

Unsupervised Event Abstraction using Pattern Abstraction and Local Process Models

Felix Mannhardt ✉ and Niek Tax ✉

Eindhoven University of Technology
{f.mannhardt,n.tax}@tue.nl

Abstract. Process mining analyzes business processes based on events stored in event logs. However, some recorded events may correspond to activities on a very low level of abstraction. When events are recorded on a too low level of abstraction, process discovery methods tend to generate overgeneralizing process models. Grouping low-level events to higher level activities, i.e., event abstraction, can be used to discover better process models. Existing event abstraction methods are mainly based on common sub-sequences and clustering techniques. In this paper, we propose to first discover local process models and, then, use those models to lift the event log to a higher level of abstraction. Our conjecture is that process models discovered on the obtained high-level event log return process models of higher quality: their fitness and precision scores are more balanced. We show this with preliminary results on several real-life event logs.

Keywords: Process Discovery, Unsupervised Learning, Event Abstraction

1 Introduction

Process mining [1] is a fast growing research discipline that concerns the analysis of events that are logged during the execution of a business process. Recorded events contain information on what was done, by whom, where, when, etc. Such event data is often readily available from business information systems such as ERP, CRM, or, workflow management systems. Process discovery, the task of automatically generating a process model that accurately describes the business process based on the event data, plays a central role in the process mining field. A variety of process discovery techniques have been developed over the years [3, 6, 9, 20], generating process models in different notations, such as Petri nets [15], EPC, and BPMN. The degree to which a discovered process model represents the event data from which is discovered is typically expressed in several quality dimensions. Two of such quality dimensions are *fitness*: the amount of behavior in the event log that is allowed by the model and *precision*: the model should not be too general by allowing for much more behavior that was not seen in the event log (i.e., it should not be underfitting).

For successful application of process discovery it is crucial that the events logged in the event log directly correspond to the activities that are recognizable

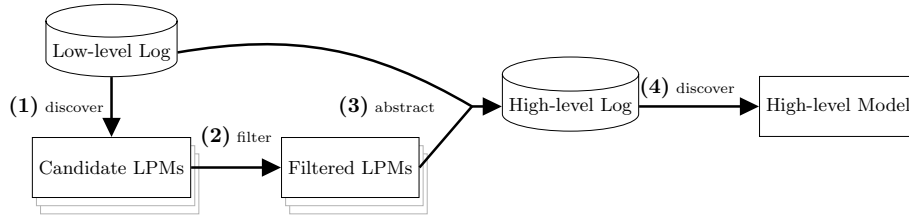


Fig. 1. An overview of the proposed unsupervised abstraction technique.

for process stakeholders. In practice, this is not always the case, and there can be an $n:m$ -relation between the recorded events and activities of the process [4, 8]. Process models that are generated by process discovery when recorded events and activities do not match have semantics that are unclear to process stakeholders. Moreover, a mismatch between events and activities can cause process discovery techniques to discover underfitting process models that allow for too much behavior (i.e., process models with low precision) [18].

A recent approach to abstract recorded events to high-level activities [13] uses *activity patterns* to capture the domain knowledge about the relation between the high-level activities and the low-level recorded events. Each activity pattern is a process model that describes the possible behavior in terms of low-level events that are conjectured to be observed during the execution of a certain high-level activity. However, such domain knowledge might not be available, and when the process contains many activities it becomes a tedious task to model each of them manually. Local Process Model (LPM) discovery [17, 19] is a technique to automatically discover frequent patterns of process behavior (i.e., LPMs) from an event log. Each LPM, like an *activity pattern*, is a process model that describes the behavior over only a subset of the event types in the log.

In this paper we explore the application of automatically discovered LPMs to replace the domain knowledge in pattern-based abstraction to form a *novel, completely automated, abstraction technique*. Figure 1 gives an overview of the proposed method. Each LPM discovered by the LPM discovery method [17, 19] is assumed to represent a high-level activity. Then, we propose a technique to filter the set of LPMs, and use this filtered set of LPMs as activity patterns with the event abstraction method proposed in [13] and discover a high-level process model using off-the-shelf discovery methods. Our proposed, integrated approach aims to *improve the precision* of process models found by process discovery techniques by abstraction of the event log while not sacrificing too much fitness.

