Rule Systems for Run-Time Monitoring: From EAGLE to RULER

Howard Barringer[1], David Rydeheard[1], and Klaus Havelund[2,⋆]

[1] School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, UK
{Howard.Barringer,David.Rydeheard}@manchester.ac.uk
[2] NASA's Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA
Klaus.Havelund@jpl.nasa.gov

Abstract. In [3], EAGLE was introduced as a general purpose rule-based temporal logic for specifying run-time monitors. A novel and relatively efficient interpretative trace-checking scheme via stepwise transformation of an EAGLE monitoring formula was defined and implemented. However, application in real-world examples has shown efficiency weaknesses, especially those associated with large-scale symbolic formula manipulation. In this paper, after briefly reviewing EAGLE, we introduce RULER, a primitive conditional rule-based system, which we claim can be more efficiently implemented for run-time checking, and into which one can compile various temporal logics used for run-time verification.

1 Introduction

In earlier work, the rule-based temporal logic EAGLE [3] was developed as a generalisation of the plethora of logics which have been used for the specification of behavioural system properties and which can be dynamically checked either on-line throughout an execution of the system or off-line over an execution trace of the system. We showed that EAGLE supported future and past time logics, interval logics, extended regular expressions, state machines, logics for real-time and data constraints, and temporal-based logics for stochastic behaviour.

The EAGLE logic is a restricted first order, fixed-point, linear-time temporal logic with chop (concatenation) over *finite* traces. As such, the logic is highly expressive and, not surprisingly, EAGLE's satisfiability (validity) problem is undecidable; checking satisfiability in a given model, however, is decidable and that is what's required for run-time verification. The syntax and semantics of EAGLE are succinct. There are four primitive temporal operators: $\bigcirc$ — next, $\odot$ — previously, $\cdot$ — concatenation, and ; — chop (overlapping concatenation, or sequential composition). Temporal equations can be used to define schema

⋆ The work of this author was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

O. Sokolsky and S. Tasiran (Eds.): RV 2007, LNCS 4839, pp. 111–125, 2007.

for temporal formulas, where the temporal predicates may be parameterized by data as well as by EAGLE formulas. The usual boolean logical connectives exist. For example, the linear-time $\Box$, $\Diamond$, $\mathcal{U}$ and $\mathcal{S}$ (always, sometime, until and since) temporal operators can be introduced through the following equational definitions.

$$\begin{aligned}
&\textbf{max } \texttt{Always}(\textbf{Form } F) = F \wedge \bigcirc \texttt{Always}(F)\\
&\textbf{min } \texttt{Sometime}(\textbf{Form} F) = F \vee \bigcirc \texttt{Sometime}(F)\\
&\textbf{min } \texttt{Until}(\textbf{Form } F_1, \textbf{Form } F_2) = F_2 \vee (F_1 \wedge \bigcirc \texttt{Until}(F_1, F_2))\\
&\textbf{min } \texttt{Since}(\textbf{Form } F_1, \textbf{Form } F_2) = (F_2 \vee (F_1 \wedge \odot \texttt{Since}(F_1, F_2)))
\end{aligned}$$

The qualifiers **max** and **min** indicate the positive and, respectively, negative interpretation that is to be given to the associated temporal predicate at trace boundaries — corresponding to maximal and minimal solutions to the equations. Thus $\bigcirc\texttt{Always}(p)$ is defined to be true in the last state of a given trace, whereas $\bigcirc\texttt{Until}(p, q)$ is false in the last state. Thus the formula $\texttt{Always}(p)$ will hold on a finite sequence from, say index i, if and only if p holds in every state from index i up to and including the final state. Whereas, if $\texttt{Until}(p, q)$ holds at index i then q must be true at some state with index $j \geq i$ and p true on all states from i up to but not including j.[1]

Even without data parametrization, the primitive concatenation temporal operators together with the recursively defined temporal predicates take the logic into the world of context-free expressivity thus enabling simple grammatical-like specification of parenthesis, call return, or login logout matching. Assume *call* and *return* are propositions denoting procedure call and return events. The temporal formula $\texttt{Match}(call, return)$ where

$$\begin{aligned}
&\textbf{min } \texttt{Match}(\textbf{Form } C, \textbf{Form } R) =\\
&\qquad C \cdot \texttt{Match}(C, R) \cdot R \cdot \texttt{Match}(C, R) \vee \texttt{Empty}()
\end{aligned}$$

with $\texttt{Empty}()$ true just on the empty sequence captures the behaviour that every *call* has a matching *return* — a call may be followed by a (possibly empty) sequence of matched calls and returns, followed a return, followed by another (possibly empty) sequence of calls and returns. Parametrization of temporal predicates by data values allows us to define real-time and stochastic logical operators. To address real-time, for example, we assume that EAGLE is monitoring time-stamped states, where the state contains a variable *clock* holding the associated real time. Then it becomes straightforward to define real-time qualified temporal operators such as *happens before real time* u.

$$\begin{aligned}
&\textbf{min } \texttt{HappensBefore}(\textbf{Form } F, \textbf{double } u) =\\
&\qquad clock < u \wedge (F \vee (\neg F \wedge \bigcirc \texttt{HappensBefore}(F, u)))
\end{aligned}$$

It should be clear how more complex real-time, and even probabilistic, temporal operators can be recursively defined.

We still claim that EAGLE presents a natural rule/equation based language for defining, even programming, monitors for complex temporal behavioural

[1] Arguments for using other interpretations over finite traces have been put forward. However, we have found that this simple interpretation has been adequate for our monitoring purposes.

patterns. EAGLE is, however, expressively rich and in general this comes with a potentially high computational cost, practically speaking. So one might ask whether EAGLE presents the most appropriate set of primitive temporal operators for run-time monitoring. The non-deterministic concatenation operator, as used above in the matching parentheses example, requires considerable care in use. In order to achieve the expected temporal behaviour pattern, the formulas passed to `Match` should specify single state sequences. If that is not the case, the concatenation operator may choose an arbitrary cut point, and therefore skip unmatched Bs or Es in order to give a positive result. Later, but currently unpublished, work developed such arguments further and proposed a variety of deterministic versions of temporal concatenation and chop for run-time monitoring, using different forms of cut, e.g. left and right minimal, left and right maximal, etc..

