Forays into Sequential Composition and Concatenation in EAGLE

Joachim Baran and Howard Barringer

The University of Manchester, School of Computer Science, Oxford Road,
Manchester, M13 9PL, United Kingdom,
{joachim.baran,howard.barringer}@cs.manchester.ac.uk

Abstract. The run-time verification logic EAGLE is equipped with two forms of binary cut operator, sequential composition (;) and concatenation ($\cdot$). Essentially, a concatenation formula $F_1 \cdot F_2$ holds on a trace if that trace can be cut into two non-overlapping traces such that F_1 holds on the first and F_2 on the second. Sequential composition differs from concatenation in that the two traces must overlap by one state. Both cut operators are non-deterministic in the sense that the cutting point is not uniquely defined. In this paper we establish that sequential composition and concatenation are equally expressive. We then extend EAGLE with deterministic variants of sequential composition and concatenation. These variants impose a restriction on either the left or right operand so that the cut point defines either the shortest or longest possible satisfiable cut trace. Whilst it is possible to define such deterministic operators recursively within EAGLE, such definitions based on the non-deterministic cut operators impose a complexity penalty. By augmenting EAGLE's evaluation calculus for the deterministic variants, we establish that the asymptotic time and space complexity of on-line monitoring for the variants with deterministic restrictions applied to the left operand is no worse than the asymptotic time and space complexity of the sub-formulæ.

1 Introduction

Although common temporal logics like propositional temporal logic, extended temporal logic and the modal μ-calculus are quite expressive [Wol83,Koz83], they define no operator analog to the most common principle in imperative programming: sequential composition. Sequential composition allows one to glue two traces together, where the last state of the first trace overlaps with the first state of the second trace. A sequential composition formula is then satisfied at the start of a trace, if the trace can be cut into two sub-traces, overlapping as above, on which both its operands hold respectively.

Concatenation defines the cut of the trace such that there is no overlapping part, so the two traces butted together form the original trace. Even though

sequential composition and concatenation seem to be closely related, this has not been formally investigated yet.[1]

In this paper the runtime verification logic EAGLE is examined [BGHS04b]. EAGLE is a temporal fixed-point logic on finite traces. EAGLE is able to perform efficient on-line monitoring without storing the execution trace [BGHS04b]. In EAGLE, both sequential composition and concatenation are part of the logics language. We show that sequential composition can be expressed in terms of concatenation and vice-versa. We then extend EAGLE with deterministic variants of concatenation and sequential composition. These variants impose a restriction on either the left or right operand so that the cut point defines either the shortest or longest possible satisfiable cut trace. We augment EAGLE's evaluation calculus by the new operators and establish that the asymptotic space complexity of on-line monitoring for the variants with restrictions applied to the left operand is no worse than the asymptotic space complexity of the sub-formulæ. Two examples now follow to provide motivation for the deterministic cut operators.
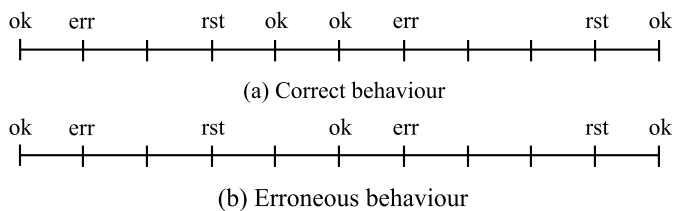

(a) Correct behaviour

(b) Erroneous behaviour

Fig. 1. Traces of a fail-safe system (Example 1)

Example 1. Consider a fail-safe system that in the occurrence of an error eventually resets itself and enters a predefined "good" system state in that way. In Figure 1(a), an acceptable observation trace is depicted, where "ok" denotes that the system is in a good state, "err" denotes the occurrence of an error and "rst" denotes a reset of the system. We allow that a reset can occur with a finite delay after an error has occured. We can formulate this behaviour by the following specification:

$$\textbf{max } \text{ErrHandler}(\textbf{Form } F) = \lceil F \rceil \cdot (\lfloor \text{err} \wedge \text{Eventually}(\text{rst}) \rfloor \cdot \text{ErrHandler}(F))$$
$$\textbf{mon } \text{FailSafe} = \text{ErrHandler}(\text{Always}(\text{ok}))$$

The formulæ $\lceil F_1 \rceil \cdot F_2$ and $\lfloor F_1 \rfloor \cdot F_2$ are constraint variants of the concatenation formula $F_1 \cdot F_2$, where the cut has to be placed so that there is no longer or shorter sub-trace satisfying F_1, respectively. For the specification above it means

---

[1] In [CHMP81], concatenation was defined in terms of sequential composition. This cannot be done in EAGLE so easily, which is shown in the following. Instead of referring to the operators as sequential composition and concatenation, Chandra *et al.* referred to them as "chop" and "chomp" respectively.

that a trace without erroneous behaviour is completely labelled with "ok"s. If an error occurs, i.e. a state labelled by "err", then the good behaviour resumes after a reset, i.e. "rst".

While this specification can also be written without formulæ of the form $\lceil F_1 \rceil \cdot F_2$ and $\lfloor F_1 \rfloor \cdot F_2$, specification without these the new operators are not necessarily as succinct as specification that make use of $\lceil F_1 \rceil \cdot F_2$ and $\lfloor F_1 \rfloor \cdot F_2$, and furthermore, will incur a significant monitoring cost penalty for the compositional recodings.

Example 2. We introduce a *conditional concatenation operator*, $\lfloor F_1 \rfloor \overrightarrow{\cdot} F_2$, based on the operator $\lfloor F_1 \rfloor \cdot F_2$ which we used in the previous example. Let $\lfloor F_1 \rfloor \overrightarrow{\cdot} F_2$ be the syntactic abbreviation for $\neg(\lfloor F_1 \rfloor \cdot \mathbf{True}) \vee (\lfloor F_1 \rfloor \cdot F_2)$. Informally, $\lfloor F_1 \rfloor \overrightarrow{\cdot} F_2$ can be interpreted as "whenever F_1 matched, do F_2 afterwards".

Consider a nested locking pattern, where we wish to detect when a thread t takes a lock l_1 and does not release it until t has taken a different lock l_2, after which we verify another property φ. Using the newly defined operator, we can formulate the corresponding specification as

$$Always(\lfloor lock(t, l_1) \wedge Until(\neg release(t, l_1), lock(t, l_2) \wedge l_1 \neq l_2) \rfloor \overrightarrow{\cdot} \varphi$$

The latter specification is not an EAGLE monitoring formula, since data parametrisation in EAGLE is bound to evaluating the current state. However, we can formulate a semantically equivalent monitoring formula in EAGLE:

mon NestedLck = Always(isLock() $\rightarrow$ Nested(getThread(), getLock()))

with the rule definition

max Nested(int t, int l) = $\lfloor$Until($\neg$release(t, l), isLock() $\wedge$ getLock() $\neq$ l)$\rfloor \overrightarrow{\cdot} \varphi$

Since EAGLE is implemented in Java, we rely on the methods isLock(), getLock(), getThread() and release() with the obvious semantics and we use integers as handles for threads and locks. It should be noted that getLock() returns the last lock obtained by the current thread, so that its return value when called in the monitoring formula NestedLck and its return value when called in the rule Nested(...) eventually differ.

The shortest trace-length restriction in the concatenation formula of the rule definition Nested(...) ensures that we match the first occurrence of a newly obtained lock, i.e. the rule parameter l and the return value of getLock() differ,where it is also ensured that the previous lock is not released yet.

The paper is structured as follows. A formal definition of EAGLE is given in Section 2. In Section 3 it is proven that sequential composition and concatenation are definable in terms of each other. In Section 4 EAGLE is extended by deterministic cut operators, where it is shown that those operators are definable in unextended EAGLE. In Section 5 EAGLE's calculus is extended by the deterministic cut operators and it is proven that deterministic cut operators enable more efficient on-line monitoring. Section 6 concludes our work.

