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. EMAIL: {Howard.Barringer, David.Rydeheard}@manchester.ac.uk
[2] NASA's Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA. EMAIL: Klaus.Havelund@jpl.nasa.gov

Abstract. In [5], 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, 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. As a formal demonstration, we provide a translation scheme for LTL with a proof of translation correctness. We then describe a parameterized form of RULER in which rule names may have data and rule parameters, thus allowing a richer collection of logics to be compiled and more compact translations.

Keywords Run-time verification, rule systems, temporal logic, grammars.

1 Introduction

In earlier work, the rule-based temporal logic EAGLE [5] 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

[2] 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.

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 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.

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

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

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

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

[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.

temporal operators such as *happens before real time u*.

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

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

We continue to believe that EAGLE presents a natural rule/equation based language for defining, even programming, monitors for complex temporal behavioural 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 Cs or Rs in order to give a positive result. Later work [2] 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 [5] 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 [6]. 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 formal semantic treatment and the following section provides a translation scheme for compiling propositional temporal logic (with past and future operators) into RULER. Although not presented, translations from other trace descriptions, e.g. regular expressions and finite state automata, have been undertaken. In Section 5, we consider RULER where rule names may be parameterized by both rule and data parameters. We present, using just rule parameters, examples of context-free and context-sensitive languages. More efficient evaluation can be achieved via encoding with data as well as rules and to that extent we show how context-free temporal logics such as the CaRet temporal logic of nested procedure calls and returns can be compiled into RuleR. The paper concludes in Section 6 with a brief reflection and indication of further work.

2 RuleR by example

A RuleR monitoring system, or rule system, comprises a set of rule names, a set of observation names, a collection of named rules, a set of possible initial states and a set of terminally excluded rule names. 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 of 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 simple example, consider the rule named r below in the context of some observation named a.

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

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 against a set of named rules defined by a rule system. Essentially, the algorithm explores in a breadth-first fashion the traces allowed by the rule system against the given observation trace. The algorithm thus unfolds all the active rules according to the given input observation states. As rule bodies are disjunctive, the algorithm computes sets of possible future states, which we call a frontier, 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. Referring to Figure 1, line 1 creates an initial frontier using the set of initial states of the rule system. Lines 3 to 8 program a single evaluation step of the rule system. In line 4, the current set of observations (obtained in line 3), are unioned with each of the rule activation states of the frontier. All inconsistent sets are deleted from the frontier and then, in line 5, a monitoring exception is raised if the frontier has become empty indicating failure of the rule system. Lines 6 to 8 then build the next frontier. For each consistent resultant state, say s, in the frontier (from the action at line 4), a set of possible successor states is created using the consequents of each applicable rule of s (line 7). The collection of sets of possible successors are then unioned, line 8, to form the next frontier of rule activation states. The evaluation algorithm reapplies this basic step until either the input is exhausted or a monitoring exception has been raised. For normal termination, line 10, the final frontier is then checked for an acceptable final state, i.e. one not containing a forbidden rule name.

```
1: create an initial frontier of rule activation states
2: WHILE input observations exist DO
3:   obtain the next set of observations
4:   add observations across the frontier of rule activation states
5:   raise monitor exception if there's no consistent resultant state
6:   FOREACH of the current and consistent resultant states,
7:     use all active rules to create a successor set of activation states
8:   union successor sets for the next frontier of rule activation states
9: OD

10: yield success iff last frontier has an acceptable final state
```

Fig. 1. The basic monitoring algorithm

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 consistent sets of literals[2].

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 $\Box((a \wedge \odot a) \Rightarrow \oplus b)$ where $\oplus$ is the strict "eventually in the future" temporal operator, or using the EAGLE temporal predicates defined in Section 1 by the monitoring formula $\texttt{Always}((a \wedge \odot a) \Rightarrow \bigcirc\texttt{Sometime}(b))$. In RULER the following set of rules characterise the required temporal behaviour

$$r_0 : \;-\!\ominus\!\!\rightarrow\; r_0, r_1, r_3 \qquad r_1 : a \;-\!\ominus\!\!\rightarrow\; r_2 \qquad r_2 :$$
$$r_3 : a, r_2 \;-\!\ominus\!\!\rightarrow\; b \mid \neg b, r_4 \qquad r_4 : \;-\!\ominus\!\!\rightarrow\; b \mid \neg b, r_4$$

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

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

[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".

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 the temporal constraint $\odot a$ (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 $\diamondplus b$, either b holds in the next observation state or $\neg b$ holds together with a continued obligation to $\diamondplus b$.

In Table 1 below, we show 8 cycles of the evaluation algorithm on a series of observations for the above rule set. Step 0 corresponds to the initial step of the algorithm (for which we assume no input) in which the frontier of possible rule activation states is initialised to the initial frontier of the rule system (entry in column headed **Rule Activations**); the column headed **Resultant States** then holds the frontier with the observation set added across it (with inconsistent sets removed). The frontier for step 1 (in column 3) is computed as follows. There is just one state in the resultant frontier of step 0. It has rules r_0, r_1 and r_3 active. Rule r_0 unconditionally requires r_0, r_1 and r_3 to be present in the next step. The condition of rule r_1 doesn't hold (a is not present in the resultant state), therefore there are no obligations from this rule. Similarly, r_2 doesn't generate any obligations for the next step. Hence the frontier for step 1 remains as given. For step 1, we have input observations of a and b. These observations are consistent with the obligated rule activation frontier for step 1 and hence can be added to each state in the frontier yielding the set in column 4. This resultant state set is used to generate the obligations for the next step. This time since a is in the state, the rule r_1 will now give rise to rule r_2 in the rule activations for step 2 alongside r_0, r_1 and r_3 unconditionally generated by rule r_0, as before. Moving forwards to step 4 where we can see that both a and $\odot a$ are now true — in the resultant state, both a and r_2 are present — and hence rule r_3 generates two possibilities for step 5. At step 5, the choice with b holding true conflicts with the observation in step 5 and therefore is eliminated from the resultant states (entry in column 4). Rule r_4 is thus active and remains activated until step 7 when b is observed to hold thus conflicting with the requirement of $\neg b$ together with r_4. But how do we determine whether any generated temporal existential

Step	Obs.	Rule Activations	Resultant 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	$\{\neg a, \neg b\}$	$\{\{r_0, r_1, r_3\}\}$	$\{\{\neg a, \neg b, r_0, r_1, r_3\}\}$

Table 1. Steps of a rule system evaluation

obligations, such as $\diamondplus b$, have indeed been satisfied? The rule system structure records, via the forbidden rule name set, those rule names that correspond to such obligations and then, at the end of monitoring, the monitoring algorithm checks whether the final set of merged observation and rule activation states contains states without those recorded 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 recorded rule names, the observation trace satisfies the rule set. The approach is similar to that of the minimal and maximal rule interpretations used in Eagle. For the above example, rule name r_4 is specified as a (terminally) forbidden rule name. The final set of merged observation and rule activation states (resultant states, step 8 in the table) has just one possible state that does not contain the forbidden rule r_4. Hence the given sequence of observations 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 $\neg b \wedge \ldots$. The determinisation thus 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 forbidden rule r_4 and so the observation trace, as is to be expected, satisfies the rule set.

Example 2. A common class of notations for specifying behavior is variants of state machines. In the following is illustrated how state machines can be translated to RuleR. Suppose for example that we want to express the following property as a state machine: "every a must be followed by b without any c in between". We assume that a, b, and c are events and that they exclude each other. We assume furthermore that an execution is terminated with a special event ϵ. This property can be stated as follows in the Rmor state machine notation, defined in [13], and used for specifying and monitoring the behavior of C programs.

```
monitor M {
  state S0 {
    when a -> S1;
  }

  live state S1 {
    when b -> S0;
    when c -> error;
  }
}
```

We will here ignore the issue of defining when the events a, b and c are generated. The state machine consists of two states $S0$, the initial state, and $S1$. Each state defines a block containing the transitions that leave the state. A *live* state is a non-final state; monitoring cannot terminate in a live state. In RMOR a state is by default final. A special *error* state indicates the occurrence of an error, corresponding to entering a non-final state forever. The semantics is that if no transition is enabled in a given moment, we stay in the current state. The following set of rules characterize the required temporal behaviour in RULER:

$$
\begin{array}{ll}
S_0 : & \\
S_1 : & \\
 & \\
r_1 : & S_0, a \multimap S_1 \\
r_2 : & S_0, \neg a \multimap S_0 \\
r_3 : & S_1, b \multimap S_0 \\
r_4 : & S_1, c \multimap r_{fail} \\
r_5 : & S_1, \neg b, \neg c \multimap S_1 \\
 & \\
g : & \multimap g, r_1, r_2, r_3, r_4, r_5 \\
r_{fail} : & \multimap r_{fail}
\end{array}
$$

We here assume that the monitoring algorithm starts with the following initial set of rule activation sets: $\{\{S_0, g, r_1, r_2, r_3, r_4, r_5\}\}$, and that the forbidden rule names are $\{S_1, r_{fail}\}$. That is, a trace fails to match the rule set if either S_1 (in case b never occurred after an a) or r_{fail} (in case a c occurred between an a and a b) occurs in the final state. The rules S_0 and S_1 are *memory* rules, tracking what state the machine is in at any point in time. Rules r_1 and r_2 represent the transitions leaving state $S0$, whereas rules r_3, r_4 and r_5 represent the transitions leaving state $S1$. Rules r_2 and r_5 handle the situation where no exiting transition is enabled, and we stay in the current state. This rule set will accept the trace $abab\epsilon$, but will reject $aba\epsilon$ as well as $abacb\epsilon$.

