol. 3, 2003, pp. 28–34.

[36] Pustejovsky, J.: The Syntax of Event Structure. Cognition, Vol. 41, 1991, No. 1-3, pp. 47–81, doi: 10.1016/0010-0277(91)90032-y.

[37] Saurí, R.—Knippen, R.—Verhagen, M.—Pustejovsky, J.: Evita: A Robust Event Recognizer for QA Systems. Proceedings of the Conference on Human Language Technology and Conference on Empirical Methods in Natural Language Processing, 2005, pp. 700–707, doi: 10.3115/1220575.1220663.

[38] Sprugnoli, R.—Tonelli, S.: One, No One and One Hundred Thousand Events: Defining and Processing Events in an Inter-Disciplinary Perspective. Natural Language Engineering, Vol. 23, 2016, No. 4, pp. 485–506, doi: 10.1017/s1351324916000292.

[39] De Swart, H.: Aspect Shift and Coercion. Natural Language and Linguistic Theory, Vol. 16, 1998, No. 2, pp. 347–385, doi: 10.1023/A:1005916004600.

[40] de Swart, H.—Verkuyl, H.: Tense and Aspect in Sentence and Discourse. ESSLLI Summer School, 1999.

[41] Vendler, Z.: Verbs and Times. Linguistics in Philosophy, Chapter 4. Cornell University Press, 1967, pp. 97–121.

[42] Sasse, H.-J.: Recent Activity in the Theory of Aspect: Accomplishments, Achievements, or Just Non-Progressive State? Linguistic Typology, Vol. 6, 2002, No. 2, pp. 199–271, doi: 10.1515/lity.2002.007.

Hasan Zafari received his B.Sc., M.Sc., and Ph.D. degrees in computer science in 2005, 2008, and 2017, respectively. Since 2008, he has been a faculty member in the Department of Computer Engineering at the Malayer Branch of Islamic Azad University. He has taught several courses on computer science over these years. His research interests include natural language processing, machine learning, deep learning, and data processing.

Maryam Hourali received her B.Sc. degree in mathematics as first rank from the University of Tehran in 2004, the M.Sc. degree in information technology from University of Science and Technology, Iran, in 2007, and the Ph.D. from Tarbiat Modares University in 2013. In 2012 she joined the Department of ICT, Malek-Ashtar University of Technology, Tehran, Iran. Her current research interests include AI, NLP, machine learning, text mining, and ontology.