2 Preliminaries

EAGLE is a temporal logic based on recursively defined temporal predicates
(rules) with four primitive temporal operators, $\bigcirc$, $\odot$, $\cdot$, $;$. Formally:

Definition 1. *Specifications in* EAGLE *are formed by a pair $\langle D, O \rangle$, where D
is the declaration part and O the observer part. Rule definitions R define named
parametrised rules N. Monitors M specify the requirements.*

$$D ::= R^* \qquad\qquad\qquad\qquad\qquad\qquad O ::= M^*$$

$$R ::= \{\mathbf{min} \,|\, \mathbf{max}\} \; N(T_1 \; x_1, \ldots, T_n \; x_n) = F \qquad M ::= \mathbf{mon} \; N = F$$

$$T ::= \mathbf{Form} \,|\, primitive \; type$$

$$F ::= \mathbf{False} \,|\, \mathbf{True} \,|\, x_i \,|\, expression \,|\, \neg F \,|\, F_1 \vee F_2 \,|\, \bigcirc F \,|\, \odot F \,|$$
$$F_1 \cdot F_2 \,|\, F_1 \,;\, F_2 \,|\, N(F_1, \ldots, F_n)$$

In the following, we use standard operators of propositional logic that are
defined by De Morgan's laws.

Formulæ are evaluated over discrete finite traces of observation states. A
sequence of states $s_1, s_2, \ldots, s_n$ constitutes a trace σ of length $|\sigma| = n$. In order to
keep track of the positions on the trace, states will be enumerated incrementally
starting with one. $\sigma^{[i,j]}$ denotes then the sub-trace $s_i, s_{i+1}, \ldots, s_j$ of a trace σ.
For sub-traces, the numbering of states will again begin from one. We write $\sigma(i)$
to denote the i-th state of the trace. The empty trace, i.e. the trace of length 0,
is abbreviated as ε.

Definition 2. *For trace $\sigma = s_1 s_2 \ldots s_{|\sigma|}$, the satisfiability relation $\sigma, i \models_D F$,
with $0 \leq i \leq |\sigma| + 1$, is defined as*

$$\sigma, i \models_D expression \quad \text{iff} \quad 1 \leq i \leq |\sigma| \text{ and } evaluate(expression)(\sigma(i)) == true$$

$$\sigma, i \models_D \bigcirc F \quad \text{iff} \quad i \leq |\sigma| \text{ and } \sigma, i+1 \models_D F$$

$$\sigma, i \models_D \odot F \quad \text{iff} \quad 1 \leq i \text{ and } |\sigma| \geq 1 \text{ and } \sigma, i-1 \models_D F$$

$$\sigma, i \models_D F_1 \cdot F_2 \quad \text{iff} \quad \exists j. i \leq j \leq |\sigma| + 1 \text{ and}$$
$$\sigma^{[1,j-1]}, i \models_D F_1 \text{ and } \sigma^{[j,|\sigma|]}, 1 \models_D F_2$$

$$\sigma, i \models_D F_1 \,;\, F_2 \quad \text{iff} \quad \exists j. i < j \leq |\sigma| + 1 \text{ and}$$
$$\sigma^{[1,j-1]}, i \models_D F_1 \text{ and } \sigma^{[j-1,|\sigma|]}, 1 \models_D F_2$$

$$\sigma, i \models_D N(F_1, \ldots, F_n) \quad \text{iff} \quad \begin{cases} \text{if } 1 \leq i \leq |\sigma| \text{ then} \\ \quad \sigma, i \models_D F[F_1/x_1, \ldots, F_n/x_n], where \\ \quad (N(T_1 \; x_1, \ldots, T_n \; x_n) = F) \in D \\ \text{if } i = 0 \text{ or } i = |\sigma| + 1 \text{ then} \\ \quad \text{if } (\mathbf{max} \; N(T_1 \; x_1, \ldots, T_n \; x_n) = F) \in D \text{ then} \\ \quad\quad \sigma, i \models_D \mathbf{True}, \\ \quad \text{if } (\mathbf{min} \; N(T_1 \; x_1, \ldots, T_n \; x_n) = F) \in D \text{ then} \\ \quad\quad \sigma, i \models_D \mathbf{False} \end{cases}$$

and the propositional constants and operators are defined in the obvious way.

If the set of declarations D follows from the context, then $\models$ is used instead of $\models_D$. In formulæ where an expression can be chosen arbitrarily, i.e. it is treated as a propositional variable, the evaluation is simplified to $evaluate(p)(\sigma(i)) ==$ $true$, where p denotes a propositional variable and p is **True** at $\sigma(i)$.

Remark: It should be noted that at trace boundaries, i.e. the absent states at index 0 and $|\sigma| + 1$, only the logical constant **True** and maximal defined rules evaluate to true, while all other formulæ including tautologies evaluate to **False**. Once the trace has been left, i.e. a step has been made onto the boundary of the trace, it is possible to step back into the trace, but stepping beyond the boundary (stepping to indices -1 and $|\sigma| + 2$) evaluates to **False**.

A specification $\langle D, O \rangle$ is satisfied by a trace σ if all monitoring formulæ of the specification are satisfied on σ. Each monitoring formula is evaluated from position one, regardless of the trace length. A trace is said to model a specification, if the specification is satisfied by the trace. The latter we denote by $\sigma \models \langle D, O \rangle$.

Definition 3. *A given trace σ satisfies a specification $\langle D, O \rangle$ if all monitoring formulæ hold on the trace from position one, i.e. $\sigma \models \langle D, O \rangle$ iff $\forall(\textbf{mon } N = F) \in O. \ \sigma, 1 \models_D F$*

In the remainder of the paper, the rule **max** Limit() = **False** is assumed to be part of every specification. It evaluates to **True** on the boundaries of a trace, i.e. when the current state is either 0 or $|\sigma| + 1$, otherwise it is **False**.

3 Interdefinability of Sequential Composition and Concatenation

Interdefinability of operators, i.e. expressibility of an operator in terms of another operator due to syntactical transformations, simplify definitions, proofs and implementations of a logic. A proof or implementation has only to focus on one of the operators then, where the obtained results can be carried forward to other operators.

For the logic EAGLE, we show that sequential composition and concatenation are equally expressive. Hence, sequential composition can be syntactically formulated in terms of concatenation and vice-versa. In Section 3.1 below we define sequential composition recursively in terms of concatenation. The other direction, however, is not so straightforward: Section 3.2 outlines our elimination procedure and argues its correctness.

3.1 Sequential Composition in Terms of Concatenation

A sequential composition formula $F_1 \ ; \ F_2$ can be expressed in terms of concatenation by simulation of the former operator's semantics using a fixed-point rule definition. We define and add the new rule **min** SequentialComposition(**Form** F_1, **Form** F_2) to every specification. The sequential composition operator can then

be removed from arbitrary formulæ, by substituting each sub-formula of the form $F_1 \, ; \, F_2$ by an application of the rule SequentialComposition(F_1, F_2) ,where the rule is given as:

$$\textbf{min} \; \text{SequentialComposition}(\textbf{Form } F_1, \textbf{Form } F_2) =$$
$$(((F_1 \wedge \bigcirc \text{Limit}()) \cdot \textbf{True}) \wedge (\text{Limit}() \cdot (F_2 \wedge \odot \text{Limit}())))\, \vee$$
$$\bigcirc \text{SequentialComposition}(\odot F_1, F_2)$$

We defined the rule SequentialComposition(F_1, F_2) as a minimal fixed-point, so that it will not be satisfied on the empty trace or the boundaries of a trace. This behaviour coincides with the semantics of the sequential composition operator. For non-empty traces, the first application of the rule body splits the trace, so that F_1 is evaluated on a sub-trace with its boundary in the next state and F_2 is evaluated on a sub-trace with its boundary in the previous state. Hence, the evaluation of F_1 and F_2 overlaps at the index at which the rule is evaluated. Additionally, the rule body contains a recursion $\bigcirc$SequentialComposition$(\odot F_1, F_2)$ that repeats the just described splitting of the trace, but now the sub-trace boundary for F_1 is shifted one index to the right, and likewise, the evaluation of F_2 begins one index later. The recursion finally terminates when the boundary of the trace is reached, on which SequentialComposition(F_1, F_2) was first invoked.