In Section 2, we introduce basic concepts and notations. Section 3 explains how LPM discovery and pattern-based abstraction can be combined to form a fully automated abstraction technique. In Section 4, we present and discuss some preliminary results, and we conclude the paper and discuss future directions in Section 5.

2 Preliminaries

In this section we introduce concepts used in later sections of this paper. X^* denotes the set of all sequences over a set X and $\sigma = \langle a_1, a_2, \dots, a_n \rangle$ a sequence of length n , with $\sigma(i) = a_i$. $\langle \rangle$ is the empty sequence.

In the context of process logs, we assume the set of all *process activities* Σ to be given. An **event** e in an event log is the occurrence of an activity $e \in \Sigma$. We call a sequence of events $\sigma \in \Sigma^*$ a **trace**. An **event log** $L \in \mathbb{N}^{\Sigma^*}$ is a finite multiset of traces. For example, the event log $L = [\langle a, b, c \rangle^2, \langle b, a, c \rangle^3]$ consists of 2 occurrences of trace $\langle a, b, c \rangle$ and three occurrences of trace $\langle b, a, c \rangle$.

A process model notation that is frequently used in the process mining area is the Petri net. A **labeled Petri net** $N = \langle P, T, F, \ell \rangle$ is a tuple where P is a finite set of **places**, T is a finite set of **transitions** such that $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation, and $\ell : T \rightarrow \Sigma$ is a labeling function that assigns process activities to transitions. Unlabeled transitions, i.e., $t \in T$ with $t \notin \text{dom}(\ell)$, are referred to as τ -transitions, or invisible transitions.

The state of a Petri net is defined by its *marking*. The marking assigns a finite number of tokens to each place. Transition of the Petri net represent activities and can be executed. The input places of a transition $t \in T$ are all places for which there is a directed edge to the transition: $\{p \in P \mid (p, t) \in F\}$. The output places of a transition are defined respectively. Executing a transition consumes one token from each of its input places and produces one token on each of its output places, i.e., the marking is changed. A transition can only be executed when there is at least one token in each of its input places.

Often it is useful to consider a Petri net in combination with an initial marking and a final marking. This allows us to define the language, $\mathcal{L}(N)$, accepted by Petri net N . The language of a Petri net is defined by the set of all possible sequences of visible transition labels (i.e., ignoring τ -transitions) that start in the initial marking and end in the final marking. This allows to check whether some behavior is part of the behavior of the Petri net, i.e., can be replayed on it.

Figure 2a shows an example of an accepting Petri net. Circles represent places and rectangles represent transitions. Invisible transitions are depicted as black rectangles. Places that belong to the initial marking contain a token and places belonging to a final marking are marked as \odot . The language of this accepting Petri net, with $\Sigma = \{A, B, C\}$ is $\{\langle A, B, C \rangle, \langle B, A, C \rangle, \langle A, B, B, C \rangle, \langle B, B, A, C \rangle, \langle B, A, B, C \rangle, \dots\}$. We refer to [15] for a comprehensive introduction of Petri nets.

2.1 Local Process Models

LPMS [19] are process models that describe the behavior seen in the event log only partially, focusing on frequently observed behavior. Typically, LPMS describe the behavior of only up to 5 activities. LPMS can be represented in any process modeling notation, such as BPMN, UML, or EPC. Here we use Petri nets to represent LPMS. A technique to generate a ranked collection of LPMS through iterative expansion of candidate process models is proposed in [19]. The search

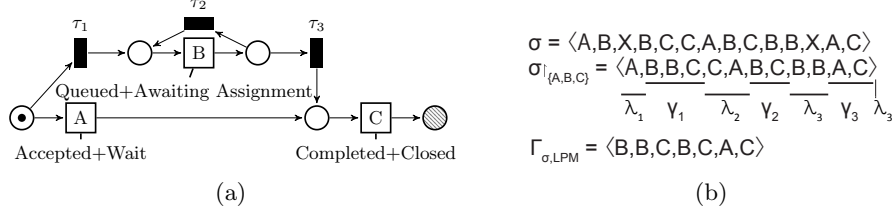


Fig. 2. (a) Example Petri net N_1 . (b) Trace σ of a log L and its segmentation on N_1 .

space of process models is fixed, depending on the event log. We define $LPMS(L)$ as the set of possible LPMs that can be constructed for given event log L . We refer the reader to [19] for a detailed description of search space $LPMS(L)$.