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.

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

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 any constraints imposed by the rule (on the acceptance of traces) will have no effect.

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 *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 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 consistent$\}$.

Definition 3. 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.

For example, the previous informal presentation of the rule

$$r_3 : \quad a, r_2 \multimap\!\!\!\!\rightarrow\ b \mid \neg b, r_4$$

is represented as the labelled structure

$$r_3 : \langle \{a, r_2\}, \{\{b\}, \{\neg b, r_4\}\}\rangle$$

where r_3 is the rule name, the condition body is the set $\{a, r_2\}$ and the body part is the set of sets $\{\{b\}, \{\neg b, r_4\}\}$ representing the two possible choices $\{b\}$ or $\{\neg b, r_4\}$.

Definition 4. 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, $I \subseteq 2^{O^{\pm} \cup R}$ is a set of consistent subsets 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).

The I component of a rule system defines an initial frontier of possible starting states, i.e. set of required observation literals and rule activations. An initial frontier of $\{\{a, r_1\}, \{\neg b, r_3\}\}$ thus has two possible starting states, one where observation a must be initially true and the rule name r_1 initially active, and one where observation b must not hold initially but rule r_2 is initially active.

Definition 5. 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.

Of interest now is 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.

Definition 6. Let $RS = \langle R, O, P, I, F\rangle$ be a rule system. A consistent set of literals L from RS holds in a configuration γ for RS, 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 $(L - R^{\pm}) \subseteq \Theta(\gamma)$.

Thus, as an example, assume a configuration $\gamma = \langle\{r_0, r_1\}, \{a, \neg b\}\rangle$ and consistent sets of literals $L_1 = \{r_0, \neg r_2, a\}$ and $L_2 = \{r_1, a, \neg c\}$. Assuming that the rule alphabet is $\{r_0, r_1, r_2\}$ and the observation alphabet is $\{a, b, c\}$, then we have that $\gamma \models L_1$ and $\gamma \not\models L_2$. L_2 doesn't hold in the configuration γ because the observation component is quiet on, i.e. doesn't mention, the observation c. We can neither conclude it truth or its falsity.

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.

Definition 7. 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 $A(\gamma') \cup \Theta(\gamma') = \theta$.

As an example, consider a named rule $r : \rho$ where the rule body ρ is the pair $\langle\{a\}, \{\{b\}, \{c, r\}\}\rangle$ where a, b and c are observation names. Below, we list some possible single steps using this rule.

$$\begin{array}{l}\langle\{r\}, \{a, b\}\rangle \xrightarrow{r:\rho} \langle\{r\}, \{c\}\rangle \\ \langle\{r\}, \{a, c\}\rangle \xrightarrow{r:\rho} \langle\{r\}, \{c\}\rangle \\ \langle\{r\}, \{a, c\}\rangle \xrightarrow{r:\rho} \langle\{\}, \{b\}\rangle\end{array}$$

Definition 8. For a set of rule names R, we say Γ' is an R*-indexed set of outcome configurations* if for each $r \in R$, $\gamma \xrightarrow{r:\rho} \Gamma'_r$.

Consider two named rules $r_a : \langle\{a\}, \{\{c\}, \{a, r_a\}\}\rangle$ and $r_b : \langle\{b\}, \{\{a\}, \{b, r_b\}\}\rangle$. The following are possible $\{r_a, r_b\}$-indexed sets of outcome configurations from the configuration $\langle\{r_a, r_b\}, \{a, b\}\rangle$.

$$\begin{array}{l}\{\langle\{r_a\}, \{a\}\rangle_{r_a}, \langle\{r_b\}, \{b\}\rangle_{r_b}\} \\ \{\langle\{\}, \{c\}\rangle_{r_a}, \langle\{r_b\}, \{b\}\rangle_{r_b}\} \\ \{\langle\{r_a\}, \{a\}\rangle_{r_a}, \langle\{\}, \{a\}\rangle_{r_b}\} \\ \{\langle\{\}, \{c\}\rangle_{r_a}, \langle\{\}, \{a\}\rangle_{r_b}\}\end{array}$$

Definition 9. A single step relation between configurations, $\gamma \longrightarrow \gamma'$, holds if and only if γ' is a consistent union of an $(A(\gamma) \cap R)$-indexed set of outcome configurations from γ. Note, we assume that an empty union set is treated as being an inconsistent union.

Thus, continuing with the above example, we have

$$\begin{array}{l}\langle\{r_a, r_b\}, \{a, b\}\rangle \longrightarrow \langle\{r_a, r_b\}, \{a, b\}\rangle \\ \langle\{r_a, r_b\}, \{a, b\}\rangle \longrightarrow \langle\{r_b\}, \{b, c\}\rangle \\ \langle\{r_a, r_b\}, \{a, b\}\rangle \longrightarrow \langle\{r_a\}, \{a\}\rangle \\ \langle\{r_a, r_b\}, \{a, b\}\rangle \longrightarrow \langle\{\}, \{a, c\}\rangle\end{array}$$

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.

Definition 10. A set of observation literals X is said to *match* an obligatory set of literals Y if and only if $X \cup Y$ is consistent and $Y \subseteq X$.

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

Definition 11 (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 $\gamma_1\gamma_2 \ldots \gamma_n\gamma_n$ such that (i) $\exists s \in I \cdot s \subseteq (A(\gamma_1) \cup \Theta(\gamma_1) \cup o_1)$, (ii) for all $i \in 1..n-1$ 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) $A(\gamma_n) \cap F = \{\}$. Hence, 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, $\tau \notin \mathcal{L}(RS)$.

Consider a rule system

$$\langle\{r\}, \{a, b\}, \{r : \langle\{\}, \{\{a, r\}, \{b\}\}\rangle\}, \{\{r\}\}, \{r\}\rangle.$$

As there is just one rule the single step relation $\longrightarrow$ is by definition the relation $\xrightarrow{r:\rho}$ where $\rho = \langle\{\}, \{\{a, r\}, \{b\}\}\rangle$, which, in turn, is

$$\{\langle\langle\{r\}, X\rangle, \langle\{r\}, \{a\}\rangle\rangle, \langle\langle\{r\}, X\rangle, \langle\{\}, \{b\}\rangle\rangle \mid X \subseteq \{a, b\}\}$$

The three state observation trace $\{\}\{a\}\{b\}$ has an accepting run, the sequence of configurations

$$\langle\{r\}, \{\}\rangle\langle\{r\}, \{a\}\rangle\langle\{\}, \{b\}\rangle.$$

Clearly condition (i) is satisfied. Similarly condition (ii) is satisfied, the input extended configurations are serially related by the single step relation. Finally, the final configuration doesn't contain a forbidden rule, i.e. $\{\}$ doesn't contain r.

Remark 12. In Section 2 we outlined the basic steps of a monitoring algorithm for propositional RuleR systems. It should be clear that the steps of the informally presented algorithm closely reflect the semantic constructions we have given above. For example, the "code" at lines 4–5 of the algorithm checks for each input state (set of observations) that there's a suitable match with the current obligations of the rule system (embodied in the frontier) — the match condition of part (ii) in the language acceptance definition. Then lines 6–8 of the "code" compute the successor frontier in accordance with the definition of the single step relation $\longrightarrow$ between configurations. Indeed, if a sufficiently detailed description of the algorithm were presented, it would not be difficult to establish that the algorithm accepts an observation trace τ for a rule system RS if and only if $\tau \in \mathcal{L}(RS)$.

4 Propositional linear temporal logic as a rule system

In this section, we show how propositional linear-time temporal logic formulas for monitoring over finite traces can be encoded in RuleR. There are two steps in our translation. The first step is to translate an arbitrary propositional linear-time temporal formula 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 [4], together with some initial assignment of proposition values. The proof that this is possible is based on the "Separation Result" of Gabbay (originally 1981 but elaborated in [11]) that shows that any mixed past, present and future linear-time temporal formula can be translated into a Boolean combinations of formulas, each of which depends purely on the past, present or future. We take the translation into the universal implicative form as given. The second part of the translation, and the part we expand on here, is 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. We use the following standard semantics over finite traces of states. Let $\tau = s_1, ..., s_i, ..., s_n$ where each state s_i denotes the subset of propositions that hold true at time i. We write $\tau, i \models \phi$ to represent the truth of the temporal formula ϕ at time i in the state trace τ. An inductive definition of $\models$ follows. In the following, we assume, without

$$
\begin{array}{ll}
\tau, i \models \textbf{true} & \\
\tau, i \not\models \textbf{false} & \\
\tau, i \models p & \text{iff } p \in \tau(i) \\
\tau, i \models \neg\phi & \text{iff } \tau, i \not\models \phi \\
\tau, i \models \phi \wedge \psi & \text{iff } \tau, i \models \phi \text{ and } \tau, i \models \psi \\
\tau, i \models \phi \vee \psi & \text{iff } \tau, i \models \phi \text{ or } \tau, i \models \psi \\
\tau, i \models \phi\,\mathcal{U}^+\psi & \text{iff there exists } k > i \text{ st } \tau, k \models \psi \text{ and} \\
& \qquad\qquad \text{for all } j \text{ where } k > j > i \text{ we have } \tau, j \models \phi \\
\tau, i \models \phi\,\mathcal{W}^+\psi & \text{iff there exists } k > i \text{ st } \tau, k \models \psi \text{ and} \\
& \qquad\qquad \text{for all } j \text{ where } k > j > i \text{ we have } \tau, j \models \phi \\
& \qquad \text{or for all } j \text{ where } |\tau| \leq j > i \text{ we have } \tau, j \models \phi
\end{array}
$$

Fig. 2. Future linear-time temporal logic semantics

loss of generality, that temporal formulas are transformed in negation normal form (NNF[5]), i.e. negation operators pushed inwards to propositional literals and cancellations applied. Let $\mathit{WFF}^+(\mathit{Obs})$ denote the set of well-formed strict future time formulas over proposition alphabet *Obs* in NNF and $\mathit{WFF}(\mathit{Obs})$ denote the set of well-formed future time formulas over proposition alphabet *Obs* in NNF (which may include the present, i.e. propositions under no future

[4] Fisher's SNF representation for temporal logic [9] 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.

time operator). For convenience, we omit the observation alphabet Obs when either it is clear from the context or unimportant.