With respect to the computational effectiveness of algorithms for EAGLE trace-checking, in [3] we showed how trace-checking of full EAGLE can be undertaken on a state-by-state basis without recording the full history, even though the logic has the same temporal expressiveness over the past as over the future; basically, our published trace-check algorithm maintains sufficient knowledge about the past in the evolving monitor formulas. Furthermore, we have shown that for restricted subsets, we can achieve close to optimal complexity bounds for monitoring; one such fragment for which we computed complexity results was the LTL (past and future) fragment of EAGLE [4]. However, considerable care must be taken with the presence of data arguments in temporal predicates, for an explosion in the size of the evolving monitor formula may occur.

What was clear to us at the time was that there were some practically useful and efficiently executable subsets of EAGLE. Despite the pleasing features of EAGLE, we still believe we should continue to search for a powerful and simpler "core" logic, one that is easy and efficient to evaluate for monitoring purposes. To that end, we present in the remainder of this paper a seemingly simpler, lower-level, rule-based system RULER. In Section 2 we introduce RULER and a simple evaluation algorithm by example. Section 3 then provides a more formal semantic treatment and indicates how propositional temporal logic (with past and future operators) can be compiled into RULER. In Section 4, we then briefly consider RULER with rule parameters and then present brief conclusions and indicate further work in Section 5.

2 RULER by Example

A RULER monitoring program comprises a collection of named rules. A rule is formed from a condition part (antecedent) and a body part (consequent). The rule's condition may be a conjunctive set of literals, whereas the body is a disjunctive set of conjunctive sets of literals, a literal being a positive or negative occurrence of a rule name or an observation name. The idea is that rules can be made active or inactive. For each active rule, if the condition part evaluates to true for the current state (formed from the current observations and previous

obligations of the rule system), then the body of the rule defines what rules are active and what observations must hold in the next state. As a very simple example, consider the rule named r below in the context of some observation named a.

$$r : \multimap\!\!\circ\!\!\rightarrow\ a, r$$

The rule has a vacuous condition. The rule's body is the conjunctive set containing observation a and rule name r. If r is active at the start of monitoring, r's body asserts that the observation a must hold in next monitoring state and the rule r must be active again, thus effectively asserting that observation a must hold in all subsequent monitoring states. If, at some future state, a fails to hold, then there will be a conflict between obligations and actuality. In this simple case, the rule will fail at that particular point.

Figure 1 outlines a basic algorithm for monitoring a sequence of observation states with a set of named rules. Essentially, the algorithm unfolds the active rules according to the given input observation states. As rule bodies are disjunctive, the algorithm computes sets of possible future states in order to avoid the need for backtracking when a rule failure occurs. A rule activation state is a set of rule name literals and observation literals. We demonstrate this algorithm in Example 1 where we consider a set of rules that capture both past time conditions and future time obligations. We will assume that we wish to monitor some temporal behaviour of a system in terms of two properties, a and b. Thus, we arrange for the system to be instrumented to produce an ordered sequence of observation states and that the letters a and b denote particular propositions over an observation state. In effect, we'll treat an observation trace as a sequence of sets of literals[2].

```
create an initial set of initial rule activation states
WHILE observations exist DO
  obtain next observation state
  merge observation state across the set of rule activation states
  raise monitoring exception if there's no self-consistent merged state
  for each of the current and self-consistent merged states,
    use activated rules to generate a successor set of activation states
  union successor sets to form the new frontier of rule activation states
OD
```

Fig. 1. The basic monitoring algorithm

Example 1. We wish to monitor the constraint that whenever property a occurs both now and in the immediate previous state then b must occur as a later observed property. We can characterise this by the linear time temporal logic formula $\Box((a \wedge \bigodot a) \Rightarrow \diamondsuit b)$ where $\diamondsuit$ is the strict "eventually in the future" temporal operator, or using the EAGLE temporal predicates defined in Section 1

[2] We don't allow both x and $\neg x$ to occur in an observation state, for any x.

by the monitoring formula $\texttt{Always}((a \wedge \bigodot a) \Rightarrow \bigcirc\texttt{Sometime}(b))$. In RULER the following set of rules characterise the required temporal behaviour

$$r_0 : \multimap\ r_0, r_1, r_3 \qquad r_1 : a \multimap\ r_2 \qquad r_2 :$$
$$r_3 : a, r_2 \multimap\ b \mid \neg b, r_4 \qquad r_4 : \multimap\ b \mid \neg b, r_4$$

assuming that the monitoring algorithm starts with an initial set of rule activation sets as $\{\{r_0, r_1, r_3\}\}$[3]. Rule r_0 acts as a generator rule; it ensures persistent activity of itself together with r_1 and r_3, i.e. the three rules are always to be active. The empty rule r_2 is used to represent that the temporal constraint $\bigodot a$ holds (hence it is initially inactive). The rule r_1 is then a generator for r_2 and can be viewed as the temporal rule "if we have a today then tomorrow we have yesterday a". Rule r_4 captures the obligation $\diamondsuit b$, either b holds in the next observation state or $\neg b$ holds together with a continued obligation to $\diamondsuit b$.

For the example observation trace in the table below, we see that in step 4, both a and $\bigodot a$ are true (in the merged state, both a and r_2 are present) and hence rule r_3 yields two possibilities for step 5. The choice with b holding true conflicts with the observation in step 5 and therefore is eliminated. Rule r_4 is thus active and remains activated until step 7 when b is observed to hold.

Step	Obs.	Rule Activations	Merged States
0	$\{\}$	$\{\{r_0, r_1, r_3\}\}$	$\{\{r_0, r_1, r_3\}\}$
1	$\{a, b\}$	$\{\{r_0, r_1, r_3\}\}$	$\{\{a, b, r_0, r_1, r_3\}\}$
2	$\{\neg a, b\}$	$\{\{r_0, r_1, r_2, r_3\}\}$	$\{\{\neg a, b, r_0, r_1, r_2, r_3\}\}$
3	$\{a, b\}$	$\{\{r_0, r_1, r_3\}\}$	$\{\{a, b, r_0, r_1, r_3\}\}$
4	$\{a, b\}$	$\{\{r_0, r_1, r_2, r_3\}\}$	$\{\{a, b, r_0, r_1, r_2, r_3\}\}$
5	$\{\neg a, \neg b\}$	$\{\{b, r_0, r_1, r_2, r_3\}, \{\neg b, r_0, r_1, r_2, r_3, r_4\}\}$	$\{\{\neg a, \neg b, r_0, r_1, r_2, r_3, r_4\}\}$
6	$\{a, \neg b\}$	$\{\{b, r_0, r_1, r_3\}, \{\neg b, r_0, r_1, r_3, r_4\}\}$	$\{\{a, \neg b, r_0, r_1, r_3, r_4\}\}$
7	$\{\neg a, b\}$	$\{\{b, r_0, r_1, r_2, r_3\}, \{\neg b, r_0, r_1, r_2, r_3, r_4\}\}$	$\{\{\neg a, b, r_0, r_1, r_2, r_3\}\}$
8	$\{\neg a, \neg b\}$	$\{\{r_0, r_1, r_3\}\}$	$\{\{\neg a, \neg b, r_0, r_1, r_3\}\}$