Theorem 1.[2] *For every formula F of* EAGLE*, we can give a semantically equivalent formula F' of* EAGLE*, where F' contains no sequential composition subformula.*

This result can be carried forward to any arbitrary EAGLE-specification, where one subsequently replaces occurrences of sequential composition formulæ – from innermost sub-formulæ to outermost sub-formulæ.

3.2   Concatenation in Terms of Sequential Composition

Concatenation can be expressed in terms of sequential composition as well. However, due to the semantics of the concatenation operator, there is no single substitution mechanism for substituting all occurrences of concatenation sub-formulæ by equivalent sequential composition sub-formulæ. For concatenation, one or even both operands can hold on the empty trace, while sequential composition requires that its operands hold on sub-traces of non-zero length. Therefore a substitution of a concatenation sub-formula by an equivalent sequential composition formula has to take into account that one or both of the concatenation's sub-formulæ might hold on the empty trace. Depending on which of the two operands of concatenation sub-formulæ can hold on the empty trace, different sequential composition formulæ have to be substituted.

In the following, it will be proven that for a given EAGLE formula, it can be determined if it holds on the empty trace (Lemma 1). From this particular result

---

[2] We omit most proofs in this paper due to page number restrictions. The full proofs were included for the review of the paper and can be obtained from the authors.

it follows immediately that concatenation is expressible in terms of sequential composition, such that for each combination of concatenation sub-formulæ which may or may not hold on the empty trace, a suitable sequential composition formula can be substituted (Theorem 2).

We show that it is sufficient to inspect an EAGLE-formula syntactically, in order to verify whether it would be satisfied on the empty trace or not. More importantly, rule applications do not have to be substituted by their rule bodies at any point, which would otherwise lead to undecidability of the problem. The latter is due to the possible encoding of a Turing-machine or equivalent device in EAGLE.[3]

Lemma 1. *For an arbitrary formula in* EAGLE, *it is decidable whether it is satisfiable on the empty trace.*

Proof. For an arbitrary formula we can inductively determine whether it holds on the empty trace or not.

Base cases: The formulæ **False**, *expression*, $\bigcirc F$, $\odot F$, $F_1 \,;\, F_2$ and $N(\ldots)$, with $(\mathbf{min}\ N(T_1\ x_1, \ldots, T_n\ x_n) = F) \in D$, are not satisfied on the empty trace, whereas **True**, and $N(\ldots)$, with $(\mathbf{max}\ N(T_1\ x_1, \ldots, T_n\ x_n) = F) \in D$, are satisfied on the empty trace.

Inductive step: The formulæ $F_1 \wedge F_2$ and $F_1 \cdot F_2$ are satisfied on the empty trace, iff F_1 and F_2 are satisfied on the empty trace. $\neg F$ is satisfied on the empty trace, when F is not satisfied on the empty trace. $\qquad\square$

We give a translation from any formula $F_1 \cdot F_2$ to an equivalent concatenation-free formula, which is parametrised by which of the operands F_1 and F_2 are satisfiable on the empty trace. An arbitrary formula $F_1 \cdot F_2$ is substituted by

$$
\begin{aligned}
\psi \quad &\text{iff} \quad \varepsilon, 1 \models \neg F_1 \wedge \neg F_2, \\
\psi \vee F_1 \quad &\text{iff} \quad \varepsilon, 1 \models \neg F_1 \wedge F_2, \\
\psi \vee (\odot\mathrm{Limit}() \wedge F_2) \quad &\text{iff} \quad \varepsilon, 1 \models F_1 \wedge \neg F_2, \\
\psi \vee F_1 \vee (\odot\mathrm{Limit}() \wedge F_2) \vee \mathrm{Limit}() \quad &\text{iff} \quad \varepsilon, 1 \models F_1 \wedge F_2,
\end{aligned}
$$

where $\psi \equiv (F_1 \,;\, (\bigcirc^2\mathrm{Limit}() \,;\, F_2)) \vee \odot(\bigcirc(F_1 \wedge \mathrm{Limit}()) \,;\, (\bigcirc^2\mathrm{Limit}() \,;\, F_2))$.

Theorem 2. *For every formula F of* EAGLE, *we can give a semantically equivalent formula F' of* EAGLE, *where F' contains no concatenation sub-formula.*

Again, this result can be carried forward to any arbitrary EAGLE-specification, where one subsequently replaces occurrences of concatenation formulæ – from innermost sub-formulæ to outermost sub-formulæ.

---

[3] It is in fact straightforward to implement a Minsky machine in EAGLE, which is Turing complete [Min61].

4 Deterministic Cut Operators

Both sequential composition and concatenation allow a trace to be split non-de-terministically, i.e. due to the semantics of the operators, several cut positions may satisfy a formula $F_1 \, ; \, F_2$ or $F_1 \cdot F_2$ on a given trace. The designer of a monitoring specification may however desire a unique position of the cut, i.e. a deterministic choice of where a trace is being cut.

In the following, mixfix operators are introduced which allow us to express deterministic cuts in specifications. These operators extend sequential composition and concatenation by additionally verifying that there is no shorter, respectively longer, sub-trace on which the sub-formula holds. It is shown that all deterministic cut operators can be formulated in unextended EAGLE. Even though the operators do not increase EAGLE's expressiveness, we show in Section 5 that the new operators enable more efficient on-line monitoring.

4.1 Syntax and Semantics of Deterministic Cut Operators

EAGLE with deterministic cut operators extends the syntax of Definition 1. For brevity just the new BNF production F is given. The other productions are left unchanged.

Definition 4. EAGLE$_{[]}$ *denotes an extension of* EAGLE *with additional mixfix operators, where the production F of Definition 1 is replaced by*

$$F ::= \textbf{False} \,|\, \textbf{True} \,|\, x_i \,|\, expression \,|\, \neg F \,|\, F_1 \vee F_2 \,|\, \bigcirc F \,|\, \odot F \,|$$
$$\quad F_1 \circ F_2 \;\;|\;\; \lfloor F_1 \rfloor \circ F_2 \;\;|\;\; \lceil F_1 \rceil \circ F_2 \;\;|\;\; F_1 \circ \lfloor F_1 \rfloor \;\;|\;\; F_1 \circ \lceil F_1 \rceil \;\;|\;\; N(F_1, \ldots, F_n)$$
$$\circ ::= \; ; \; | \cdot$$

In conjunction with a concatenation or sequential composition operator, we write $\lfloor F \rfloor$ and $\lceil F \rceil$ to denote that F is only satisfied on its respectively shortest and longest sub-trace of all the sub-traces that satisfy the unrestricted F. In the following, we will then refer to $\lfloor F \rfloor$ and $\lceil F \rceil$ as the minimally and maximally trace length restricting formulæ, respectively. As with the definition of EAGLE$_{[]}$'s syntax, only the extensions to EAGLE's semantics is given.