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. The propositional constant **true** gives rise to the rule system $\langle \{r_{.g}\}, \{\}, \{r_{.g} : \multimap \{\{r_{.g}\}\}\}, \{\{r_{.g}\}\}, \{\} \rangle$ where the initial frontier of states contains just a singleton set with the rule $r_{.g}$ thus placing no constraints on the observation state. The rule $r_{.g}$ is used to keep the rule system active as any length of observation trace should be accepted by the system. On the other hand, **false** translates to a system with an empty set of initial states, i.e. denoting vacuity — there are no observation traces that satisfy false. For a proposition p, we have $\overrightarrow{T}(p) = \langle \{r_{.g}\}, \{p\}, \{r_{.g} : \multimap \{\{r_{.g}\}\}\}, \{\{p, r_{.g}\}\}, \{\} \rangle$, indicating a rule system with an observation atom p with an initial frontier containing the set $\{p, r_{.g}\}$. The rule $r_{.g}$ is included as any observation trace that has p present initially is acceptable, therefore the rule system must keep active until the end of the observation trace. Negated atoms translate in a similar way, apart from the initial frontier containing the set comprising the negated atom and the rule $r_{.g}$. The logical conjunction (disjunction) of formulas ϕ and ψ translate to a product (union) operations that can be defined for rule systems. For the rule system products, we require that the initial frontier is formed as a consistent product, whereas for the union operation, we simply use the union of the initial frontiers. 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 $\phi \mathcal{U}^+ \psi \Leftrightarrow \bigcirc(\psi \vee (\phi \wedge (\phi \mathcal{U}^+ \psi)))$.

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

$$\langle \{r_{.g}, r_{a\mathcal{U}^+b}\}, \{a, b\}, \left\{ \begin{array}{l} r_{.g} : \multimap r_{.g} \\ r_{a\mathcal{U}^+b} : \multimap b, r_{.g} \mid a, r_{.g}, r_{a\mathcal{U}^+b} \end{array} \right\}, \{\{r_{a\mathcal{U}^+b}\}\}, \{r_{a\mathcal{U}^+b}\} \rangle$$

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

$$\langle \{r_{.g}, r_{c\mathcal{W}^+d}\}, \{a, c, d\}, \left\{ \begin{array}{l} r_{.g} : \multimap r_{.g} \\ r_{c\mathcal{W}^+d} : \multimap d, r_{.g} \mid c, r_{.g}, r_{c\mathcal{W}^+d} \end{array} \right\}, \{\{a, r_{.g}, r_{c\mathcal{W}^+d}\}\}, \{\} \rangle$$

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

$$
\begin{array}{ll}
\overrightarrow{T}(\mathbf{true}) & = \langle \{r_{.g}\}, \{\}, \{r_{.g} : \multimap\!\!\!\rightarrow \{\{r_{.g}\}\}\}, \{\{r_{.g}\}\}, \{\} \rangle \\
\overrightarrow{T}(\mathbf{false}) & = \langle \{\}, \{\}, \{\}, \{\}, \{\} \rangle \\
\overrightarrow{T}(p) & = \langle \{r_{.g}\}, \{p\}, \{r_{.g} : \multimap\!\!\!\rightarrow \{\{r_{.g}\}\}\}, \{\{p, r_{.g}\}\}, \{\} \rangle \\
\overrightarrow{T}(\neg q) & = \langle \{r_{.g}\}, \{q\}, \{r_{.g} : \multimap\!\!\!\rightarrow \{\{r_{.g}\}\}\}, \{\{\neg q, r_{.g}\}\}, \{\} \rangle \\
\overrightarrow{T}(\phi \wedge \psi) & = \text{LET } \langle R_\phi, O_\phi, P_\phi, I_\phi, F_\phi \rangle = \overrightarrow{T}(\phi) \text{ AND } \langle R_\psi, O_\psi, P_\psi, I_\psi, F_\psi \rangle = \overrightarrow{T}(\psi) \\
& \quad \text{IN } \langle R_\phi \cup R_\psi,\ O_\phi \cup O_\psi,\ P_\phi \cup P_\psi,\ I_\phi \times I_\psi,\ F_\phi \cup F_\psi \rangle \\
\overrightarrow{T}(\phi \vee \psi) & = \text{LET } \langle R_\phi, O_\phi, P_\phi, I_\phi, F_\phi \rangle = \overrightarrow{T}(\phi) \text{ AND } \langle R_\psi, O_\psi, P_\psi, I_\psi, F_\psi \rangle = \overrightarrow{T}(\psi) \\
& \quad \text{IN } \langle R_\phi \cup R_\psi,\ O_\phi \cup O_\psi,\ P_\phi \cup P_\psi,\ I_\phi \cup I_\psi,\ F_\phi \cup F_\psi \rangle \\
\overrightarrow{T}(\phi\,\mathcal{U}^+\psi) & = \text{LET } \langle R_\phi, O_\phi, P_\phi, I_\phi, F_\phi \rangle = \overrightarrow{T}(\phi) \text{ AND} \\
& \qquad \langle R_\psi, O_\psi, P_\psi, I_\psi, F_\psi \rangle = \overrightarrow{T}(\psi) \\
& \quad \text{IN } \langle R_\phi \cup R_\psi \cup \{r_{\phi\,\mathcal{U}^+\psi}\},\ O_\phi \cup O_\psi, \\
& \qquad P_\phi \cup P_\psi \cup \{r_{\phi\,\mathcal{U}^+\psi} : \multimap\!\!\!\rightarrow I_\psi \cup (I_\phi \times \{\{r_{\phi\,\mathcal{U}^+\psi}\}\}), \\
& \qquad \{\{r_{\phi\,\mathcal{U}^+\psi}\}\},\ F_\phi \cup F_\psi \cup \{r_{\phi\,\mathcal{U}^+\psi}\} \rangle \\
\overrightarrow{T}(\phi\,\mathcal{W}^+\psi) & = \text{LET } \langle R_\phi, O_\phi, P_\phi, I_\phi, F_\phi \rangle = \overrightarrow{T}(\phi) \text{ AND} \\
& \qquad \langle R_\psi, O_\psi, P_\psi, I_\psi, F_\psi \rangle = \overrightarrow{T}(\psi) \\
& \quad \text{IN } \langle R_\phi \cup R_\psi \cup \{r_{\phi\,\mathcal{W}^+\psi}\},\ O_\phi \cup O_\psi, \\
& \qquad P_\phi \cup P_\psi \cup \{r_{\phi\,\mathcal{W}^+\psi} : \multimap\!\!\!\rightarrow I_\psi \cup (I_\phi \times \{\{r_{\phi\,\mathcal{W}^+\psi}\}\}), \\
& \qquad \{\{r_{\phi\,\mathcal{W}^+\psi}\}\},\ F_\phi \cup F_\psi \rangle
\end{array}
$$

Fig. 3. Translation of future linear-time temporal formulas

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

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

where

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

The correctness of the translation is given by the following result.

Theorem 14. *For any NNF formula $\phi \in$ WFF(Obs) and finite observation trace τ over observation propositions Obs, the suffix of τ from index i, denoted by $\tau^{(i)}$, is in the language $\mathcal{L}(\overrightarrow{T}(\phi))$ if and only $\tau, i \models \phi$.*

Proof. We proceed by induction over the structure of the formula ϕ. For the base cases, we have:-

(i) ϕ is **true**. The generated rule system has just one rule of the form

$$
r_{.g} : \multimap\!\!\!\rightarrow r_{.g}
$$

with an initial frontier I of $\{\{r_{.g}\}\}$ and an empty forbidden rule set F. This rule has an empty condition part, and hence always generates obligations when the rule is active. Furthermore, the rule's obligation set places no constraint on observations, it only ensures that the rule is active again for subsequent application. As there are also no constraints on rule names active in any final configuration, any finite observation trace will be accepted by the rule system. This corresponds to all possible linear observation sequences that are models for the temporal formula **true**. The desired result follows since *true* is independent of the past.