But how do we determine whether any generated temporal existential obligations, such as $\diamondsuit b$, have indeed been satisfied? Essentially, the rule system structure notes those rules that correspond to such obligations and then, at the end of monitoring, one must check whether the final merged state set contains states without those noted rules active. If there are no such states, then the given (finite) observation trace fails to satisfy the rule set. If there is at least one of the possible final states not containing such noted rules, the observation trace satisfies the rule set. The approach is exactly that of the minimal and maximal rule interpretations used in EAGLE. In the above, the final set of merged states has just one state that does not contain the noted rule r_4 and hence the observation satisfies the given rule set.

The rule set in fact contained an optimisation; the choices appearing in rules r_3 and r_4 were made deterministic, either b or $\neg b \wedge ....$ The determinisation thus

[3] The absence of r_2 from this set gives $\neg r_2$ a positive interpretation; this is not the case, however, for observation literals a and b where absence is taken as meaning "undetermined".

reduced the number of possible successor states that are generated at any one time. For example, if the rules r_3 and r_4 had been defined as

$$r_3 : a, r_2 \multimap b \mid r_4 \qquad\qquad r_4 : \multimap b \mid r_4$$

the rule activations for step 7 would be $\{\{b, r_0, r_1, r_2, r_3\}, \{r_0, r_1, r_2, r_3, r_4\}\}$, yielding merged states $\{\{\neg a, b, r_0, r_1, r_2, r_3\}, \{\neg a, b, r_0, r_1, r_2, r_3, r_4\}\}$. Then, step 8 would have had $\{\{r_0, r_1, r_3\}, \{b, r_0, r_1, r_3\}, \{r_0, r_1, r_3, r_4\}\}$ for rule activations and $\{\{\neg a, \neg b, r_0, r_1, r_3\}, \{\neg a, \neg b, r_0, r_1, r_3, r_4\}\}$ for its merged states, one of which does not contain the noted (minimal) rule r_4 and so the observation trace, as is to be expected, satisfies the rule set.

2.1 Inhibiting Rule Activation

The informal semantics we've used above has rules being activated in the next step if they appear positively in some applied consequent of some currently applicable rule. In particular, rules that are not mentioned in a consequent of some rule can not be activated by that rule; however, some other rule may indeed activate them. Consider, for example, the contrived (sub)set of rules below.

$$r_0 : \multimap r_2 | r_3 \qquad\qquad r_1 : \multimap r_3 | r_4$$

Assume at some stage that r_0 and r_1 are activated in the same step. Rule r_0 therefore generates the partial successor states $\{r_2\}$ and $\{r_3\}$. Rule r_1 will then extend these states to yield the possible (partial) states $\{r_2, r_3\}$, $\{r_2, r_4\}$, $\{r_3\}$ and $\{r_3, r_4\}$. Suppose it was desired that rules r_2 and r_3 were mutually exclusive. One way would be to modify the rules as below.

$$r_0 : \multimap r_2, \neg r_3 | r_3, \neg r_2 \qquad\qquad r_1 : \multimap r_3 | r_4$$

Assuming again both r_0 and r_1 active, the possible successor activation sets are now $\{r_2, \neg r_3, r_4\}$, $\{\neg r_2, r_3\}$ and $\{\neg r_2, r_3, r_4\}$ — since the potential rule activation set $\{r_2, \neg r_3, r_3, \neg r_2\}$ is inconsistent. The negation of a rule should be interpreted as a forced "non-activation" of the rule.

In the examples above, we indicated how various temporal conditions could be translated into collections of these low-level single-shot (or step?) rules. In a certain sense, rule names can be viewed as propositions denoting temporal subformulas. However, it is important to emphasise that a negated rule name does not correspond to the negation of a subformula that the rule name may be viewed as representing. More strictly, one should view a positive occurrence of a rule name as meaning that the rule will be applied and in doing so will generate possible traces that satisfy the associated subformula. A negative occurrence of a rule name (in the rule activation state) simply means that the rule is NOT applied and hence places no constraints on the generation of traces.

In summary, we can use rules to activate other rules (positive appearance of a rule in a consequent), to not inhibit activation (no mention of a rule in a consequent), and to inhibit activation (negative appearance of a rule in a consequent).

3 Propositional RULER Trace Semantics

We now present a formalization of propositional rule systems and an evaluation semantics over traces of observations.

Preliminary definitions. Let X denote a set of atoms. We then use X^- to denote the set of negated atoms of X, i.e. $X^- = \{\neg x \mid x \in X\}$, and let $X^\pm$ denote the set of literals of X, i.e. $X \cup X^-$. We use the term X-literal to refer to a member of $X^\pm$. A set of X-literals L is said to be *self-consistent* if and only if for any $x \in X$ it is not the case that both $x \in L$ and $\neg x \in L$. Let L^{-*}_X denote the negative closure of L with respect to the atoms X, i.e. the set $L \cup \{\neg l | l \notin L, l \in X\}$. Given LS_1 and LS_2 as sets of self-consistent sets of literals, the product $LS_1 \times LS_2$ is the set $\{ls_1 \cup ls_2 \mid ls_1 \in LS_1, ls_2 \in LS_2,$ and $ls_1 \cup ls_2$ is self-consistent$\}$.