To evaluate a given LPM on a given event log L , its traces $\sigma \in L$ are first projected on the set of activities Σ in the LPM, i.e., $\sigma' = \sigma|_{\Sigma}$. The projected trace σ' is then segmented into γ -segments that fit the behavior of the LPM and λ -segments that do not fit the behavior of the LPM, i.e., $\sigma' = \lambda_1 \gamma_1 \lambda_2 \gamma_2 \dots \lambda_n \gamma_n \lambda_{n+1}$ such that $\gamma_i \in \mathcal{L}(LPM)$ and $\lambda_i \notin \mathcal{L}(LPM)$. We define $\Gamma_{\sigma, LPM}$ to be a function that projects trace σ on the LPM activities and obtains its subsequences that fit the LPM, i.e., $\Gamma_{\sigma, LPM} = \gamma_1 \gamma_2 \dots \gamma_n$.

Let our LPM N_1 under evaluation be the Petri net of Figure 2a and let $\sigma = \langle A, B, X, B, C, C, A, B, C, B, B, X, A, C \rangle$ be an example trace. Function $act(LPM)$ obtains the set of process activities in the LPM, e.g. $act(N_1) = \{A, B, C\}$. Projection on the activities of the LPM gives $\sigma|_{act(N_1)} = \langle A, B, B, C, C, A, B, C, B, B, A, C \rangle$. Figure 2b shows the segmentation of the projected traces on the LPM, leading to $\Gamma_{\sigma, LPM} = \langle B, B, C, B, C, A, C \rangle$. The segmentation starts with a non-fitting segment $\lambda_1 = \langle A \rangle$, followed by a fitting segment $\gamma_1 = \langle B, B, C \rangle$, which completes one run through the model from initial to final marking. The second event C in σ cannot be replayed on LPM , since it only allows for one C and γ_1 already contains a C . This results in a non-fitting segment $\lambda_2 = \langle C, A \rangle$. $\gamma_2 = \langle B, C \rangle$ again represents a run through the model from initial to final marking, and $\lambda_3 = \langle B, B \rangle$ does not fit the LPM. $\gamma_3 = \langle A, C \rangle$ again represents a run through the model, and we end with an empty non-fitting segment λ_4 . We lift segmentation function Γ to event logs, $\Gamma_{L, LPM} = \{\Gamma_{\sigma, LPM} | \sigma \in L\}$. An alignment-based [2] implementation of Γ , as well as a method to rank and select LPMs based on their support, i.e., the number of events in $\Gamma_{L, LPM}$, is described in [19].

2.2 Pattern-based Abstraction

An overview of the pattern-based abstraction method proposed in [13] is depicted in Figure 3. The input is an event log and a set of activity patterns. Each activity pattern is a process model that represents the behavior expected for one execution of a high-level activity. Moreover, a mapping from activities to life-cycle transitions of the high-level activity is required, i.e., $\theta : T \not\rightarrow LT$ with, e.g., $LT = \{start, complete\}$. Mapping θ allows to obtain information on when

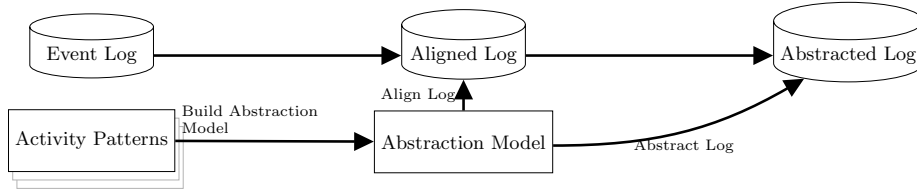


Fig. 3. An overview of the pattern-based abstraction method described in [13]. These steps jointly form step 3 of Figure 1.”

activities started and when they were completed, which some process discovery algorithms, such as [10], are able to leverage.