(ii) ϕ is **false**. The generated rule system has no rules and an empty initial frontier. There are thus no initial observations that can be matched with the initial frontier, hence the language of the rule system is empty, corresponding to the empty set of models for the temporal formula **false**. The desired result follows.

(iii) ϕ is a proposition p. The generated rule system differs from that generated for the formula **true** in that the initial frontier I also contains the observation proposition p, hence requiring it to be present in the initial state of all observation traces accepted by the rule system. The single rule of the system places no further constraints on the observations. Hence the language accepted by the rule system corresponds to the set of traces τ over Obs such that $\tau, 0 \models p$, and thus, as p is independent of the past, $\tau^{(i)} \in \mathcal{L}(\overrightarrow{T}(p))$ if and only if $\tau, i \models p$.

(iv) ϕ is $\neg q$ for proposition q. The argument here is similar to (iii) apart from the requirement that q can not be present initially.

(v) ϕ is $X \wedge Y$ for $X, Y \in \mathit{WFF}(Obs)$. Wlog. we assume that the rule name alphabets of $\overrightarrow{T}(X)$ and $\overrightarrow{T}(Y)$ are disjoint. Consider a trace τ such that $\tau, i \models X \wedge Y$. Thus by the temporal logic semantics we have that $\tau, i \models X$ and $\tau, i \models Y$. Using the inductive hypotheses, there are thus accepting runs σ_X and σ_Y over $\tau^{(i)}$ of rule systems $\overrightarrow{T}(X)$ and $\overrightarrow{T}(Y)$, respectively. Let σ_{XY} denote the pointwise union of the two accepting runs (respecting the activation and observation set structure). We argue that σ_{XY} will be an accepting run of the rule system $\overrightarrow{T}(X \wedge Y)$: (i) the initial set $I(\overrightarrow{T}(X \wedge Y))$ comprises all the union pairings of the elements of $I(\overrightarrow{T}(X))$ and $I(\overrightarrow{T}(Y))$, hence, by construction, at least one of the unions will "hold" initially on σ_{XY}; and (ii) as the rule name sets of each translation are disjoint, the run can be viewed as the parallel run of rule systems for X and for Y.
For the other direction, we consider a trace τ such that $\tau, i \not\models X \wedge Y$. Assume wlog. that $\tau, i \not\models X$. Thus, by the induction hypothesis, there is no accepting run σ_X of $\overrightarrow{T}(X)$. By the construction of $\overrightarrow{T}(X \wedge Y)$ there can not be an accepting run, for if there were we could project out the run on the rule names of X and thereby construct an accepting run of $\overrightarrow{T}(X)$.

(vi) ϕ is $X \vee Y$ for $X, Y \in \mathit{WFF}(Obs)$. As above, wlog. we assume that the rule name alphabets of $\overrightarrow{T}(X)$ and $\overrightarrow{T}(Y)$ are disjoint. Consider a trace τ such that $\tau, i \models X \vee Y$. From the temporal logic semantics we have $\tau, i \models X$ or $\tau, i \models Y$. By the induction hypothesis we thus have $\tau^{(i)} \in \mathcal{L}(\overrightarrow{T}(X))$ or $\tau^{(i)} \in \mathcal{L}(\overrightarrow{T}(Y))$. But if $\tau^{(i)} \in \mathcal{L}(\overrightarrow{T}(X))$ then there is an accepting run σ

and, by definition of $\overrightarrow{T}$, the rule system $\overrightarrow{T}(X \vee Y)$ can use the same run σ to accept $\tau^{(i)}$. Similarly for Y. Hence $\tau^{(i)} \in \mathcal{L}(\overrightarrow{T}(X \vee Y))$. A similar argument applies for the other direction and hence the desired result.

(vii) ϕ is $X\,\mathcal{U}^{+}Y$ for $X, Y \in \mathit{WFF}(\mathit{Obs})$. We establish $\tau^{(i)} \in \mathcal{L}(\overrightarrow{T}(X\,\mathcal{U}^{+}Y))$ for any $i \in 1..|\tau|$ by a downward induction on i under the overall hypotheses $\tau^{(i)} \in \mathcal{L}(\overrightarrow{T}(X))$ and $\tau^{(i)} \in \mathcal{L}(\overrightarrow{T}(Y))$ for any $i \in 1..|\tau|$.

Base case. This is when $i = |\tau|$. By the temporal logic semantics, $\tau, |\tau| \not\models X\,\mathcal{U}^{+}Y$ — there is no future. There is, however, no accepting run of $\overrightarrow{T}(X\,\mathcal{U}^{+}Y)$ of length 1 because $\{r_{X\,\mathcal{U}^{+}Y}\}$ must be in I and also a subset of F. Hence $\tau^{(|\tau|)} \in \mathcal{L}(\overrightarrow{T}(X\,\mathcal{U}^{+}Y))$ iff $\tau, |\tau| \models X\,\mathcal{U}^{+}Y$.

Inductive step. We assume $\tau^{(j)} \in \mathcal{L}(\overrightarrow{T}(X\,\mathcal{U}^{+}Y))$ iff $\tau, j \models X\,\mathcal{U}^{+}Y$ for j, $1 < j \leq |\tau]$ and show $\tau^{(j-1)} \in \mathcal{L}(\overrightarrow{T}(X\,\mathcal{U}^{+}Y))$ iff $\tau, j\text{–}1 \models X\,\mathcal{U}^{+}Y$. By the overall inductive hypotheses and the sub-induction hypothesis, we claim that $\tau^{(j)} \in \mathcal{L}(\overrightarrow{T}(Y \vee (X \wedge X\,\mathcal{U}^{+}Y)))$ iff $\tau, j \models Y \vee (X \wedge X\,\mathcal{U}^{+}Y)$. First observe that there's an accepting run of $\overrightarrow{T}(Y \vee (X \wedge X\,\mathcal{U}^{+}Y))$ on $\tau^{(j)}$ if there's an accepting run of $\overrightarrow{T}(Y)$ on $\tau^{(j)}$ or there's an accepting run on $\tau^{(j)}$ for both $\overrightarrow{T}(X)$ and $\overrightarrow{T}(X\,\mathcal{U}^{+}Y)$; this follows from the fact that the set of sets of initial conditions $I(\overrightarrow{T}(Y \vee (X \wedge X\,\mathcal{U}^{+}Y)))$ is the same as $I(\overrightarrow{T}(Y)) \cup (I(\overrightarrow{T}(X)) \times I(\overrightarrow{T}(X\,\mathcal{U}^{+}Y)))$. By the inductive hypotheses, we thus have $\tau, j \models Y$ or both $\tau, j \models X$ and $\tau, j \models X\,\mathcal{U}^{+}Y$, and therefore $\tau, j \models Y \vee (X \wedge X\,\mathcal{U}^{+}Y)$, which means, by the temporal semantics of $\mathcal{U}^{+}$ that $\tau, j-1 \models X\,\mathcal{U}^{+}Y$. Furthermore, if we have an accepting run of $\overrightarrow{T}(Y \vee (X \wedge X\,\mathcal{U}^{+}Y))$ on $\tau^{(j)}$ then the rule system $\overrightarrow{T}(X\,\mathcal{U}^{+}Y)$ will have an accepting run on $\tau^{(j-1)}$; the rule $r_{X\,\mathcal{U}^{+}Y}$ obligates $I(\overrightarrow{T}(Y)) \cup (I(\overrightarrow{T}(X)) \times \{\{r_{X\,\mathcal{U}^{+}Y}\}\})$ on the next step, and as $I(\overrightarrow{T}(X\,\mathcal{U}^{+}Y)) = \{\{r_{X\,\mathcal{U}^{+}Y}\}\}$, we will have an accepting run on $\tau^{(j-1)}$.

Second, there's no accepting run of $\overrightarrow{T}(Y \vee (X \wedge X\,\mathcal{U}^{+}Y))$ on $\tau^{(j)}$ if there's no accepting run on $\tau^{(j)}$ for $\overrightarrow{T}(Y)$ and for at least one of $\overrightarrow{T}(X)$ and $\overrightarrow{T}(X\,\mathcal{U}^{+}Y)$. Hence, by the inductive hypotheses we have $\tau, j \not\models Y \vee (X \wedge X\,\mathcal{U}^{+}Y)$ and therefore, by the semantics of $\mathcal{U}^{+}$, we have $\tau, j-1 \not\models X\,\mathcal{U}^{+}Y$. But if there's no accepting run of $\overrightarrow{T}(Y \vee (X \wedge X\,\mathcal{U}^{+}Y))$ on $\tau^{(j)}$ then there's no accepting run of $\overrightarrow{T}(X\,\mathcal{U}^{+}Y)$ on $\tau^{(j-1)}$, for if this were not the case we would have, in effect, an accepting run of $\overrightarrow{T}(Y \vee (X \wedge X\,\mathcal{U}^{+}Y))$ on $\tau^{(j)}$, which is not the case.

Hence the desired result.

(viii) ϕ is $X\,\mathcal{W}^{+}Y$ for $X, Y \in \mathit{WFF}(\mathit{Obs})$. This case is argued in a similar way to the case for $\mathcal{U}^{+}$ apart from the fact that the base case has the formula $X\,\mathcal{W}^{+}Y$ true on the final observation state with the rule $r_{X\,\mathcal{W}^{+}Y}$ not forbidden from the final sets of activated rules.