Rule Systems. Given disjoint sets of rule names R and observations O, a *rule* ρ is a pair $\langle C, B\rangle$ where C, the condition part, is a conjunctive set of $(R \cup O)$-literals, and B, the body part, is a disjunctive set of conjunctive sets of $(R \cup O)$-literals. A *named rule* is then an association $r : \rho$ where $r \in R$ is a rule name and ρ is a rule. A *rule system* RS is a tuple $\langle R, O, P, I, F\rangle$ where R and O are, respectively, disjoint sets of rule names and observations, and P is a set of disjointly R-named rules over R and O, $I \subseteq O^\pm \cup R$ is a self-consistent subset of observation literals and rule names, and $F \subseteq R$ is a set of terminally excluded rule names (rule names that may not appear in the very final monitoring state). A *configuration* γ for a rule system RS is a pair $\langle A, \Theta\rangle$ where A is a consistent set of R-literals, called the activity set, and Θ is a consistent set of O-literals, called the observation state. We also write $A(\gamma)$ to denote the activity set of a configuration γ, similarly $\Theta(\gamma)$ for the observation state.

We next define the interpretation of a set of literals in a configuration. The presence of a positively signed rule name r in the activity set means that the rule ρ associated with r is active. On the other hand, the presence of a negatively signed rule name r, or the absence of r, in the activity set means that the rule ρ associated with r is not active. For observation atoms, however, undefinedness of an O-literal o, i.e. the absence of o from the observation state of the configuration, means that the observation literal o may be either true or false.

Modelling and step relation. Let $RS = \langle R, O, P, I, F\rangle$ be a rule system. A self-consistent set of literals L from RS holds in a configuration γ for RS, which is denoted by $\gamma \models L$, if and only (i) the set of rule name literals mentioned in L is contained in the negative closure of $A(\gamma)$, i.e. $(L - O^\pm) \subseteq A(\gamma)^{-*}_R$, and (ii) observation literals within L are contained in the configuration's set of observations $(L - R^\pm) \subseteq \Theta(\gamma)$. We can now define a single step relation over configurations for a given named rule. This relation can then be used to define a single step relation for a rule system. An $r{:}\rho$*-step relation* $\xrightarrow{r:\rho}$ between configurations is such that $\gamma \xrightarrow{r:\rho} \gamma'$ if and only if (i) $r \in A(\gamma)$, (ii) $\gamma \models C(\rho)$, and (iii) there is a $\theta \in B(\rho)$ such that $A(\gamma') \cup \Theta(\gamma') = \theta$. Then for a set of rule names R, let Γ' be an R*-indexed set of outcome configurations* such that for each

$r \in R$, $\gamma \xrightarrow{r:\rho} \Gamma'_r$. We then define the step relation $\longrightarrow$ between configurations such that $\gamma \longrightarrow \gamma'$ if and only γ' is a consistent union of an $(A(\gamma) \cap R)$-indexed set of outcome configurations from γ. Note that an empty union set is treated as being an inconsistent union.

The single step relation for the rule system can now be used to define the notion of an accepting run of a rule system over a given observation trace. This requires matching obligations against actual observations. As we have adopted a classical interpretation for observation literals (which is not the case for rule name literals), we thus have the following.

Matching. An actual set of observation literals X is said to *match* an obligatory set of literals Y if and only if $X \cup Y$ is self-consistent and $Y \subseteq X$.

Finally, we can define the language accepted by a rule system.

Language acceptance. An accepting run of a rule system $RS = \langle R, O, P, I, F\rangle$ on an observation trace $\tau = o_1 o_2 \ldots o_n$ is a sequence of configurations $\gamma_1\gamma_2 \ldots \gamma_n\gamma_{n+1}$ such that (i) $A(\gamma_1) \in I$, (ii) for all $i \in 1..n$ the actual and obligated observations, o_i and $\Theta(\gamma_i)$ respectively, match and $\langle A(\gamma_i), \Theta(\gamma_i) \cup o_i\rangle \longrightarrow \gamma_{i+1}$, and (iii) $A(\gamma_{n+1}) \cap F = \{\}$. Thus, the language accepted by a rule system RS, $\mathcal{L}(RS)$, is the set of all finite observation traces τ accepted by RS. Furthermore, we say a rule system RS is violated by an observation trace τ if RS has no accepting run on τ, alternatively, $\tau \notin \mathcal{L}(RS)$.

We now claim that the monitoring algorithm of Section 2 accepts an observation trace τ for a rule system RS if and only if $\tau \in \mathcal{L}(RS)$. Indeed, the steps of the algorithm closely reflect the semantic construction we have given.

3.1 Propositional Linear Temporal Logic as a Rule System

Having formally defined propositional rule systems, we are now in a position to show how linear-time temporal logic formulas for monitoring over finite traces can be encoded in RuleR. Our translation is based on the separation result of Gabbay (originally 1981 but elaborated in [8]), which can then be used to show that any mixed past, present and future linear-time temporal formula can be translated into a collection of universal implications of the form *non-strict past formula* implies *pure future formula*, a minor variation of the rule forms used in the executable temporal logic MetateM [2]. Our starting point is thus to show how such separated temporal implications can be represented in RuleR.[4]

The Pure Future Part. The pure future linear-time temporal formulas are built from propositions, the boolean connectives and, or, and negation, $\wedge$, $\vee$ and $\neg$, respectively, and a strict until and unless operator, $\mathcal{U}^+$ and $\mathcal{W}^+$. All other standard future time operators are definable from this set. Without loss

[4] Fisher's SNF representation for temporal logic [6] is close to RuleR rule forms and an alternative translation to a rule system could be given via SNF. However, we believe our direct translation has interest in its own right and might lead to an easier SNF translation.

of generality, we assume formulas are further transformed in negation normal form (NNF[5]), i.e. negation operators pushed inwards to propositional literals and cancellations applied. Let WFF^+ denote the set of well-formed strict future time formulas in NNF and WFF denote the set of well-formed future time formulas in NNF (which may include the present, i.e. propositions under no future time operator).