Definition 5. *On traces $\sigma = s_1 s_2 \ldots s_{|\sigma|}$ the satisfiability relation $\sigma, i \models_D F$, with $0 \leq i \leq |\sigma| + 1$, is extended by*

$$\sigma, i \models_D \lfloor F_1 \rfloor \cdot F_2 \;\; \text{iff} \;\; \exists j. i \leq j \leq |\sigma| + 1 \text{ and } \sigma^{[1,j-1]}, i \models_D F_1 \text{ and}$$
$$\sigma^{[j,|\sigma|]}, 1 \models_D F_2 \text{ and } \neg \exists k. i - 1 \leq k < j - 1 \text{ and } \sigma^{[1,k]}, i \models F_1$$

$$\sigma, i \models_D \lceil F_1 \rceil \cdot F_2 \;\; \text{iff} \;\; \exists j. i \leq j \leq |\sigma| + 1 \text{ and } \sigma^{[1,j-1]}, i \models_D F_1 \text{ and}$$
$$\sigma^{[j,|\sigma|]}, 1 \models_D F_2 \text{ and } \neg \exists k. j \leq k \leq |\sigma| \text{ and } \sigma^{[1,k]}, i \models F_1$$

$$\sigma, i \models_D F_1 \cdot \lfloor F_2 \rfloor \;\; \text{iff} \;\; \exists j. i \leq j \leq |\sigma| + 1 \text{ and } \sigma^{[1,j-1]}, i \models_D F_1 \text{ and}$$
$$\sigma^{[j,|\sigma|]}, 1 \models_D F_2 \text{ and } \neg \exists k. j < k \leq |\sigma| + 1 \text{ and } \sigma^{[k,|\sigma|]}, 1 \models F_2$$

$$\sigma, i \models_D F_1 \cdot \lceil F_2 \rceil \;\; \text{iff} \;\; \exists j. i \le j \le |\sigma| + 1 \text{ and } \sigma^{[1,j-1]}, i \models_D F_1 \text{ and}$$
$$\sigma^{[j,|\sigma|]}, 1 \models_D F_2 \text{ and } \neg \exists k. 1 \le k < j \text{ and } \sigma^{[k,|\sigma|]}, 1 \models F_2$$

$$\sigma, i \models_D \lfloor F_1 \rfloor \,;\, F_2 \;\; \text{iff} \;\; \exists j. i < j \le |\sigma| + 1 \text{ and } \sigma^{[1,j-1]}, i \models_D F_1 \text{ and}$$
$$\sigma^{[j-1,|\sigma|]}, 1 \models_D F_2 \text{ and } \neg \exists k. i \le k < j-1 \text{ and } \sigma^{[1,k]}, i \models F_1$$

$$\sigma, i \models_D \lceil F_1 \rceil \,;\, F_2 \;\; \text{iff} \;\; \exists j. i < j \le |\sigma| + 1 \text{ and } \sigma^{[1,j-1]}, i \models_D F_1 \text{ and}$$
$$\sigma^{[j-1,|\sigma|]}, 1 \models_D F_2 \text{ and } \neg \exists k. j \le k \le |\sigma| \text{ and } \sigma^{[1,k]}, i \models F_1$$

$$\sigma, i \models_D F_1 \,;\, \lfloor F_2 \rfloor \;\; \text{iff} \;\; \exists j. i < j \le |\sigma| + 1 \text{ and } \sigma^{[1,j-1]}, i \models_D F_1 \text{ and}$$
$$\sigma^{[j-1,|\sigma|]}, 1 \models_D F_2 \text{ and } \neg \exists k. j \le k \le |\sigma| \text{ and } \sigma^{[k,|\sigma|]}, 1 \models F_2$$

$$\sigma, i \models_D F_1 \,;\, \lceil F_2 \rceil \;\; \text{iff} \;\; \exists j. i < j \le |\sigma| + 1 \text{ and } \sigma^{[1,j-1]}, i \models_D F_1 \text{ and}$$
$$\sigma^{[j-1,|\sigma|]}, 1 \models_D F_2 \text{ and } \neg \exists k. 1 \le k < j-1 \text{ and } \sigma^{[k,|\sigma|]}, 1 \models F_2$$

We depict three applications of the deterministic cut operators in Figure 2 below, where we show the evaluation of $\sigma, 1 \models \text{Eventually}(\text{err}) \,;\, \lceil \text{rst} \rceil$, $\sigma, 1 \models \varphi$ and $\sigma, 1 \models \psi$ on an example trace σ as shown below:

$$\varphi \equiv \lfloor \text{Eventually}(\text{err} \wedge \text{Eventually}(\text{rst})) \rfloor \cdot \text{ok}$$
$$\psi \equiv \textbf{True} \cdot \lceil \lfloor \text{err} \wedge \text{Eventually}(\text{rst}) \rfloor \cdot \text{ok} \rceil$$
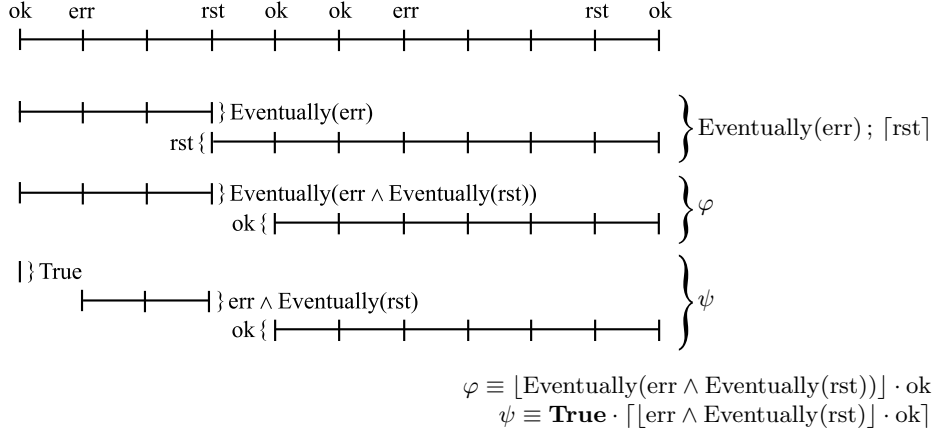
Fig. 2. Examples of deterministic cut operator applications

Remark: It should be noted that while $\sigma, 1 \models \text{Eventually}(\text{err}) \,;\, \lceil \text{rst} \rceil$ is satisfied on the example trace, we have $\sigma, 1 \not\models \text{Eventually}(\text{err}) \,;\, \lceil \text{ok} \rceil$. The longest sub-trace on which "ok" is satisfied is the whole trace, but for that cut the left-hand formula Eventually(err) is not true.

4.2 Definability of Deterministic Cut Operators in EAGLE

It is not apparent whether the mixfix variants of sequential composition and concatenation can also be defined in EAGLE. In the following it will be shown that EAGLE$_{[]}$ is not more expressive than EAGLE. The translation of the maximal

mixfix operators into EAGLE is given first, followed by the translation for the minimal mixfix operators.