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). We spell out the semantics of the $\mathcal{S}^-$ and $\mathcal{Z}^-$ temporal operators.

$$\begin{aligned}
\tau, i \models \phi\, \mathcal{S}^- \psi \text{ iff } & \text{there exists } k \text{ st } 1 \leq k < i \text{ and } \tau, k \models \psi \text{ and} \\
& \quad\text{for all } j \text{ where } k < j < i \text{ we have } \tau, j \models \phi \\
\tau, i \models \phi\, \mathcal{Z}^- \psi \text{ iff } & \text{there exists } 1 \leq k < i \text{ st } \tau, k \models \psi \text{ and} \\
& \quad\text{for all } j \text{ where } k < j < i \text{ we have } \tau, j \models \phi \\
& \text{or for all } j \text{ where } 1 \leq j < i \text{ we have } \tau, j \models \phi
\end{aligned}$$

Without loss of generality, we assume that past time temporal formulas are in negation normal form. Let us first informally consider the translation of pure past time temporal queries. The temporal equivalence $\phi\, \mathcal{S}^- \psi \Leftrightarrow \odot(\psi \vee (\phi \wedge (\phi\, \mathcal{S}^- \psi)))$ 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). Let us consider a simple example. We will use the presence of the rule name $r_{p\,\mathcal{S}^- q}$ in the rule activation state to denote that the pure past temporal formula $p\,\mathcal{S}^- q$ holds at that moment in the evaluation. We then use a rule, named $r_{p:p\,\mathcal{S}^- q?}$, to calculate whether $r_{p\,\mathcal{S}^- q}$ should be made active because p held in the previous moment (similarly for the other possible way for $p\,\mathcal{S}^- q$ 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 $r_{g.p\,\mathcal{S}^- q?}$, to act as a generator (hence the "g" in its name) for the set of rules that determine the truth of $p\,\mathcal{S}^- q$ based on the previous values of its subformulas.

$$\begin{aligned}
& r_{q.p\,\mathcal{S}^- q?} : q \multimap r_{p\,\mathcal{S}^- q} \\
& r_{p.p\,\mathcal{S}^- q?} : p, r_{p\,\mathcal{S}^- q} \multimap r_{p\,\mathcal{S}^- q} \\
& r_{g.p\,\mathcal{S}^- q?} : \ \multimap r_{g.p\,\mathcal{S}^- q?}, r_{p.p\,\mathcal{S}^- q?}, r_{q.p\,\mathcal{S}^- q?}
\end{aligned}$$

Naturally, our translation must be generalised to take into account that the subformulas $\psi\, \mathcal{S}^- \phi$ 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 extension of a rule system $\langle R, O, P, I, S, Q \rangle$ where the fields R, O, P and I are as defined in a rule system, S denotes a set of initial starting values[6]for the rules representing since (zince) formulas and Q is a set of set of rule names denoting the past time queries calculated by the rule system. For example, a formula $\phi\, \mathcal{S}^- \psi$ must be false initially and so we include the rule name $\neg r_{\phi\, \mathcal{S}^- \psi}$ in the set S, on the other hand,

[6] We distinguish this set S from I as I may also contain initial values of observation propositions. S, in a sense, is the past-time counterpart to the forbidden set F in a rule system proper.

a formula $\phi\,\mathcal{Z}^{-}\psi$ is evaluated as true at the beginning of time and therefore we include the associated rule name $r_{\phi\,\mathcal{Z}^{-}\psi}$ in the set S.

Figure 4 gives the translation for the present and pure past time temporal formulas. Note that the key difference between the translation of the $\mathcal{S}^{-}$ and $\mathcal{Z}^{-}$ temporal connectives is in the definition of the S component. The correctness of

$$
\begin{aligned}
&\overleftarrow{T}(\mathbf{true}) &&= \langle \{\}, \{\}, \{\}, \{\{\}\}, \{\}, \{\{\}\} \rangle\\
&\overleftarrow{T}(\mathbf{false}) &&= \langle \{\}, \{\}, \{\}, \{\}, \{\}, \{\} \rangle\\
&\overleftarrow{T}(p) &&= \langle \{\}, \{p\}, \{\}, \{\{\}\}, \{\}, \{\{p\}\} \rangle\\
&\overleftarrow{T}(\neg q) &&= \langle \{\}, \{q\}, \{\}, \{\{\}\}, \{\}, \{\{\neg q\}\} \rangle\\
&\overleftarrow{T}(\phi \wedge \psi) &&= \text{LET } \langle R_\phi, O_\phi, P_\phi, I_\phi, S_\phi, Q_\phi \rangle = \overleftarrow{T}(\phi) \text{ AND}\\
& && \qquad \langle R_\psi, O_\psi, P_\psi, I_\psi, S_\psi, Q_\phi \rangle = \overleftarrow{T}(\psi)\\
& && \quad \text{IN } \langle R_\phi \cup R_\psi,\ O_\phi \cup O_\psi,\ P_\phi \cup P_\psi,\ I_\phi \times I_\psi,\ S_\phi \cup S_\psi,\ Q_\phi \times Q_\psi \rangle\\
&\overleftarrow{T}(\phi \vee \psi) &&= \text{LET } \langle R_\phi, O_\phi, P_\phi, I_\phi, S_\phi, Q_\phi \rangle = \overleftarrow{T}(\phi) \text{ AND}\\
& && \qquad \langle R_\psi, O_\psi, P_\psi, I_\psi, S_\psi, Q_\psi \rangle = \overleftarrow{T}(\psi)\\
& && \quad \text{IN } \langle R_\phi \cup R_\psi,\ O_\phi \cup O_\psi,\ P_\phi \cup P_\psi,\ I_\phi \cup I_\psi,\ S_\phi \cup S_\psi,\ Q_\phi \cup Q_\psi \rangle
\end{aligned}
$$

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

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

Fig. 4. Translation of present and past time temporal query formulas

this translation is given by the following theorem.

Theorem 15. *For any NNF formula $\phi \in WFF^{-0}$ and observation trace τ and index $i \in 1..|\tau|$, $\tau, i \models \phi$ if and only there's an accepting run γ on observation trace $\tau[1..i]$ via the rules of $\overleftarrow{T}(\phi)$ and for some $s \in Q(\overleftarrow{T}(\phi))$ we have $s \subseteq \mathcal{A}(\gamma_i) \cup \Theta(\gamma_i)$.*

Proof. The proof proceeds by induction over the structure of the formula $\phi \in WFF^{-0}$. The interesting subcases are for the temporal operators, each requiring a further inductive proof over the index i. We consider just the case for $\phi = X\,\mathcal{S}^{-}Y$. The argument for $X\,\mathcal{Z}^{-}Y$ is similar.

Base case. This is when $i = 1$. Note that $\tau, 1 \not\models X\,\mathcal{S}^{-}Y$ as there is no past. $Q(\overleftarrow{T}(X\,\mathcal{S}^{-}Y)) = \{\{r_{X\,\mathcal{S}^{-}Y}\}\}$. However, the rule $r_{X\,\mathcal{S}^{-}Y}$ can't exist in any resultant state of the frontier at index 1 since it is required by the S initial frontier set that $\neg r_{X\,\mathcal{S}^{-}Y}$ is present. Similarly for the other direction. Thus there is no accepting run of $\overleftarrow{T}(X\,\mathcal{S}^{-}Y)$ on $\tau[1..1]$.

Inductive Step. Here we assume the result at index $i < |\tau|$ and show it holds for $i+1$. We make use, of course, of the result holding for the sub-formulas X and Y at any index i (the inductive hypothesis of the outer-level structural induction proof. We also assume, wlog., that the rule names of the systems for the translations of X and Y are disjoint. Three forms of rules are included in the translation over and above the rules obtained for X and Y. The first rule, the generator, ensures that the second and third forms are always active. Let us consider the if direction of the theorem. We have $\tau, i \models X\,\mathcal{S}^{-}Y$. By the semantics of the temporal operator $\mathcal{S}^{-}$, $\tau, i+1 \models X\,\mathcal{S}^{-}Y$ if and only if (i) $\tau, i \models Y$ or (ii) $\tau, i \models X$ and $\tau, i \models X\,\mathcal{S}^{-}Y$. For subcase (i), the structural inductive hypothesis gives that there's an accepting run of $\overleftarrow{T}(X)$ on $\tau[1..i]$ where for some $x \in Q(\overleftarrow{T}(X))$ we have $x \subseteq \mathcal{A}(\gamma_i) \cup \Theta(\gamma_i) \cup \tau_i$. Hence by the second rule form we will have that $r_{X\,\mathcal{S}^{-}Y}$ will be present in $\mathcal{A}(\gamma_{i+1}) \cup \Theta(\gamma_{i+1})$. Alternatively, for subcase (ii), the hypotheses will mean that for some $x \in Q(\overleftarrow{T}(X))$ we have both x and $r_{X\,\mathcal{S}^{-}Y}$ present in $x \subseteq \mathcal{A}(\gamma_i) \cup \Theta(\gamma_i) \cup \tau_i$. The third rule thus ensures that $r_{X\,\mathcal{S}^{-}Y}$ is present in $\mathcal{A}(\gamma_{i+1}) \cup \Theta(\gamma_{i+1})$. Thus we can conclude that there will be an accepting run on $\tau[1..i+1]$. The converse follows similar argumentation.