We define a translation $\overrightarrow{T} : WFF \rightarrow RuleSystem$ inductively over the structure of the temporal formulas. Let ϕ and ψ denote arbitrary members of WFF. The base cases of the translation are straightforward, e.g. for an atom p, we have $\overrightarrow{T}(p) = \langle \{\}, \{p\}, \{\}, \{\{p\}\}, \{\} \rangle$, indicating a rule system with an atom p with an initial set of active rule names containing the singleton set $\{p\}$. Negated atoms translate in a similar way. The propositional constant **true** gives rise to the rule system $\langle \{\}, \{\}, \{\}, \{\{\}\}, \{\} \rangle$ whereas **false** translates to a system with an empty set of initial states. As one might expect, the logical conjunction (disjunction) of formulas ϕ and ψ translate to the obvious product (union) operations that can be defined for rule systems. This leaves the most interesting part of the translation, namely an until formula $\phi \mathcal{U}^+ \psi$. Recall that the semantics of the strict until operator gives the temporal equivalence $\phi \mathcal{U}^+ \psi \Leftrightarrow \bigcirc(\psi \vee (\phi \wedge (\phi \mathcal{U}^+ \psi)))$.

$$
\begin{aligned}
\overrightarrow{T}(\phi \mathcal{U}^+ \psi) = \textsc{let}\ & \langle R_\phi, O_\phi, P_\phi, I_\phi, F_\phi \rangle = \overrightarrow{T}(\phi)\ \textsc{and} \\
& \langle R_\psi, O_\psi, P_\psi, I_\psi, F_\psi \rangle = \overrightarrow{T}(\psi) \\
\textsc{in}\ & \langle R_\phi \cup R_\psi \cup \{r_{\phi \mathcal{U}^+ \psi}\}, \\
& O_\phi \cup O_\psi, \\
& P_\phi \cup P_\psi \cup \{r_{\phi \mathcal{U}^+ \psi} :\!\multimap\ I_\psi \cup (I_\phi \times \{\{r_{\phi \mathcal{U}^+ \psi}\}\}), \\
& \{\{r_{\phi \mathcal{U}^+ \psi}\}\}, \\
& F_\phi \cup F_\psi \cup \{r_{\phi \mathcal{U}^+ \psi}\} \rangle
\end{aligned}
$$

For ease of understanding, we have subscripted the rule names by the subformulas they represent. As the until operator has a strong interpretation, requiring its second argument to be satisfied, the associated rule name for the until formula must be included in the F set of the rule system. As might be expected, the translation of the unless formula differs from the until translation just in the non-inclusion of the rule for the unless formula in the F set.

Example 2. Assume a, b, c and d are atomic propositions. The translation of $a \mathcal{U}^+ b$ yields the rule system

$$\langle \{r_{a \mathcal{U}^+ b}\}, \{a, b\}, \{r_{a \mathcal{U}^+ b} : \multimap\ b \mid a, r_{a \mathcal{U}^+ b}\}, \{\{r_{a \mathcal{U}^+ b}\}\}, \{r_{a \mathcal{U}^+ b}\} \rangle$$

Similarly, the translation of $a \wedge (c \mathcal{W}^+ d)$ yields the rule system

$$\langle \{r_{c \mathcal{W}^+ d}\}, \{a, c, d\}, \{r_{c \mathcal{W}^+ d} : \multimap\ d \mid c, r_{c \mathcal{W}^+ d}\}, \{\{a, r_{c \mathcal{W}^+ d}\}\}, \{\} \rangle$$

Thus the translation of $(a \mathcal{U}^+ b) \mathcal{U}^+ (a \wedge (c \mathcal{W}^+ d))$ yields the rule system

$$\langle \{r_0, r_1, r_2\}, \{a, b, c, d\}, \left\{ \begin{array}{l} r_0 : \multimap\ b \mid a, r_0 \\ r_1 : \multimap\ d \mid c, r_1 \\ r_2 : \multimap\ a, r_1 \mid r_0, r_2 \end{array} \right\}, \{\{r_2\}\}, \{r_0, r_2\} \rangle$$

[5] Some authors refer to this as positive normal form.

where

$$r_0 = r_{a\,\mathcal{U}^+\,b}, \quad r_1 = r_{c\,\mathcal{W}^+\,d}, \quad r_2 = r_{(a\,\mathcal{U}^+\,b)\,\mathcal{U}^+\,(a\wedge(c\,\mathcal{W}^+\,d))}$$

Past Time Temporal Queries. The pure past time fragment of linear-time temporal logic is constructed in a mirror fashion to the pure future part, i.e. from propositions, the boolean connectives ($\wedge$, $\vee$ and $\neg$), and just the temporal operators $\mathcal{S}^-$ (the strict *since*, false at the beginning of time) and its weak version $\mathcal{Z}^-$ (true at the beginning of time). Without loss of generality, we assume that past time temporal formulas are in negation normal form, i.e. with negations pushed inwards to atomic propositions/literals and double negations cancelled. Let us first consider the translation of pure past time temporal queries. The temporal equivalence $\phi\,\mathcal{S}^-\psi \Leftrightarrow \bigodot(\psi \vee (\phi \wedge (\phi\,\mathcal{S}^-\psi)))$ should serve as a reminder of the semantics that needs to be captured by the translation. The basic idea for handling the past is an old one, namely, we use the translation rules to calculate the value of the temporal query as we proceed in time (rather than evaluating the query over the history). We will use the presence of the rule name $r_{\phi\,\mathcal{S}^-\psi}$ in the rule activation state to denote whether the temporal formula $\phi\,\mathcal{S}^-\psi$ held in the previous moment (similarly for r_ϕ and r_ψ). We then use a rule, named $r_{\psi:\psi\,\mathcal{S}^-\phi?}$, to calculate whether $r_{\phi\,\mathcal{S}^-\psi}$ should be made active because ψ held in the previous moment (similarly for the other possible way for $\psi\,\mathcal{S}^-\phi$ to hold). These query rules must be universally active in order to determine truth values for the next moment. Thus we use a rule, named say $r_{g.\phi\,\mathcal{S}^-\psi?}$, to act as a generator (hence the "g" in its name) for a pair of (sets of) rules that determine the truth of $\phi\,\mathcal{S}^-\psi$ based on the previous values of its subformulas.

$$r_{\psi.\phi\,\mathcal{S}^-\psi?} : r_\psi \multimap r_{\phi\,\mathcal{S}^-\psi} \qquad r_{\phi.\phi\,\mathcal{S}^-\psi?} : r_\phi, r_{\phi\,\mathcal{S}^-\psi} \multimap r_{\phi\,\mathcal{S}^-\psi}$$
$$r_{g.\phi\,\mathcal{S}^-\psi?} : \multimap r_{g.\phi\,\mathcal{S}^-\psi?}, r_{\phi.\phi\,\mathcal{S}^-\psi?}, r_{\psi.\phi\,\mathcal{S}^-\psi?}$$

Naturally, our translation must take into account the fact that the subformulas ψ and ϕ may be boolean combinations of pure past time temporal formulas (represented by rule names) and/or literals. Let WFF^- denote the set of pure past temporal formulas and WFF^{-0} the set of present and pure past time temporal formulas. We thus define a translation $\overleftarrow{T}$ that will translate a past time temporal formula (from WFF^{-0}) into an intermediate form (of a rule system) whose initial activation set, as a disjunctive set of conjunctive sets of rule names and/or literals, is to be viewed as representing the given temporal query. The difference from a proper rule system is that we use the F set to represent the initial values of rules, e.g. a formula $\phi\,\mathcal{S}^-\psi$ must be false initially and so the rule name $\neg r_{\phi\,\mathcal{S}^-\psi}$ would be included in the set F. As with the future time translation the base cases are clear, as is conjunction and disjunction. Figure 2 shows the translation for the interesting case of the strict since operator.