For the maximal mixfix-operators, we will use the rules[4]

min NonMtMxLT(**Form** F_1, **Form** F_2) =
$\quad((((F_1 \wedge \bigcirc \text{Limit}()) \cdot F_2) \rightarrow \neg((F_1 \wedge \text{Eventually}(\bigcirc^2 \text{Limit}())) \cdot \textbf{True})) \vee$
$\quad\quad \bigcirc \text{NonMtMxLT}(\odot F_1, F_2,),$

min NonMtMxRT(**Form** F_1, **Form** F_2) =
$\quad((F_1 \cdot (F_2 \wedge \bigcirc \text{Limit}())) \rightarrow \neg(\textbf{True} \cdot (F_2 \wedge \text{Eventually}(\bigcirc^2 \text{Limit}())))) \vee$
$\quad\quad \bigcirc \text{NonMtMxRT}(F_1, \odot F_2),$

min NonMtMxOvrlpngLT(**Form** F_1, **Form** F_2) =
$\quad((((F_1 \wedge \bigcirc \text{Limit}()) \, ; \, F_2) \rightarrow \neg((F_1 \wedge \text{Eventually}(\bigcirc^2 \text{Limit}())) \, ; \, \textbf{True})) \vee$
$\quad\quad \bigcirc \text{NonMtMxOvrlpngLT}(\odot F_1, F_2,),$

min NonMtMxOvrlpngRT(**Form** F_1, **Form** F_2) =
$\quad((F_1 \, ; \, (F_2 \wedge \bigcirc \text{Limit}())) \rightarrow \neg(\textbf{True} \, ; \, (F_2 \wedge \text{Eventually}(\bigcirc^2 \text{Limit}())))) \vee$
$\quad\quad \bigcirc \text{NonMtMxOvrlpngRT}(F_1, \odot F_2),$

in order to denote the semantics of $\lceil F_1 \rceil \cdot F_2$, $F_1 \cdot \lceil F_2 \rceil$, $\lceil F_1 \rceil \, ; \, F_2$ and $F_1 \, ; \, \lceil F_2 \rceil$ on non-empty traces, respectively. Since $\lceil F_1 \rceil \cdot F_2$ and $F_1 \cdot \lceil F_2 \rceil$ could be satisfiable on the empty trace, their corresponding rules in the respective translations have to be accompanied by a formula that explicitly handles the formulæ holding on the empty trace.

We outline the semantics of NonMtMxLT(F_1, F_2) only, since the semantics of the remaining rules can be explained similarly. When NonMtMxLT(F_1, F_2) is substituted for $\lceil F_1 \rceil \cdot F_2$, the first invocation of its rule body will cause a cut of the form $(((F_1 \wedge \bigcirc \text{Limit}()) \cdot F_2) \rightarrow \neg((F_1 \wedge \text{Eventually}(\bigcirc^2 \text{Limit}())) \cdot \textbf{True}))$. The sub-formula $(F_1 \wedge \bigcirc \text{Limit}()) \cdot F_2$ denotes that the cut is enforced so that the right boundary of the left-subtrace follows immediately the current index at which the rule body is evaluated at. Then, the implication following the formula $\neg((F_1 \wedge \text{Eventually}(\bigcirc^2 \text{Limit}())) \cdot \textbf{True}))$ assures that F_1 is not satisfied on any sub-trace for which the cut is made further to the right. Alternatively, the rule body enters a recursion due to the disjunctive formula $\bigcirc \text{NonMtMxLT}(\odot F_1, F_2)$. With each recursion, the cut is moved one index further to the right, where the recursion terminates as soon as the boundary of the trace under inspection is reached.

With these rule definitions, the maximal mixfix operators can be expressed in EAGLE as

$$\lceil F_1 \rceil \cdot F_2 \equiv (((F_1 \wedge \text{Limit}()) \cdot F_2) \rightarrow \neg((F_1 \wedge \neg \text{Limit}()) \cdot \textbf{True})) \vee$$
$$\text{NonMtMxLT}(F_1, F_2)$$
$$F_1 \cdot \lceil F_2 \rceil \equiv ((F_1 \cdot (F_2 \wedge \text{Limit}())) \rightarrow \neg(\textbf{True} \cdot (F_2 \wedge \neg \text{Limit}()))) \vee$$
$$\text{NonMtMxRT}(F_1, F_2)$$
$$\lceil F_1 \rceil \, ; \, F_2 \equiv \text{NonMtMxOvrlpngLT}(F_1, F_2)$$
$$F_1 \, ; \, \lceil F_2 \rceil \equiv \text{NonMtMxOvrlpngRT}(F_1, F_2)$$

---

[4] NonMtMxLT spells out as **N**on**E**mpty**M**a**x**imal**L**eft**T**race, etc.

Theorem 3. *For each of the formulæ $\lceil F_1 \rceil \cdot F_2$, $F_1 \cdot \lceil F_2 \rceil$, $\lceil F_1 \rceil \,;\, F_2$ and $F_1 \,;\, \lceil F_2 \rceil$ of* EAGLE$_{[]}$ *there exists a semantically equivalent formula in* EAGLE.

For the minimal mixfix-operators, the translations are much simpler. Here, we only need an additional rule

$$\textbf{min } \text{ShorterNonEmptyTrace}(\textbf{Form } F) =$$
$$((F \wedge \bigcirc \text{Limit}()) \cdot \bigcirc \textbf{True}) \vee \bigcirc \text{ShorterNonEmptyTrace}(\odot F)$$

which is satisfied when there is a shorter non-empty sub-trace of the current trace under inspection on which F is satisfied. In the actual translation, it is then sufficient to verify whether the restricted sub-formula cannot be satisfied on a shorter sub-trace. For example, $\lfloor F_1 \rfloor \cdot F_2$ becomes $(F_1 \wedge \neg \text{ShorterNonEmptyTrace}(F_1)) \cdot F_2$, which reflects the semantics of $\lfloor F_1 \rfloor \cdot F_2$ under the assumption that F_1 is not satisfied on the empty trace. Since for mixfix concatenation formulæ it is the case that the trace length restricted formula can also be satisfied on the empty trace, we have to add a formula to the translations which explicitly addresses this.

When also considering the minimal mixfix operators' semantics on the empty trace, we get the following translations into EAGLE:

$$\lfloor F_1 \rfloor \cdot F_2 \equiv ((F_1 \wedge \text{Limit}()) \cdot F_2) \ \vee$$
$$(((F_1 \wedge \neg \text{ShorterNonEmptyTrace}(F_1)) \cdot F_2) \rightarrow \neg(F_1 \wedge \text{Limit}() \cdot \textbf{True}))$$
$$F_1 \cdot \lfloor F_2 \rfloor \equiv (F_1 \cdot (F_2 \wedge \text{Limit}())) \ \vee$$
$$((F_1 \cdot (F_2 \wedge \neg \text{ShorterNonEmptyTrace}(F_2))) \rightarrow \neg(\textbf{True} \cdot (F_2 \wedge \text{Limit}())))$$
$$\lfloor F_1 \rfloor \,;\, F_2 \equiv (F_1 \wedge \neg \text{ShorterNonEmptyTrace}(F_1)) \,;\, F_2$$
$$F_1 \,;\, \lfloor F_2 \rfloor \equiv F_1 \,;\, (F_2 \wedge \neg \text{ShorterNonEmptyTrace}(F_2))$$

Theorem 4. *For each of the formulæ $\lfloor F_1 \rfloor \cdot F_2$, $F_1 \cdot \lfloor F_2 \rfloor$, $\lfloor F_1 \rfloor \,;\, F_2$ and $F_1 \,;\, \lfloor F_2 \rfloor$ of* EAGLE$_{[]}$ *there exists a semantically equivalent formula in* EAGLE.

5   On-line Monitoring of Deterministic Cut Operators

In [BGHS04b], a calculus for EAGLE was presented that defines directly an on-line monitoring algorithm in which observation states are consumed on a step-by-step basis in tandem with a partial evaluation of the monitoring formula. Here, EAGLE's calculus is extended by rules that encode the semantics of the mixfix operators of EAGLE$_{[]}$. For the calculus of EAGLE$_{[]}$, we establish that the asymptotic space complexity of on-line monitoring for the variants with restrictions applied to the left operand is no worse than the asymptotic space complexity of the sub-formulæ. For the operators with restrictions applied to the right operand, we show that the space complexity coincides with the corresponding non-deterministic operators.