Activity patterns are composed to an abstraction model and, then, an alignment technique [2] is used to obtain a high-level event log. Through the use of the alignment technique, we can capture approximate executions of activity patterns. Then, we use the alignment information to create a corresponding high-level event log by only retaining those events that were aligned to activities $t \in T$ that are mapped to high-level activities, i.e., $t \in \text{dom}(\theta)$. Activity patterns may use any kind of process models with clear semantics, thus, the abstraction method also works with LPMs. For example, assume that LPM N_1 represents the behavior expected for some high-level activity. Mapping function θ could be defined such that the transition corresponding to A and transition τ_1 are mapped to *start* and the transition corresponding to C is mapped to *complete*. It is possible to automatically obtain such mapping function θ , e.g., by mapping the source activities to the start and the and the sink activities to the complete life-cycle transitions.

3 Unsupervised Abstraction Technique

We use discovered LPMs to replace the domain knowledge originally used in the pattern-based event abstraction method. Our proposed method discovers a high-level process model in the following four steps shown graphically in Fig. 1.

1. We discover a **fixed number of candidate LPMs** based on the ranking proposed in [19]. The LPM ranking proposed in [19] is based on support (i.e., frequency). When setting the number of LPMs to use for abstraction to k , we select the top k LPMs of the discovered ranking of LPMs R . For event logs with large numbers of activities the original LPM discovery algorithm [19] becomes computationally infeasible. For such logs, faster, approximate, techniques for discovering LPMs can be applied [17].
2. The LPMs in ranking R can overlap in the activities that they describe. For the purpose of event abstraction this can be undesirable, because this results in multiple similar patterns for pattern-based abstraction, making it unclear which low-level event belongs to which high-level activity. Therefore, we introduce a **diversity score** for the i -th LPM of LPM ranking R as:

$$div(R, i) = \begin{cases} \max_{j=1}^{i-1} \frac{|act(R(i)) \cap act(R(j))|}{|act(R(i))| + |act(R(i)) - |act(R(i)) \cap act(R(j))|} & \text{if } i > 1, \\ 1 & \text{if } i = 1. \end{cases}$$

Based on this definition of diversity we introduce a diversity threshold t_{div} to **filter out** all LPMs from R where $div(R, i) \leq t_{div}$. This filter removes LPMs from R when there is a stronger LPM, higher in ranking R , that describes (almost) the same set of activities.

3. We use those LPMs as **activity patterns** and obtain a high-level event log with the **abstraction method** described in [13]. In work [13] several composition functions are presented that allow fine grained control of the interaction between activities. In our case, we do not assume any domain knowledge on the process, thus, we limit the choice of composition parameter for our method to: *interleaving* and *parallel*. Interleaving means that any two high-level activities cannot occur at the same time, whereas parallel means that no such restriction is placed.
4. Based on the high-level event log, we **discover a high-level process model** using existing methods such as the Inductive Miner (IM) [9].

For example, take LPM N_1 shown in Fig. 2 and assume that we apply the proposed method to trace σ . We assume that LPM N_1 represents the behavior of some high-level activity H . Three executions (i.e., γ -segments) of the LPM N_1 are present in trace σ , thus, the high-level activity H was executed three times. When applying pattern-based abstraction, we obtain the high-level trace $\langle H, H, H \rangle^1$. Some low-level events in σ could not be matched to a high-level activity, e.g., the first event of the trace: A . Since we cannot assume that the discovered LPMs represent the entire behavior of the process, we add all those low-level events to the resulting trace. Therefore, we use the trace $\langle A, H, C, A, H, B, B, H \rangle$, which contains less events than the original trace σ , for discovery.