Separated Temporal Implicative Forms. We can now bring together the above two translations $\overrightarrow{T}$ and $\overleftarrow{T}$ to generate a rule system corresponding to the MetateM-like rule form $\phi_{\text{past}} \Rightarrow \bigcirc\psi_{\text{future}}$ which are of universal nature,

$$
\begin{aligned}
&\overleftarrow{T}(\phi\,\mathcal{S}^{-}\psi) = \\
&\quad \text{LET } \langle R_\phi, O_\phi, P_\phi, I_\phi, F_\phi\rangle = \overleftarrow{T}(\phi) \text{ AND} \\
&\qquad \langle R_\psi, O_\psi, P_\psi, I_\psi, F_\psi\rangle = \overleftarrow{T}(\psi) \\
&\quad \text{IN } \langle R_\phi \cup R_\psi \cup \{r_{\phi\,\mathcal{S}^{-}\psi}, r_{g.\phi\,\mathcal{S}^{-}\psi?}\} \cup \{r_{\phi\,\mathcal{S}^{-}\psi?x} \mid x \in I_\phi\} \cup \{r_{\phi\,\mathcal{S}^{-}\psi?x} \mid x \in I_\psi\}, \\
&\qquad O_\phi \cup O_\psi, \\
&\qquad P_\phi \cup P_\psi \cup \\
&\qquad\quad \{r_{g.\phi\,\mathcal{S}^{-}\psi?} : \multimap \{r_{g.\phi\,\mathcal{S}^{-}\psi?}\} \cup \{r_{\phi\,\mathcal{S}^{-}\psi?x} \mid x \in I_\phi\} \cup \{r_{\phi\,\mathcal{S}^{-}\psi?x} \mid x \in I_\psi\}\} \cup \\
&\qquad\quad \{r_{\phi\,\mathcal{S}^{-}\psi?x} : x \multimap r_{\phi\,\mathcal{S}^{-}\psi} \mid x \in I_\psi\} \cup \\
&\qquad\quad \{r_{\phi\,\mathcal{S}^{-}\psi?x} : x, r_{\phi\,\mathcal{S}^{-}\psi} \multimap r_{\phi\,\mathcal{S}^{-}\psi} \mid x \in I_\phi\}, \\
&\qquad \{\{r_{\phi\,\mathcal{S}^{-}\psi}, r_{g.\phi\,\mathcal{S}^{-}\psi?}\}\}, \\
&\qquad F_\phi \cup F_\psi \cup \{\neg r_{\phi\,\mathcal{S}^{-}\psi}\}\rangle
\end{aligned}
$$

Fig. 2. Translation of $\phi\,\mathcal{S}^{-}\psi$

i.e. globally hold. Assuming both ϕ_{past} and ψ_{future} are in a negation normal form, then, in the context of

$$
\begin{aligned}
&\langle R_{\text{past}}, O_{\text{past}}, P_{\text{past}}, I_{\text{past}}, F_{\text{past}}\rangle = \overleftarrow{T}(\phi_{\text{past}}) \\
&\langle R_{\text{future}}, O_{\text{future}}, P_{\text{future}}, I_{\text{future}}, F_{\text{future}}\rangle = \overrightarrow{T}(\psi_{\text{future}})
\end{aligned}
$$

in which we assume, without loss of generality, the rule name sets are disjoint, the rule system below will represent the translation of the separated implicative form, i.e. $RS = T(\phi_{\text{past}} \Rightarrow \bigcirc\psi_{\text{future}})$.

$$
\begin{aligned}
RS = \langle &R_{\text{past}} \cup R_{\text{future}} \cup \{r_{g.\phi_{\text{past}} \Rightarrow \psi_{\text{future}}}\} \cup \{r_{x \Rightarrow \psi_{\text{future}}} \mid x \in I_{\text{past}}\}, \\
&O_{\text{past}} \cup O_{\text{future}}, \\
&P_{\text{past}} \cup P_{\text{future}} \cup \\
&\qquad \{r_{g.\phi_{\text{past}} \Rightarrow \psi_{\text{future}}} : \multimap \{r_{g.\phi_{\text{past}} \Rightarrow \psi_{\text{future}}}\} \cup \{r_{x \Rightarrow \psi_{\text{future}}} \mid x \in I_{\text{past}}\}\}, \\
&\qquad \{r_{x \Rightarrow \psi_{\text{future}}} : x \multimap I_{\text{future}} \mid x \in I_{\text{past}}\}, \\
&\{\{r_{g.\phi_{\text{past}} \Rightarrow \psi_{\text{future}}}, r_{x \Rightarrow \psi_{\text{future}}} \mid x \in I_{\text{past}}\} \cup F_{\text{past}}\}, \\
&F_{\text{future}}\rangle
\end{aligned}
$$

Example 3. Assuming a, b, c, p and q denote propositions, we give the RULER translation of the universal separated temporal implication

$$c \wedge (b\,\mathcal{S}^{-}a) \;\Rightarrow \bigcirc(\Diamond p \wedge \Diamond q).$$