The extended calculus allows us an efficient evaluation, which can not be achieved by substituting appearances of mixfix operators by their semantically

equivalent Eagle-formulæ. For example, in the extended calculus the evaluation of $F_1 \,;\, \lfloor F_2 \rfloor$ takes $|\sigma|$ applications of $eval\langle\!\langle \ldots \rangle\!\rangle$, while the semantically equivalent Eagle-formula $F_1 \,;\, (F_2 \wedge \neg \text{ShorterNonEmptyTrace}(F_2))$ takes already $|\sigma|^2$ applications of $eval\langle\!\langle \ldots \rangle\!\rangle$ due to evaluation of the sequential composition operator in the formula, plus the evaluation steps for the rule ShorterNonEmptyTrace(F_2).

5.1 Eagle's On-Line Monitoring Algorithm

The evaluation calculus presented in [BGHS04b] used four functions. First, a formula is initialised using $init\langle\!\langle \ldots \rangle\!\rangle$, which substitutes rules by their rule bodies. Second, $eval\langle\!\langle \ldots \rangle\!\rangle$ evaluates the resulting formula in the current state, where $update\langle\!\langle \ldots \rangle\!\rangle$ takes care of $\odot$-operators so that a history of states does not need to be stored. Third, $value\langle\!\langle \ldots \rangle\!\rangle$ determines the truth value of the verification at the boundaries of the trace.

In the following, $\rho b.F(b)$ is a closed term which denotes a fixed-point, such that $\rho b.F(b) = F(\rho b.F(b))$, where b represents the recursion variable. Furthermore, named operators are introduced. The named operators are indeed functions of some type $\mathbf{Form} \times \ldots \times \mathbf{Form} \to \mathbf{Form}$ such that it is possible to rewrite a formula during evaluation.

Rules are assumed to have their parameters ordered by their type in the form $N(\mathbf{Form}\ F_1, \ldots, \mathbf{Form}\ F_m, primitive\ type\ x_1, \ldots, primitive\ type\ x_n) = F$. W.l.o.g. all definitions can be rewritten into this form by simply reordering the rule's arguments. The arguments are then written as two vectors $\vec{F}$ and $\vec{P}$ with types $\overrightarrow{\mathbf{Form}}$ and $\vec{T}$ respectively. Similar to the rewriting of $\bigcirc$, each rule N is rewritten as $\overline{N} : \mathbf{Form} \times \vec{T} \to \mathbf{Form}$ during initialisation, where the first argument denotes a recursive application of the rule body of N.

Definition 6. *A monitoring formula F holds on a trace $\sigma = s_1 s_2 \ldots s_{|\sigma|}$, iff the formula $value\langle\!\langle eval\langle\!\langle \ldots eval\langle\!\langle eval\langle\!\langle init\langle\!\langle F, \mathtt{null}, \mathtt{null}\rangle\!\rangle, s_1\rangle\!\rangle, s_2\rangle\!\rangle \ldots, s_{|\sigma|}\rangle\!\rangle\rangle\!\rangle$ evaluates to **True**, where $\mathtt{null}$ denotes a special element that is not equivalent to any other formula of Eagle. Instances of vector types $\overrightarrow{\mathbf{Form}}$ and $\vec{T}$ are denoted by $\langle F_1, \ldots, F_n \rangle$ and $\langle p, \ldots, r \rangle$ respectively. For both vector types, $\vec{\emptyset}$ denotes the empty vector. The rules for the temporal operators and temporal predicates are:*

$$init\langle\!\langle \bigcirc F, Z, b' \rangle\!\rangle = \underline{\text{Next}}(init\langle\!\langle F, Z, b' \rangle\!\rangle)$$

$$init\langle\!\langle \odot F, Z, b' \rangle\!\rangle = \underline{\text{Previous}}(\alpha, value\langle\!\langle \alpha \rangle\!\rangle), \ where\ \alpha = init\langle\!\langle F, Z, b' \rangle\!\rangle$$

$$init\langle\!\langle F_1 \circ F_2, Z, b' \rangle\!\rangle = init\langle\!\langle F_1, Z, b' \rangle\!\rangle \circ init\langle\!\langle F_2, Z, b' \rangle\!\rangle, \ where\ \circ \in \{\cdot, ; \}$$

$$init\langle\!\langle N(\vec{F}, \vec{P}), N(\vec{F}, \vec{P}'), b' \rangle\!\rangle = \overline{N}(b', \vec{P})$$

$$init\langle\!\langle N(\vec{F}, \vec{P}), Z, b' \rangle\!\rangle = \overline{N}(\rho b.init\langle\!\langle F[\hat{F}/\vec{F}], N(\vec{F}, \vec{P}), b\rangle\!\rangle, \vec{P}), \ where$$

$$\hat{F} = init\langle\!\langle \vec{F}, Z, b' \rangle\!\rangle \text{ and } Z \not\equiv \overline{N}(\vec{F}, \ldots)$$

$$value\langle\!\langle \underline{\mathrm{Next}}(F) \rangle\!\rangle = \begin{cases} F & \text{if at the beginning of the trace} \\ \mathbf{False} & \text{if at the end of the trace or } |\sigma| = 0 \end{cases}$$

$$value\langle\!\langle \underline{\mathrm{Previous}}(F, \hat{F}) \rangle\!\rangle = \begin{cases} \mathbf{False} & \text{if at the beg. of the trace or } |\sigma| = 0 \\ value\langle\!\langle F \rangle\!\rangle & \text{if at the end of the trace} \end{cases}$$

$$value\langle\!\langle F_1 \cdot F_2 \rangle\!\rangle = value\langle\!\langle F_1 \rangle\!\rangle \wedge value\langle\!\langle F_2 \rangle\!\rangle$$

$$value\langle\!\langle F_1 \,;\, F_2 \rangle\!\rangle = \mathbf{False}$$

$$value\langle\!\langle \overline{N}(\vec{F}, \vec{P}) \rangle\!\rangle = \begin{cases} \mathbf{True} & \text{if } (\mathbf{max}\ N(\ldots)) \in R, \\ \mathbf{False} & \text{otherwise} \end{cases}$$

$$eval\langle\!\langle \underline{\mathrm{Next}}(F), s \rangle\!\rangle = update\langle\!\langle F, s, \mathtt{null}, \mathtt{null} \rangle\!\rangle$$

$$eval\langle\!\langle \underline{\mathrm{Previous}}(F, \hat{F}), s \rangle\!\rangle = eval\langle\!\langle \hat{F}, s \rangle\!\rangle$$

$$eval\langle\!\langle F_1 \cdot F_2, s \rangle\!\rangle = \underline{\mathrm{if}}\ value\langle\!\langle F_1 \rangle\!\rangle = \mathbf{True}\ \underline{\mathrm{then}}\ (\alpha \cdot F_2) \vee eval\langle\!\langle F_2, s \rangle\!\rangle$$
$$\underline{\mathrm{else}}\ \alpha \cdot F_2,\ where\ \alpha = eval\langle\!\langle F_1, s \rangle\!\rangle$$

$$eval\langle\!\langle F_1 \,;\, F_2, s \rangle\!\rangle = \underline{\mathrm{if}}\ value\langle\!\langle \alpha \rangle\!\rangle = \mathbf{True}\ \underline{\mathrm{then}}\ (\alpha \,;\, F_2) \vee eval\langle\!\langle F_2, s \rangle\!\rangle$$
$$\underline{\mathrm{else}}\ \alpha \,;\, F_2,\ where\ \alpha = eval\langle\!\langle F_1, s \rangle\!\rangle$$

$$eval\langle\!\langle \overline{N}(\rho b'.F(b'), \vec{P}), s \rangle\!\rangle = eval\langle\!\langle F(\rho b'.F(b'))[eval\langle\!\langle \vec{P}, s \rangle\!\rangle / \vec{p}], s \rangle\!\rangle$$

$$update\langle\!\langle \underline{\mathrm{Next}}(F), s, Z, b' \rangle\!\rangle = \underline{\mathrm{Next}}(update\langle\!\langle F, s, Z, b' \rangle\!\rangle)$$