Note that our technique does not discover the names of the high-level activities represented by the LPMs. However, LPMs can be labeled to its corresponding business activity based on domain knowledge.

4 Preliminary Results and Discussion

To be able to deal with the computational complexity of LPM mining for logs with many activities, we discover LPMs using activity clustering based on Markov clustering, as proposed in [17]. As proposed in [14], we expand high-level activities in the discovered process model with the corresponding LPMs to provide a fair comparison based on the low-level event log. The resulting *expanded process model* can be related to the low-level events with existing techniques that determine the models quality in terms of fitness and precision [2]. We evaluate the quality of the expanded process models in terms of F-score [7], i.e., the harmonic mean between fitness and precision.

¹ The abstraction technique also provides events for the start and complete life-cycle transitions. So far, we only use the complete transition.

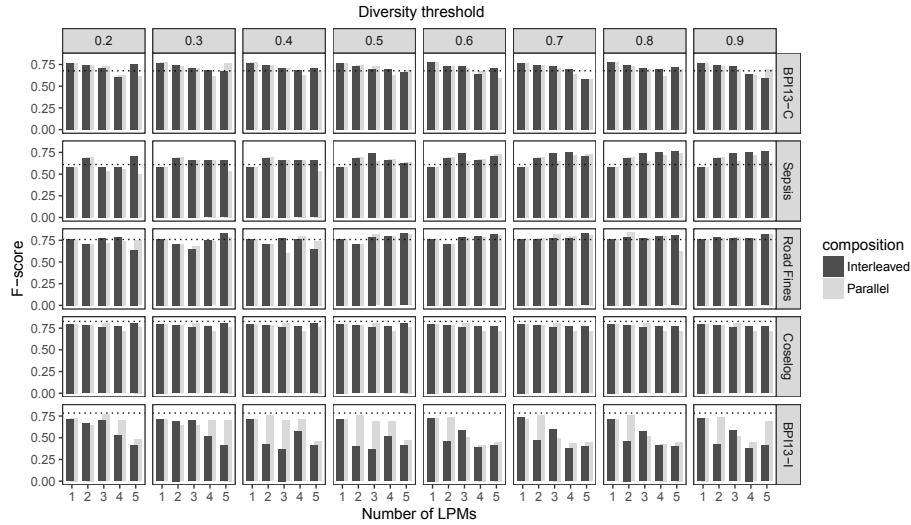


Fig. 4. The F-scores of models discovered with the IM infrequent after applying the proposed method compared to the F-score obtained without the method (dotted line).

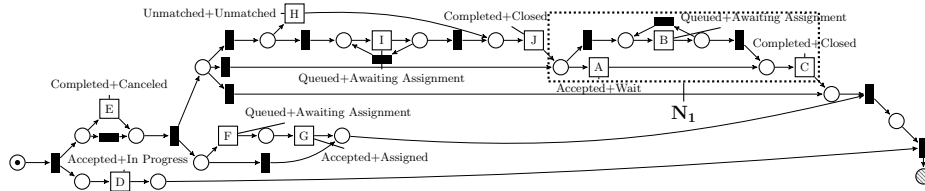


Fig. 5. A Petri net discovered for the BPI13-C event log with parallel composition.

Figure 4 shows the F-score of the process models discovered with the IM infrequent [9] process discovery algorithm with 20% noise filtering. Horizontally it shows the results for different LPM *diversity thresholds* (0.2 to 0.9). Vertically it shows the results for five different event logs: the BPI challenge 2013 incidents (I) and closed problems (C) log[16], the CoSeLoG receipt phase log [5], the road fines log [11], and the sepsis log [12]. The results are shown for 1 to 5 LPMs used in abstraction and parallel and interleaving composition. The dotted line indicates the F-score of the process model discovered from the original event log, i.e., without abstraction.

Figure 5 shows an expanded process model that is discovered for the BPI13-C log based on the proposed method. The LPM N_1 was used as activity pattern and is part of the process model. Thus, the process model can hierarchically decomposed into sub-processes based on the activity patterns. Its precision score is improved from 0.53 to 0.86 at the expense of fitness, which drops from 0.84 to 0.65. The F-score improved from 0.65 to 0.74.