Recall that $\Diamond p$ will be translated as $p \vee \Diamond^{+} p$, i.e. $p \vee \mathbf{true}\,\mathcal{U}^{+} p$, similarly for $\Diamond q$. Using the following abbreviations

$$
\begin{aligned}
&r_0 = r_{g.b\,\mathcal{S}^{-}a?} \qquad r_1 = r_{b\,\mathcal{S}^{-}a?b} \qquad r_2 = r_{b\,\mathcal{S}^{-}a?a} \qquad r_3 = r_{b\,\mathcal{S}^{-}a} \\
&r_4 = r_{\mathbf{true}\,\mathcal{U}^{+}p} \qquad r_5 = r_{\mathbf{true}\,\mathcal{U}^{+}q} \qquad r_6 = r_{g.c \wedge (b\,\mathcal{S}^{-}a) \Rightarrow \bigcirc((p \vee \mathbf{true}\,\mathcal{U}^{+}p) \wedge (q \vee \mathbf{true}\,\mathcal{U}^{+}q))} \\
&r_7 = r_{c \wedge (b\,\mathcal{S}^{-}a) \Rightarrow \bigcirc((p \vee \mathbf{true}\,\mathcal{U}^{+}p) \wedge (q \vee \mathbf{true}\,\mathcal{U}^{+}q))}
\end{aligned}
$$

the rule system will thus have rules

$$
\begin{aligned}
&r_0 : \multimap r_0, r_1, r_2 \qquad r_1 : b, r_3 \multimap r_3 \qquad r_2 : a \multimap r_3 \\
&r_3 : \qquad r_4 : \multimap p \mid r_4 \qquad r_5 : \multimap q \mid r_5 \\
&r_6 : \multimap r_6, r_7 \qquad r_7 : c, r_3 \multimap p, q \mid p, r_5 \mid r_4, q \mid r_4, r_5
\end{aligned}
$$

with an initial rule activation set as $\{\{\neg r_3, r_0, r_1, r_2, r_6, r_7\}\}$ and the forbidden rule set as $\{r_4, r_5\}$.

The correctness of our translation scheme for propositional LTL over finite traces with respect to the given semantics for RuleR follows from the correctness of separation, then an inductive proof establishing the correctness of the translation of the universal separated implicative temporal forms.

4 Parameterized RuleR

The propositional RuleR system corresponds to regular-based languages, which are a subclass of propositional Eagle. Here, we extend RuleR to include rule definitions parameterized by rules. The evaluation strategy used on this seemingly small extension increases the formal expressivity of RuleR to be beyond context-free languages. Consider the following rule definition, indeed schema, that has been extended to include formal rule arguments.

$$r(\rho) : a \multimap b, \rho \mid c, r(\rho)$$

Suppose that the rule r is active with the propositional rule r_0 substituted for ρ, i.e. $r(r_0)$ is active. Informally, the evaluation of $r(r_0)$ will first determine the truth of the condition part a, then, assuming it holds, continue to create a set of activation states for the next step corresponding to $\{\{b, r_0\}, \{c, r(r_0)\}\}$. Let us give a few examples that show how the expressivity of rule parameterized RuleR jumps into the context sensitive languages.

Example 4. Consider a rule system with the rules

$$\begin{array}{ll} r_b(\rho) : \multimap b, \neg a, \rho & r_{ab}(\rho) : \multimap b, \neg a, \rho \mid a, \neg b, r_{ab}(r_b(\rho)) \\ r_{end} : \multimap r_{fail} & r_{fail} : \multimap r_{fail} \end{array}$$

together with an initial rule activation set as $\{\{a, r_{ab}(r_{end})\}\}$ and the final forbidden rule set $\{r_b, r_{ab}, r_{fail}\}$ (meaning that no occurrence of rule r_b, r_{ab}, nor r_{fail}, may appear as an obligation in a final rule activation state). All accepted observation traces will match against a trace of $n \geq 1$ occurrences of a followed by n occurrences of b. Essentially, barring the first a, the rule r_{ab} represents the non-terminal S of the context free grammar $S = ab \mid aSb$ in which r_{ab}'s actual argument represents the continuation string for concatenation to the string of a's generated. It is straightforward to establish that the class of context free languages are a subset of parameterized RuleR. We extend the above example to represent traces of the form $a^n b^n c^m$, for $n, m \geq 1$. Take the rule set

$$\begin{array}{l} r_{ab}(\rho) : \multimap b, \neg a, \neg c, \rho \mid a, \neg b, \neg c, r_{ab}(r_b(\rho)) \\ r_b(\rho) : \multimap b, \neg a, \neg c, \rho \\ r_c : \multimap c, \neg a, \neg b, r_{end} \mid c, \neg a, \neg b, r_c \\ r_{end} : \multimap r_{fail} \\ r_{fail} : \multimap r_{fail} \end{array}$$

together with an initial activation set as $\{\{a, r_{ab}(r_c)\}\}$ and the final forbidden rule activation set $\{r_{ab}, r_b, r_c, r_{fail}\}$. This system will clearly accept traces of the

form $a^n b^n$ (represented by the r_{ab} rule) followed by one or more c's (determined by the r_c argument to the initial rule activation r_{ab}). Now we can encode the intersection of the languages $a^n b^n c^m$ and $a^m b^n c^n$ $(n, m \geq 1)$, thus yielding the context sensitive language containing words of the form $a^n b^n c^n$.

$$
\begin{array}{l}
r_{ab}(\rho) : \multimap b, \neg a, \neg c, \rho \mid a, \neg b, \neg c, r_{ab}(r_b(\rho)) \\
r_b(\rho) : \multimap b, \neg a, \neg c, \rho \\
r_c : \multimap c, \neg a, \neg b \mid c, \neg a, \neg b, r_c \\
\\
r_a(\rho) : \multimap b, \neg a, \neg c, \rho \mid a, \neg b, \neg c, r_a(\rho) \\
r_{bc}(\rho) : \multimap c, \neg a, \neg b, \rho \mid b, \neg a, \neg c, r_{bc}(r_{c1}(\rho)) \\
r_{c1}(\rho) : \multimap c, \neg a, \neg b, \rho \\
\\
r_{end} : \multimap r_{fail} \\
r_{fail} : \multimap r_{fail}
\end{array}
$$

Here the rule system has an initial activation set $\{\{a, r_{ab}(r_c), r_a(r_{bc}(r_{end}))\}\}$ and final forbidden rule activation set $\{r_{ab}, r_b, r_c, r_a, r_{bc}, r_{c1}, r_{fail}\}$.

As in EAGLE, we can also parameterize RULER rules by data values, thus introducing variables and predicated atoms. It is through such means that RULER can be used for encoding/interpreting real-time and stochastic logics.