$$update\langle\!\langle \underline{\mathrm{Previous}}(F, \hat{F}), s, Z, b' \rangle\!\rangle = \underline{\mathrm{Previous}}(update\langle\!\langle F, s, Z, b' \rangle\!\rangle, eval\langle\!\langle F, s \rangle\!\rangle)$$

$$update\langle\!\langle F_1 \circ F_2, s, Z, b' \rangle\!\rangle = update\langle\!\langle F_1, s, Z, b' \rangle\!\rangle \circ F_2,\ where\ \circ \in \{\,;\,, \cdot\}$$

$$update\langle\!\langle \alpha, s, \alpha, b' \rangle\!\rangle = \overline{N}(b', \vec{P}),\ where\ \alpha \equiv \overline{N}(\rho b.F(b), \vec{P})$$

$$update\langle\!\langle \alpha, s, \hat{F}, Z \rangle\!\rangle = \overline{N}(\rho b'.update\langle\!\langle F(\rho b'.F(b')), s, \alpha, \vec{P} \rangle\!\rangle, \vec{P}),$$
$$where\ \alpha \equiv \overline{N}(\rho b.F(b), \vec{P})\ and\ Z \not\equiv \overline{N}(\vec{F}, \ldots)$$

The rules for propositional constants and operators are defined in the obvious way.

We provide a rather simple example that shows the evaluation of the temporal operators $\bigcirc$ and $\odot$. As example trace we have chosen a trace of length one, where in its only state the proposition p is true.
Evaluating $\bigcirc\odot p$ on $\langle\{p\}\rangle$:

1. $value\langle\!\langle eval\langle\!\langle init\langle\!\langle \bigcirc\odot p, \mathtt{null}, \mathtt{null} \rangle\!\rangle, s_1 \rangle\!\rangle \rangle\!\rangle$
2. $value\langle\!\langle eval\langle\!\langle \underline{\mathrm{Next}}(init\langle\!\langle \odot p, \mathtt{null}, \mathtt{null} \rangle\!\rangle), s_1 \rangle\!\rangle \rangle\!\rangle$

For the next step, $init\langle\!\langle \odot p, \mathtt{null}, \mathtt{null} \rangle\!\rangle$ is rewritten to $\underline{\mathrm{Previous}}(init\langle\!\langle p, \mathtt{null}, \mathtt{null} \rangle\!\rangle, value\langle\!\langle init\langle\!\langle p, \mathtt{null}, \mathtt{null} \rangle\!\rangle \rangle\!\rangle)$. The first parameter of $\underline{\mathrm{Previous}}(\ldots)$ stores the formula that would be evaluated after the $\odot$-operator. In the second parameter, the past is stored, which is referring to the left boundary of the trace now.

3. $value\langle\!\langle eval\langle\!\langle$
   $\underline{\mathrm{Next}}(\underline{\mathrm{Previous}}(init\langle\!\langle p, \mathtt{null}, \mathtt{null} \rangle\!\rangle, value\langle\!\langle init\langle\!\langle p, \mathtt{null}, \mathtt{null} \rangle\!\rangle \rangle\!\rangle)), s_1 \rangle\!\rangle \rangle\!\rangle$

4. $value\langle\!\langle eval\langle\!\langle \underline{\mathrm{Next}}(\underline{\mathrm{Previous}}(p, value\langle\!\langle p\rangle\!\rangle)), s_1\rangle\!\rangle\rangle\!\rangle$
5. $value\langle\!\langle eval\langle\!\langle \underline{\mathrm{Next}}(\underline{\mathrm{Previous}}(p, \mathbf{False})), s_1\rangle\!\rangle\rangle\!\rangle$
6. $value\langle\!\langle update\langle\!\langle \underline{\mathrm{Previous}}(p, \mathbf{False}), s_1, \mathtt{null}, \mathtt{null}\rangle\!\rangle\rangle\!\rangle$

When $\underline{\mathrm{Next}}(\ldots)$ is evaluated, it is rewritten to $update\langle\!\langle\ldots\rangle\!\rangle$. The last two parameters of $update\langle\!\langle\ldots\rangle\!\rangle$ are only used in conjunction with the evaluation of rules, so that they can be ignored in this example. $update\langle\!\langle\ldots\rangle\!\rangle$ rewrites the arguments of appearances of $\underline{\mathrm{Previous}}(\ldots)$, since due to the next operator, previous states occur now as being shifted one state to the end of the trace. Hence, the first parameter of $\underline{\mathrm{Previous}}(\ldots)$, which was used to store the initialised parameter of the $\odot$-operator, is evaluated in this state and the result of this evaluation is placed in the second parameter.

7. $value\langle\!\langle \underline{\mathrm{Previous}}(update\langle\!\langle p, s_1, \mathtt{null}, \mathtt{null}\rangle\!\rangle, eval\langle\!\langle p, s_1\rangle\!\rangle)\rangle\!\rangle$

$value\langle\!\langle\ldots\rangle\!\rangle$ is now referring to the right boundary of the trace. Hence, the second parameter of $\underline{\mathrm{Previous}}(\ldots)$, which is storing the result of the last state, determines the outcome of $value\langle\!\langle\ldots\rangle\!\rangle$.

8. $value\langle\!\langle \underline{\mathrm{Previous}}(p, \mathbf{True})\rangle\!\rangle$
9. $value\langle\!\langle\mathbf{True}\rangle\!\rangle$
10. $\mathbf{True}$

5.2 EAGLE's Monitoring Algorithm Extended

The following ternary rules $\underline{\mathrm{LMxConcat}}(\ldots)$, $\underline{\mathrm{RMnConcat}}(\ldots)$, $\underline{\mathrm{RMxConcat}}(\ldots)$, $\underline{\mathrm{LMxSeqComp}}(\ldots)$, and $\underline{\mathrm{RMnSeqComp}}(\ldots)$, $\underline{\mathrm{RMxSeqComp}}(\ldots)$, with self-explanatory correspondences to their respective mixfix operators, are used during evaluation to sort out the shortest/longest sub-traces by updating the third argument depending on whether the first two arguments allow a cut to be made or not.

Definition 7. (Extension of Definition 6) EAGLE*'s calculus is extended by mixfix variants of sequential composition, such that*

$$init\langle\!\langle \lfloor F_1 \rfloor \circ F_2, Z, b'\rangle\!\rangle = \lfloor init\langle\!\langle F_1, Z, b'\rangle\!\rangle\rfloor \circ init\langle\!\langle F_2, Z, b'\rangle\!\rangle$$

$$init\langle\!\langle \lceil F_1 \rceil \circ F_2, Z, b'\rangle\!\rangle = \underline{\varphi}(init\langle\!\langle F_1, Z, b'\rangle\!\rangle, init\langle\!\langle F_2, Z, b'\rangle\!\rangle, \mathtt{null})$$

$$init\langle\!\langle F_1 \circ \lfloor F_2 \rfloor, Z, b'\rangle\!\rangle = \underline{\varphi}(init\langle\!\langle F_1, Z, b'\rangle\!\rangle, init\langle\!\langle F_2, Z, b'\rangle\!\rangle, \mathtt{null})$$

$$init\langle\!\langle F_1 \circ \lceil F_2 \rceil, Z, b'\rangle\!\rangle = \underline{\varphi}(init\langle\!\langle F_1, Z, b'\rangle\!\rangle, init\langle\!\langle F_2, Z, b'\rangle\!\rangle, \underline{\mathrm{List}}(\mathbf{False}, \mathbf{False}, \mathtt{null}))$$