Separated temporal implicative forms. We now bring together the above two translations $\overrightarrow{T}$ and $\overleftarrow{T}$ to define a translation T for the MetateM-like rule form $\phi_{\text{past}} \Rightarrow \bigcirc\psi_{\text{future}}$ which is of universal nature, i.e. globally holds. A translation scheme is given in Figure 5. The translation of the present and/or past time formula ϕ will yield a set of sets of rule names that represent the query. Presence of one of the sets of rule instances at some stage during the evaluation denotes the truth of the ϕ at that point. The translation of the future time formula ψ will provide a set of sets of rule instances / observations (the initial frontier) such that a successful run starting with that initial frontier over an observation trace will mean the observation trace will satisfy the formula. Thus, the translation for the separated implicative rule needs to create a set of rule such that each element of $Q(\overleftarrow{T}(\phi))$ is combined as an antecedent with the initial frontier $I(\overrightarrow{T}(\psi))$ as consequent. Each such constructed RuleR rule then needs to be made global.

For $\phi \in \mathit{WFF}^{-0}$ and $\psi \in \mathit{WFF}$ and both in NNF

$$
\begin{array}{l}
T(\phi, \psi) = \\
\quad \text{LET } \langle R_{\text{past}}, O_{\text{past}}, P_{\text{past}}, I_{\text{past}}, S_{\text{past}}, Q_{\text{past}} \rangle = \overleftarrow{T}(\phi) \text{ AND} \\
\qquad \langle R_{\text{future}}, O_{\text{future}}, P_{\text{future}}, I_{\text{future}}, F_{\text{future}} \rangle = \overrightarrow{T}(\psi) \\
\quad \text{IN } \langle R_{\text{past}} \cup R_{\text{future}} \cup \{r_{g.\phi_{\text{past}} \Rightarrow \psi_{\text{future}}}\} \cup \{r_{x \Rightarrow \psi_{\text{future}}} \mid x \in Q_{\text{past}}\}, \\
\qquad O_{\text{past}} \cup O_{\text{future}}, \\
\qquad P_{\text{past}} \cup P_{\text{future}} \cup \\
\qquad\quad \{r_{g.\phi_{\text{past}} \Rightarrow \psi_{\text{future}}} : \ \multimap \ \{r_{g.\phi_{\text{past}} \Rightarrow \psi_{\text{future}}}\} \cup \{r_{x \Rightarrow \psi_{\text{future}}} \mid x \in Q_{\text{past}}\}\}, \\
\qquad\quad \{r_{x \Rightarrow \psi_{\text{future}}} : x \multimap I_{\text{future}} \mid x \in Q_{\text{past}}\}, \\
\qquad \{\{r_{g.\phi_{\text{past}} \Rightarrow \psi_{\text{future}}}, r_{x \Rightarrow \psi_{\text{future}}} \mid x \in Q_{\text{past}}\} \cup S_{\text{past}}\} \times (I_{\text{past}} \backslash R_{\text{past}}), \\
\qquad F_{\text{future}} \rangle
\end{array}
$$

Fig. 5. Translation of global temporally separated implicative rules

Example 16. 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 $p \vee \oplus p$, i.e. $p \vee \mathbf{true}\ \mathcal{U}^{+} p$, similarly for $\Diamond q$. Using the following abbreviations

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

the translation yields the rule system with rules

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

and 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 $\{r_4, r_5\}$.

Theorem 17. *For any (finite) observation trace τ, present or past time formula $\phi \in \mathit{WFF}^{-0}$ and future time formula $\psi \in \mathit{WFF}$ we have $\tau, 0 \models \Box(\phi \Rightarrow \bigcirc\psi)$ if and only if there's an accepting run of $T(\phi, \psi)$ for the observation trace τ.*

Proof. From the definition of the temporal operator $\Box$, we need to establish that for all $i \in 1..|\tau|$ $\tau, i \models \phi \Rightarrow \bigcirc\psi$ if and only if there's an accepting run of $T(\phi, \psi)$ for the observation trace τ.

if: We assume that for any i if $\tau, i \models \phi$ then $\tau, i+1 \models \psi$. We show that there's an accepting run of $T(\phi, \psi)$ on τ. Wlog. we assume that $\overleftarrow{T}(\phi)$ and $\overrightarrow{T}(\psi)$ have distinct rule names. Given $\tau, i \models \phi$, by Theorem 15 we have that there's an accepting run γ over $\tau[1..i]$ for which some set $x \in Q(\overleftarrow{T}(\phi))$ is a subset of one of the resultant states at index i. The translation $T(\phi, \psi)$ generates a rule $r_{x \Rightarrow \psi} : x \multimap I(\overrightarrow{T}(\psi))$ for each $x \in Q(\overleftarrow{T}(\phi))$ which is made global, i.e. active for every step. The rule system frontier at $i+1$ thus features $I(\overrightarrow{T}(\psi))$. But by Theorem 14, $\tau, i+1 \models \psi$ if and only there's an accepting run of $\overrightarrow{T}(\psi)$ on $\tau^{(i+1)}$. As $T(\phi, \psi)$ contains all the rules of $\overrightarrow{T}(\psi)$ and $I(\overrightarrow{T}(\psi))$ features in the frontier at step $i+1$, we thus have that there'll be an accepting run of $T(\phi, \psi)$ over τ.

only if: For this case, we show that under the assumption that for some i we have $\tau, i \models \phi$ and $\tau, i+1 \not\models \psi$ then there isn't an accepting run of $T(\phi, \psi)$ over τ. By Theorems 15 and 14 we have, respectively, there's an accepting run of $\overleftarrow{T}(\phi)$ on $\tau[1..i]$ but not an accepting run of $\overrightarrow{T}(\psi)$ on $\tau^{(i+1)}$. The rule system for $T(\phi, \psi)$ thus has a run over $\tau[1..i]$ for which some set $x \in Q(\overleftarrow{T}(\phi))$ is a subset of one of the resultant states at index i. The globally active rule $r_{x \Rightarrow \psi} : x \multimap I(\overrightarrow{T}(\psi))$ for each $x \in Q(\overleftarrow{T}(\phi))$ thus embeds $I(\overrightarrow{T}(\psi))$ across the frontier at $i+1$. Now given that there's no accepting run of $\overrightarrow{T}(\psi)$ on $\tau^{(i+1)}$ and the rules of $\overrightarrow{T}(\psi)$ are a part of $T(\phi, \psi)$, there'll be no accepting run over τ.

Finally, in order to obtain a translation for an arbitrary linear-time temporal logic formula, one needs to translate the given formula to the above separated implicative temporal rule forms together with any initial constraints. The translation is detailed but can be generated from the algorithm for separation given outlined in [11]. As was often argued with MetateM, however, it is often more natural to write temporal properties directly in this particular separated implicative form.

5 Parameterized RuleR

In this section, we consider versions of RuleR in which rule definitions can be parameterized by both rules and by data, observations may be represented by predicate instances, and boolean combinations of relations over arithmetic terms allowed in antecedents and consequents of the rule bodies.

5.1 RuleR with rule arguments

We first consider an extension of RuleR to include rule definitions parameterized by rules. The propositional RuleR system corresponds to regular languages, which are a subclass of propositional EAGLE. The evaluation strategy used on this first, seemingly small, extension increases the formal expressivity of RULER to 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 has been activated with the propositional rule r_0 substituted for ρ, i.e. $r(r_0)$ is active. The evaluation of the rule body associated with $r(r_0)$ 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 RULER, when parameterized by rules, can yield context-sensitive languages.

Example 18. Let us consider the classic context-free language $\{a^n b^n | n \geq 1\}$. We encode it in RULER as a rule system accepting the language $\{\{a, \neg b\}^n \{\neg a, b\}^n \mid n \geq 1\}$ where a and b are observation propositions. We use the rule schema

$$r_b(\rho) : \multimap b, \neg a, \rho \qquad r_{ab}(\rho) : \multimap b, \neg a, \rho \mid a, \neg b, r_{ab}(r_b(\rho))$$
$$r_{end} : \multimap r_{fail} \qquad r_{fail} : \multimap r_{fail}$$

together with its initial rule activation set as $\{\{a, r_{ab}(r_{end})\}\}$ and its final forbidden rule set as $\{r_b, r_{ab}, r_{fail}\}$ (meaning that no occurrence of rule r_b, r_{ab}, nor r_{fail}, may appear as an obligation in a final rule activation state). Below in Table 2 we spell out the evaluation over an input trace corresponding to three as followed by three bs (we assume only one of a or b may be true at any one time). Let us look at the situations in step 2 and step 3. The resultant state in