In our preliminary results we found that for three out of five event logs the F-score of the process model can be improved by abstracting the event log prior to process discovery. Furthermore, it seems that abstracting with parallel composition does only improve the process model over abstraction with interleaving composition in one case, which is beneficial since the interleaving composition is computationally less expensive. The optimal number of LPMs used for abstraction differs between event logs, with the optimal number of LPMs being 5 for the sepsis log and 1 for the BPI13-C log. To make an abstraction technique based on LPM discovery and pattern-based abstraction fully automated, further experimentation would be needed. We need to analyze whether the optimal number of LPMs depends on properties of the event log and for which logs the good results can be expected.

5 Conclusions and Future Work

In this paper we have described a technique to abstract an event log to a higher-level event log using Local Process Model (LPM) discovery and pattern-based abstraction. We have shown on five real life event logs that the abstraction approach applied prior to process discovery can result in more precise process models.

We found that (1) the number of LPMs that should be used for abstraction, (2) the diversity threshold, and (3) the composition method which result in good process models being discovered are very dependent on the event log on which the technique is applied. In future work, we want to investigate this interplay between event log properties and the parameters of the abstraction approach that are needed to discover process models that strike a good balance between precision and fitness. We aim to apply this insight in the relation between parameters settings of the abstraction technique and properties of the event log in a technique that can abstract an event log completely automatically, where no manual parameter setting is needed.

References

1. van der Aalst, W.M.P.: Process mining: data science in action. Springer (2016)
2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. *WIREs Data Min. Knowl. Discovery* 2(2), 182–192 (2012)
3. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.* 16(9), 1128–1142 (2004)
4. Baier, T., Mendling, J.: Bridging abstraction layers in process mining by automated matching of events and activities. In: *BPM, LNCS*, vol. 8094, pp. 17–32. Springer (2013)
5. Buijs, J.: Receipt phase of an environmental permit application process (WABO), CoSeLoG project. Eindhoven University of Technology. Dataset (2014), <http://dx.doi.org/10.4121/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6>

6. Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: BPMN miner: automated discovery of BPMN process models with hierarchical structure. *Information Systems* 56, 284–303 (2016)
7. De Weerd, J., De Backer, M., Vanthienen, J., Baesens, B.: A robust F-measure for evaluating discovered process models. In: CIDM. pp. 148–155. IEEE (2011)
8. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity mining by global trace segmentation. In: BPM. pp. 128–139. Springer (2009)
9. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM. LNBIP, vol. 171, pp. 66–78. Springer (2013)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Using life cycle information in process discovery. In: BPM. LNBIP, vol. 256, pp. 204–217. Springer (2015)
11. de Leoni, M., Mannhardt, F.: Road traffic fine management process. Eindhoven University of Technology. Dataset (2015), <http://dx.doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5>
12. Mannhardt, F.: Sepsis cases - event log. Eindhoven University of Technology. Dataset (2016), <http://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460>
13. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M., Toussaint, P.J.: From low-level events to activities - a pattern-based approach. In: BPM. LNCS, vol. 9850, pp. 125–141. Springer (2016)
14. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M., Toussaint, P.J.: Guided process discovery - a pattern-based approach. *Information Systems* (submitted)
15. Reisig, W.: Petri nets: an introduction, vol. 4. Springer (2012)
16. Steeman, W.: BPI challenge 2013. Ghent University. Dataset (2013), <http://dx.doi.org/10.4121/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6>
17. Tax, N., Sidorova, N., van der Aalst, W.M.P., Haakma, R.: Heuristic approaches for generating local process models through log projections. In: CIDM. pp. 1–8. IEEE (2016)
18. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Event abstraction for process mining using supervised learning techniques. In: IntelliSys. IEEE (2016)
19. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.: Mining local process models. *Journal of Innovation in Digital Ecosystems* 3(2), 183–196 (2016)
20. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. *Fundamenta Informaticae* 94(3), 387–412 (2009)