Example 5. Let us assume that each observation state is time-stamped by the unique presence of a grounded predicate $clock(t)$ for some real value t, e.g. $clock(49738.22264)$. The data parameterized rule schema

$$
\begin{array}{rl}
r(k : \mathbb{R}) : clock(?n : \mathbb{R}) \multimap & clock(?t : \mathbb{R}), p, t - n < k \mid \\
& clock(?t : \mathbb{R}), \neg p, t - n < k, r(k - t + n)
\end{array}
$$

defines a constraint that the atom p must be consistent with an observation state within k time units from the observation state in which the rule $r(k)$ is required to hold. The $?n : \mathbb{R}$ appearing as argument to the *clock* predicate name in the rule's condition means that the variable n is to be bound to some value from $\mathbb{R}$ by the current observation state. The occurrence of $clock(?t : \mathbb{R})$ in the rule consequent means that there is an obligation on the next observation state to binding t with some value. Suppose we have an observation state containing just $\{clock(1), \neg p, r(3)\}$. The rule $r(3)$, through binding n to 3, gives rise to the set $\{\{clock(?t : \mathbb{R}), p, t - 1 < 3\}, \{clock(?t : \mathbb{R}), \neg p, t - 1 < 3, r(4 - t)\}\}$. If the next actual observation state is $\{clock(3), \neg p\}$, the merge with the obligation sets yields the frontier set $\{\{clock(3), \neg p, r(1)\}\}$, which gives rise, through $r(1)$, to obligations $\{\{clock(?t : \mathbb{R}), p, t - 3 < 1\}, \{clock(?t : \mathbb{R}), p, t - 3 < 1, r(4 - t)\}\}$. If we have another observation this time with $\{clock(4), p\}$ then the merge yields the empty frontier set as $4 - 3 < 1$, which appears in both possible futures, is clearly false. Hence the actual behaviour does not conform to that required by the initial $r(3)$. On the other hand, had the observation state been, say, $\{clock(3.9), p\}$, then the rule set would be satisfied.

5 Conclusions

We have introduced a low-level rule system RULER as a kind of "byte-code" for run-time monitoring logics. A basic monitoring algorithm was described for

the propositional subset of RuleR. Having presented formally the semantics of the propositional subset, we demonstrated how linear time temporal logic with both past and future operators can translate to such rule systems, and then briefly, and informally, presented RuleR where rules are parameterized by rule names. On the face of it, the propositional subset of RuleR looks rather like a grammatical representation of the transition relation of an alternating automaton, i.e. with conjunctive and disjunctive branching, see for example [7]. However, RuleR, even the propositional subset, has more to it; the rules have the capability to switch other rules on or off as an evaluation of a rule system over a trace proceeds. We are referring to such systems as reactive rule systems / grammars / Kripke structures [9]. Whilst regular grammars are closed under our notion of reactivity (including switching grammar rule sets), it can easily be shown that reactive context free grammars take us beyond context free. Some relationship with state-alternating context-free grammars [10] is clear, however, a more detailed study of reactive grammars and their place in the complexity hierarchy is work in progress, see [5] for some initial results and examples. A feature we haven't yet mentioned is rule priority in RuleR. Given the ability to switch rules on and off, conflicts may occur. Sometimes the conflicts may be desired, but in other situations we may wish one rule to override another, as is the case in defeasible reasoning. Of course, this changes the nature of the logics expressible quite considerably. In addition to rule parameters, RuleR has data parameters, just as in Eagle. The semantic details are not difficult and we adopt an approach similar to that in first-order MetateM [1], but, as in Eagle, some care needs to be taken to avoid "rule activation state set" explosion in practice.

The low-level simplicity of RuleR leads to the main advantage for its potential use over Eagle. If optimal (asymptotic) complexity bounds have been established for a particular subset logic of Eagle, such as for the LTL subset, in general a RuleR encoding will be no better asymptotically. However, we assert that smaller constants arise through the significant reduction in the symbolic processing that has to be undertaken at run-time in the interpretation of Eagle formulas. Of course, there would be a one-off translation cost from the LTL formula to the appropriate rule systems. This a compilation versus interpretation gain.

A prototype Java implementation of the monitoring algorithm for propositional RuleR has been developed, as a proof of concept. We are, however, not yet at the stage where we can properly evaluate the practical effectiveness of RuleR, which requires the fully parameterized version of RuleR. We hope to report on this in the near future.

References

1. Barringer, H., Fisher, M., Gabbay, D., Owens, R., Reynolds, M.: The Imperative Future: Principles of Executable Temporal Logic. Research Studies Press (1996)
2. Barringer, H., Fisher, M., Gabbay, D., Gough, G., Owens, R.: An Introduction. In: Formal Aspects of Computing, vol. 7(5), pp. 533–549. Springer, London (1995)

3. Barringer, H., Goldberg, A., Havelund, K., Sen, K.: Rule-Based Runtime Verification. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, Springer, Heidelberg (2004)
4. Barringer, H., Goldberg, A., Havelund, K., Sen, K.: Run-time Monitoring in Eagle. In: Brunnstein, K., Händler, W., Haefner, K. (eds.) RGU 1974. LNCS, vol. 17, p. 264. Springer, Heidelberg (2004)
5. Barringer, H., Rydeheard, D., Gabbay, D.: Reactive Grammars: An Initial Exploration, Draft paper(2007), see `http://www.cs.man.ac.uk/~david/reactive.html`
6. Fisher, M.D.: A Normal Form for Temporal Logics and its Applications in Theorem-Proving and Execution. Journal of Logic and Computation 7(4), 429–456 (1997)
7. Finkbeiner, B., Sipma, H.: Checking Finite Traces Using Alternating Automata. Formal Methods in System Design 24(2), 101–127 (2004)
8. Gabbay, D.M.: Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In: Banieqbal, B., Pnueli, A., Barringer, H. (eds.) Temporal Logic in Specification. LNCS, vol. 398, pp. 67–89. Springer, Heidelberg (1989)
9. Gabbay, D.M.: Introducing Reactive Kripke Semantics and Arc Accessibility. In: Gabbay, D.M. (ed.) To appear in Festschrift in Honour of Boaz Traktenbrot (2007)
10. Moriya, E., Hofbauer, D., Huber, M., Otto, F.: On State-Alternating Context-Free Grammars. Theoretical Computer Science 337(11), 183–216 (2005)