Step	Obs.	Rule Activations	Resultant States
0	$\{a, \neg b\}$	$\{\{a, r_{ab}(r_{end})\}\}$	$\{\{a, \neg b, r_{ab}(r_{end})\}\}$
1	$\{a, \neg b\}$	$\left\{\begin{array}{l}\{b, \neg a, r_{end}\},\\ \{a, \neg b, r_{ab}(r_b(r_{end}))\}\end{array}\right\}$	$\{\{a, \neg b, r_{ab}(r_b(r_{end}))\}\}$
2	$\{a, \neg b\}$	$\left\{\begin{array}{l}\{b, \neg a, r_b(r_{end})\},\\ \{a, \neg b, r_{ab}(r_b(r_b(r_{end})))\}\end{array}\right\}$	$\{\{a, \neg b, r_{ab}(r_b(r_b(r_{end})))\}\}$
3	$\{\neg a, b\}$	$\left\{\begin{array}{l}\{b, \neg a, r_b(r_b(r_{end}))\},\\ \{a, \neg b, r_{ab}(r_b(r_b(r_b(r_{end}))))\}\end{array}\right\}$	$\{\{\neg a, b, r_b(r_b(r_{end}))\}\}$
4	$\{\neg a, b\}$	$\{\{b, \neg a, r_b(r_{end})\}\}$	$\{\{\neg a, b, r_b(r_{end})\}\}$
5	$\{\neg a, b\}$	$\{\{b, \neg a, r_{end}\}\}$	$\{\{\neg a, b, r_{end}\}\}$

Table 2. Example evaluation on $aaabbb$

step 2 activates rule r_{ab} with argument $r_b(r_b(r_{end}))$ denoting that if a b occurs next in step 3 then two further bs will be required to match the as that have been seen so far. Expansion of the particular rule call gives two possible rule activation sets in step 3, the second of which is now inconsistent with the given observations $\{\neg a, b\}$. Hence the resultant state with just rule r_b active (with argument $r_b(r_{end})$. By step 5, the evaluation of the rule system over the trace is successful — the final (resultant) state has no forbidden rules.

Suppose, now, the input hadn't terminated and continued with two more observations states. The evaluation would proceed from step 5 as in Table 3 below. The "illegal" input (appearing in steps 6 and 7) is consumed, however, the

Step	Obs.	Rule Activations	Resultant States
5	$\{\neg a, b\}$	$\{\{b, \neg a, r_{end}\}\}$	$\{\{\neg a, b, r_{end}\}\}$
6	$\{a, \neg b\}$	$\{\{r_{fail}\}\}$	$\{\{a, \neg b, r_{fail}\}\}$
7	$\{\neg a, b\}$	$\{\{r_{fail}\}\}$	$\{\{\neg, b, r_{fail}\}\}$

Table 3. Evaluation trace on *aaabbbab*

rule r_{fail} is activated through r_{end} in the merged state of step 5 and maintained active until the input is completed. The evaluation then fails since the rule r_{fail} is one of the forbidden rules. Terminating earlier than expected results in either r_{ab} or r_b present in the final (merged) state, which are also forbidden rules. Alternatively, if the input has too few bs and then continues with as, the frontier of merged states becomes vacuous which then causes the rule system to stop and fail to accept the input, as shown in the alternative continuation from step 4 in Table 4 below. In fact, all accepted observation traces will match against a trace

Step	Obs.	Rule Activations	Resultant States
4	$\{\neg a, b\}$	$\{\{b, \neg a, r_b(r_{end})\}\}$	$\{\{\neg a, b, r_b(r_{end})\}\}$
5	$\{a, \neg b\}$	$\{\{b, \neg a, r_{end}\}\}$	$\{\}$

Table 4. Evaluation trace on *aaabba*

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 with production rule $S = ab \mid aSb$ in which r_{ab}'s actual argument represents the continuation string for concatenation to the string of a's generated.

Note, however, that the space requirements for each successive evaluation grows linearly while as are being observed; this is, of course, a heavy penalty for monitoring, but it is the cost for using just rule parameters. Example 21 gives a constant space rule system encoding.

Example 19. It is straightforward to establish that the class of context-free languages are a strict subset of RULER when parameterized by rules. Here we encode the language comprising words $a^n b^n c^n$ for $n \geq 1$. Let us first extend the above example to accept traces of the form $\{a, \neg b, \neg c\}^n \{\neg a, b, \neg c\}^n \{\neg a, \neg b, c\}^m$, for $n, m \geq 1$. We use the rule set

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

together with an initial activation set defined as $\{\{a, r_{ab}(r_c)\}\}$ and the final forbidden rule activation set defined as $\{r_{ab}, r_b, r_c, r_{fail}\}$. 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}). In a similar way we can encode the language $a^m b^n c^n$ for $n, m \geq 1$. Use the rules

$$
\begin{aligned}
r_a(\rho) &: \multimap b, \neg a, \neg c, \rho \mid a, \neg b, \neg c, r_a(\rho) \\
r_{bc}(\rho) &: \multimap c, \neg a, \neg b, \rho \mid b, \neg a, \neg c, r_{bc}(r_{c1}(\rho)) \\
r_{c1}(\rho) &: \multimap c, \neg a, \neg b, \rho \\
r_{end} &: \multimap r_{fail} \\
r_{fail} &: \multimap r_{fail}
\end{aligned}
$$

with an initial frontier set $\{\{a, r_a(r_{bc}(r_{end})\}\}$ and forbidden rule activation set $\{r_a, r_{bc}, r_{c1}, r_{fail}\}$.

Now we can encode the intersection of the rule systems accepting languages, $a^n b^n c^m$ and $a^m b^n c^n$ $(n, m \geq 1)$, to yield a rule system accepting the context-sensitive language $a^n b^n c^n$, for $n \geq 1$[7]. This is given as the union of the above rule sets and product of the initial frontier sets and union of the forbidden rule activation set. So, the product rule system has an initial activation set $\{\{a, r_{ab}(r_c), r_a(r_{bc}(r_{end}))\}\}$ and final forbidden rule activation set $\{r_{ab}, r_b, r_c, r_a, r_{bc}, r_{c1}, r_{fail}\}$.

5.2 Including data parameters in RuleR

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, as well as enabling encodings of monitoring formulas that can be more efficiently evaluated.

Example 20. 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

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

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

[7] represented, of course, as the set of traces $\{a, \neg b, \neg c\}^n \{\neg a, b, \neg c\}^n \{\neg a, \neg b, c\}^n$ for $n \geq 1$.

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.

Example 21. In contrast to the encoding of context-sensitive language $a^n b^n c^n$ for $n > 1$ given in Example 19, we provide a simpler and less memory expensive encoding using data parameters. The rule system first counts the number of as, then attempts to match the observation with n bs followed by n cs. Any mismatch on this requirement leads to non acceptance. The evaluation of the rule system requires constant space for each evaluation step in contrast to the previous given example that requires linear space. Informally, if the rule $r_a(n)$ is active then n as have been previously observed; if the rule $r_b(n, m)$ is active then n more bs are required and m cs; and if $r_c(m)$ is active then m cs are still required. The rule set is given as below.

$$\begin{array}{lll}
r_a(n): & \multimap & a, \neg b, \neg c, r_a(n+1) \mid \\
 & & \neg a, b, \neg c, r_b(n-1, n) \\
r_b(n,m): & \multimap & \neg a, b, \neg c, n > 0, r_b(n-1, m) \mid \\
 & & \neg a, \neg b, c, n = 0, m > 1, r_c(m-1) \mid \\
 & & \neg a, \neg b, c, n = 0, m = 1, r_{end} \\
r_c(m): & \multimap & \neg a, \neg b, c, m > 1, r_c(m-1) \mid \\
 & & \neg a, \neg b, c, m = 1, r_{end} \\
r_{end}: & \multimap & r_{fail} \\
r_{fail}: & \multimap & r_{fail}
\end{array}$$

The rule system has an initial frontier set given as $\{\{a, r_a(1)\}\}$ and the forbidden rule set is $\{r_a, r_b, r_c, r_{fail}\}$.

5.3 Using both data and rule parameters in RuleR

In [1], Alur, Etessami and Madhusudan introduced a temporal logic of nested calls and returns using abstract temporal modalities for skipping over states corresponding to procedure body invocations, e.g. an abstract next time operator that jumps from a call state to its matching return state. They hint at past time versions and other modalities. A recent paper of Rosu, Chen and Ball, [15] develop specialised run-time monitors for such a past time version of CaRet, called ptCaRet. For ptCaRet, it is assumed that observation states include, over and above other monitored information, at most one of the following propositions, *call*, *return*, *begin* and *end*. It is further assumed that *begin*-states immediately follow *call*-states, and *end*-states immediately precede *return*-states, and that observation traces are prefixes of traces with matched call and return states. Whilst it is clear that such context-free languages are encodable in the RuleR parameterized by rules, we wish RuleR to be thought of as a low-level system for effective encoding of run-time monitors described in a wide variety of logics and trace languages. In the next example, we show how a uniform and practically efficient encoding of such ptCaRet temporal operators can be given in RuleR.

Example 22. In this example, we first encode the abstract previous temporal operator introduced in [15]. We assume each observation state contains at most one of the propositions *call*, *begin*, *end* and *return*, and their ordering follows the requirements given in [15]. Table 5 depicts a possible observation trace over properties represented by the propositions b, p, q and r. For ease, we have separated out the states according to the nesting of call states. Assuming $\widehat{\odot}$ denotes the abstract yesterday temporal operator, then we have that:

Observation State Number										
1	2	3	4	5	6	7	8	9	10	11
b	*call* p								*return* p	
		begin q	*call* q				*return* q	*end* q		
				begin r	r	*end* r				

Table 5. A nested call return observation trace

(i) $\widehat{\odot} p$ is true only at the observation states numbered 3, 10 and 11;
(ii) $\widehat{\odot} q$ is true only at the observation states numbered 4, 5, 8 and 9;
(iii) $\widehat{\odot} r$ is true only at the observation states number 6 and 7.

With a strict abstract since operator defined by (the least solution of)

$$\phi \widehat{\mathcal{S}^-} \psi \equiv \widehat{\odot}(\psi \vee (\phi \wedge \phi \widehat{\mathcal{S}^-} \psi))$$

then we have that:

(i) $p \widehat{\mathcal{S}^-} b$ is true only at observation states 2, 3, 10 and 11;
(ii) $(p \vee q) \widehat{\mathcal{S}^-} b$ is true only at observation states 2, 3, 4, 5, 8, 9, 10 and 11.

In order to model the abstract temporal operators in RuleR one needs to determine matching calls and returns. We thus introduce a rule $level(n)$ that will record for each observation state the current call stack nesting level. We also need to determine the value of a given formula at some matching call observation state. We introduce the rule $RMC(n, x)$ that records whether the rule argument x was true on the most recent call at stack level n. We will then use the rule $AY(x)$ to encode the truth of abstract yesterday temporal operator, which is then defined by the following two rules.

$$\begin{array}{l} AYg1(x) : end, level(m), RMC(m-1, x) \multimap AY(x) \\ AYg2(x) : \neg end, x \multimap AY(x) \end{array}$$

The first rule determines the value of $AY(x)$ for a *return* observation state. Recall that an *end* state must immediately precede a *return* state. Therefore if the *end* state is currently at stack level m and the rule argument x was true at the matching call state for stack level $m-1$ then $AY(x)$ must hold in the return state (which for RuleR means that the rule must be present). On the other hand, if we are currently not at an end state and the rule argument x holds then $AY(x)$ must also hold in the next observation state.

There are three rules to maintain the call stack level counter, $level(n)$.

$$\begin{array}{l} Lg1 : call, level(n) \multimap \neg level(n), level(n+1) \\ Lg2 : end, level(n) \multimap \neg level(n), level(n-1) \\ Lg3 : \neg call, \neg end, level(n) \multimap level(n) \end{array}$$

The rule $RMC(n, x)$ is also defined using three rules.

$$\begin{array}{l} RMCg1(x) : return, level(n) \multimap \neg RMC(n, x) \\ RMCg2(x) : call, level(n), x \multimap RMC(n, x) \\ RMCg3(x) : RMC(n, x), level(m), n < m \multimap RMC(n, x) \end{array}$$

The first rule ensures that $RMC(n, x)$ is switched off for the state following a return state at stack level n. The second rule ensures $RMC(n, x)$ is switched on in the state following a call state at level n in which x is true. The third rule in the above group propagates $RMC(n, x)$ over more deeply nested observation states.

The above rule schema then need to be switched on. For example, if it is required to monitor the abstract temporal property $\widehat{\odot} p$ we would include the generator rule

$$\begin{array}{ll} rg : \multimap & rg, \\ & AYg1(p), AYg2(p), \\ & Lg1, Lg2, Lg3, \\ & RMCg1(p), RMCg2(p), RMCg3(p) \end{array}$$

with an initial frontier set of rules as $\{\{rg, Lg1, Lg3, AYg2(p), RMCg2(p), level(0)\}\}$.

The encoding of the abstract string since temporal operator can be given in a similar way using the following four generator rules for determining whether the rule name $AS(x,y)$ should be present in a rule activation state.

$$
\begin{aligned}
&ASg1(x,y) : \neg end, y \multimap AS(x,y)\\
&ASg2(x,y) : end, level(m), RMC(m-1,y) \multimap AS(x,y)\\
&ASg3(x,y) : \neg end, x, AS(x,y) \multimap AS(x,y)\\
&ASg4(x,y) : end, level(m), RMC(m-1,x), RMC(m-1, AS(x,y))\\
&\qquad\qquad \multimap AS(x,y)
\end{aligned}
$$

Whilst the above has shown the translation for a specific formula, a uniform translation can be based on these schema as was done for the translation of the past time fragment of LTL into RuleR.

6 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, informally, presented the version of 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 [10]. 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 proceeds over a trace. We are referring to such systems as reactive rule systems / grammars / Kripke structures [12]. 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 [14] is clear, however, a more detailed study of reactive grammars and their place in the complexity hierarchy is work in progress, see [7] for some initial results and examples.

By the very nature of our goal, namely to have RuleR as a low-level target language into which one can compile higher-level monitoring logics, we have kept RuleR syntactically small and concise. There are, however, commonly occurring patterns of rules where some additional syntax in RuleR could simplify the rule system presentation. One example would be to introduce the notion of "persistent" rule activations in addition to the basic single-step interpretation RuleR has for rule activation; this would certainly remove the need for the "generator" rules we have used throughout most of the examples. It is also quite common to find a collection of rules always being switched on, or off, together, for example, a set of rules characterising some reoccurring state. Hence another extension might be to introduce names referring to such sets of rules and then treating the new name in a similar way to a rule name.

An associated feature we haven't yet fully treated in RuleR is rule priority. 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 handling priority and preferences in default logic (defeasible reasoning) [8]. Of course, this changes the nature of the logics expressible quite considerably and is an area of future development. We note, also, that priorities would be very convenient for defining the notion of hierarchical state charts in which transitions leaving an outer (super) state have higher priority than transitions leaving an inner state.

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 [3], 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 compilation from full Eagle to RuleR remains to be developed.

A prototype Java implementation of RuleR, including data and rule parameters, has been developed, as a proof of concept. All examples in the paper have been run through the system. Furthermore, the translation algorithm for LTL into RuleR has also been implemented and, similarly, a version of the nested call return temporal logic has been implemented. Whilst it is too early to report on case study performance, the prototype is, so far, supporting our assertions that compiling to RuleR rather than into Eagle does lead to more efficient run-time monitoring. Further developments and refinements of the current prototype are in hand and include, for example, hierarchical compositions of RuleR monitors.

Acknowledgement

The authors are very grateful to Djihed Afifi for his contributions to the prototype RuleR implementation that's enabled initial experimentation and double check on the translations from temporal logic to RuleR.

References

1. R. Alur, K. Etessami and P. Madhusudan. A temporal logic of nested calls and returns. Proc. of the 10th TACAS, vol. 2988, LNCS, pp467-481, Springer, 2004.
2. J. Baran and H. Barringer. Forays into Sequential Composition and Concatenation in Eagle. *to appear in* Proc. of *Run-time Verification Workshop, RV2008, Budapest, Hungary*, LNCS, Vol. ??, pp??-??, Springer, 2008.

3. H. Barringer, M. Fisher, D. Gabbay, R. Owens and M. Reynolds. *The Imperative Future: Principles of Executable Temporal Logic*. Research Studies Press. 1996.
4. H. Barringer, M. Fisher, D. Gabbay, G. Gough and R. Owens. MetateM: An Introduction. Formal Aspects of Computing, Vol. 7, No. 5, pp533-549, Springer London, 1995.
5. H. Barringer, A. Goldberg, K. Havelund and K. Sen. Rule-Based Runtime Verification. Proc. of the *VMCAI'04, 5th Int. Conf. on Verification, Model Checking and Abstract interpretation*, Venice. Vol. 2937, LNCS, Springer-Verlag, 2004.
6. H. Barringer, A. Goldberg, K. Havelund and K. Sen. Run-time Monitoring in Eagle. Proc. of *PADTAD '04, Santa Fe, New Mexico*, IEEE Computer Society, IDPDS'04, Vol. 17, No. 17, pp264b, 2004.
7. H. Barringer, D. Rydeheard and D. Gabbay. Reactive Grammars: An Initial Exploration. Draft paper, see http://www.cs.man.ac.uk/ david/reactive.html, 2007.
8. G. Brewka. Reasoning about Priorities in Default Logic. Proc. of *AAAI National Conference on Artificial Intelligence*, vol. 2, pp940-945, The AAAI Press/The MIT Press, 1994.
9. M.D. Fisher. A Normal Form for Temporal Logics and its Applications in Theorem-Proving and Execution. Journal of Logic and Computation, Vol. 7, No. 4, pp429–456, 1997.
10. B. Finkbeiner and H. Sipma. Checking Finite Traces Using Alternating Automata. Formal Methods in System Design, Vol. 24, No. 2, pp101–127, Springer Netherlands, 2004.
11. D.M. Gabbay. Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. Proc. of *Coll. on Temporal Logic in Specification, Altrincham*, Vol. 398, LNCS, pp67–89, Springer-Verlag, 1989.
12. D.M. Gabbay. Introducing Reactive Kripke Semantics and Arc Accessibility. To appear in Festschrift in Honour of Boaz Traktenbrot, 2007
13. K. Havelund. Runtime verification of C programs. Submitted for conference publication, January 2008.
14. E. Moriya, D. Hofbauer, M. Huber and F. Otto. On State-Alternating Context-Free Grammars. Theoretical Computer Science, Vol. 337, No. 11, pp183–216, 2005.
15. G. Rosu, F. Chen and T. Ball. Synthesising Monitors for Safety Properties — This Time With Calls and Returns — *to appear in* Proc. of *Run-time Verification Workshop, RV2008, Budapest, Hungary*, LNCS, Vol. ??, pp??-??, Springer, 2008.