$$value\langle\!\langle\mathtt{null}\rangle\!\rangle = \mathbf{False}$$

$$value\langle\!\langle\underline{\mathrm{List}}(F_1, F_2, F_3)\rangle\!\rangle = \underline{\mathrm{if}}\ value\langle\!\langle F_2\rangle\!\rangle = \mathbf{True}\ \underline{\mathrm{then}}\ F_1$$
$$\underline{\mathrm{else}}\ value\langle\!\langle F_3\rangle\!\rangle$$

$$value\langle\!\langle\lfloor F_1\rfloor \cdot F_2\rangle\!\rangle = value\langle\!\langle F_1\rangle\!\rangle \wedge value\langle\!\langle F_2\rangle\!\rangle$$

$$value\langle\!\langle\underline{\mathrm{LMxConcat}}(F_1, F_2, F_3)\rangle\!\rangle = \underline{\mathrm{if}}\ value\langle\!\langle F_1\rangle\!\rangle = \mathbf{True}\ \underline{\mathrm{then}}\ value\langle\!\langle F_2\rangle\!\rangle$$
$$\underline{\mathrm{else}}\ value\langle\!\langle F_3\rangle\!\rangle$$

$$value\langle\!\langle\underline{\text{RMnConcat}}(F_1, F_2, F_3)\rangle\!\rangle = \underline{\text{if}}\ value\langle\!\langle F_2\rangle\!\rangle = \textbf{True}\ \underline{\text{then}}\ value\langle\!\langle F_1\rangle\!\rangle$$
$$\underline{\text{else}}\ value\langle\!\langle F_3\rangle\!\rangle$$
$$value\langle\!\langle\underline{\text{RMxConcat}}(F_1, F_2, F_3)\rangle\!\rangle = value\langle\!\langle\text{Append}(F_3, \underline{\text{List}}(value\langle\!\langle F_1\rangle\!\rangle, F_2, \texttt{null}))\rangle\!\rangle$$
$$value\langle\!\langle\lfloor F_1\rfloor\ ;\ F_2\rangle\!\rangle = \textbf{False}$$
$$value\langle\!\langle\underline{\text{LMxSeqComp}}(F_1, F_2, F_3)\rangle\!\rangle = value\langle\!\langle\underline{\text{RMnSeqComp}}(F_1, F_2, F_3)\rangle\!\rangle =$$
$$value\langle\!\langle\underline{\text{RMxSeqComp}}(F_1, F_2, F_3)\rangle\!\rangle = value\langle\!\langle F_3\rangle\!\rangle$$

$$eval\langle\!\langle\texttt{null}, s\rangle\!\rangle = \texttt{null}$$
$$eval\langle\!\langle\underline{\text{List}}(F_1, F_2, F_3), s\rangle\!\rangle = \underline{\text{List}}(F_1, \beta, \gamma)$$
$$eval\langle\!\langle\lfloor F_1\rfloor \cdot F_2, s\rangle\!\rangle = \underline{\text{if}}\ value\langle\!\langle F_1\rangle\!\rangle = \textbf{True}\ \underline{\text{then}}\ \beta$$
$$\underline{\text{else}}\ \lfloor\alpha\rfloor \cdot F_2$$
$$eval\langle\!\langle\underline{\text{LMxConcat}}(F_1, F_2, F_3), s\rangle\!\rangle = \underline{\text{if}}\ value\langle\!\langle F_1\rangle\!\rangle = \textbf{True}\ \underline{\text{then}}$$
$$\underline{\text{LMxConcat}}(\alpha, F_2, \beta)$$
$$\underline{\text{else}}\ \underline{\text{LMxConcat}}(\alpha, F_2, \gamma)$$
$$eval\langle\!\langle\underline{\text{RMnConcat}}(F_1, F_2, F_3), s\rangle\!\rangle =$$
$$\underline{\text{RMnConcat}}(\alpha, F_2, eval\langle\!\langle\underline{\text{List}}(value\langle\!\langle F_1\rangle\!\rangle, F_2, F_3), s\rangle\!\rangle)$$
$$eval\langle\!\langle\underline{\text{RMxConcat}}(F_1, F_2, F_3), s\rangle\!\rangle =$$
$$\underline{\text{RMxConcat}}(\alpha, F_2, eval\langle\!\langle\text{Append}(F_3, \underline{\text{List}}(value\langle\!\langle F_1\rangle\!\rangle, F_2, \texttt{null})), s\rangle\!\rangle)$$
$$eval\langle\!\langle\lfloor F_1\rfloor\ ;\ F_2, s\rangle\!\rangle = \underline{\text{if}}\ value\langle\!\langle\alpha\rangle\!\rangle = \textbf{True}\ \underline{\text{then}}\ \beta$$
$$\underline{\text{else}}\ \lfloor\alpha\rfloor\ ;\ F_2$$
$$eval\langle\!\langle\underline{\text{LMxSeqComp}}(F_1, F_2, F_3), s\rangle\!\rangle = \underline{\text{if}}\ value\langle\!\langle\alpha\rangle\!\rangle = \textbf{True}\ \underline{\text{then}}$$
$$\underline{\text{LMxSeqComp}}(\alpha, F_2, \beta)$$
$$\underline{\text{else}}\ \underline{\text{LMxSeqComp}}(\alpha, F_2, \gamma)$$
$$eval\langle\!\langle\underline{\text{RMnSeqComp}}(F_1, F_2, F_3), s\rangle\!\rangle =$$
$$\underline{\text{RMnSeqComp}}(\alpha, F_2, eval\langle\!\langle\underline{\text{List}}(value\langle\!\langle\alpha\rangle\!\rangle, F_2, F_3), s\rangle\!\rangle)$$
$$eval\langle\!\langle\underline{\text{RMxSeqComp}}(F_1, F_2, F_3), s\rangle\!\rangle =$$
$$\underline{\text{RMxSeqComp}}(\alpha, F_2, eval\langle\!\langle\text{Append}(F_3, \underline{\text{List}}(value\langle\!\langle\alpha\rangle\!\rangle, F_2, \texttt{null})), s\rangle\!\rangle)$$
$$update\langle\!\langle\lfloor F_1\rfloor \circ F_2, s, Z, b'\rangle\!\rangle = \lfloor update\langle\!\langle F_1, s, Z, b'\rangle\!\rangle\rfloor \circ F_2$$
$$update\langle\!\langle\underline{\varphi}(F_1, F_2, F_3), s, Z, b'\rangle\!\rangle =$$
$$\underline{\varphi}(update\langle\!\langle F_1, s, Z, b'\rangle\!\rangle, F_2, F_3)$$

where $\alpha \equiv eval\langle\!\langle F_1, s\rangle\!\rangle$, $\beta \equiv eval\langle\!\langle F_2, s\rangle\!\rangle$, $\gamma \equiv eval\langle\!\langle F_3, s\rangle\!\rangle$, $\circ \in \{\ ;\ , \cdot\}$, and φ denotes the rule $\underline{\text{LMxSeqComp}}$, $\underline{\text{RMnSeqComp}}$, $\underline{\text{RMxSeqComp}}$, $\underline{\text{LMxConcat}}$, $\underline{\text{RMnConcat}}$, $\underline{\text{RMxConcat}}$, which is apparent from the context.

Theorem 5. *The semantics of* Eagle$_{[]}$ *'s calculus (Definition 7) coincide with the semantics of the corresponding logic (Definition 5).*

5.3 On-line Monitoring Complexity

We now consider the time and space requirements of the evaluation of concatenation, sequential composition and the mixfix operators. In [BGHS04a], we

showed that the time and space complexity of the future LTL fragment of EA-
GLE is independent of the length of the monitoring trace. We first show below
that the evaluation of a non-deterministic cut operator $F_1 \circ F_2$, $\circ \in \{\,;\,,\cdot\,\}$, whose
operands are free of cut formula may require $O(|\sigma|^2)$ number of calls to evalu-
ate F_2 (Theorem 6). However, for the mixfix operators with restrictions on the
left, the complexity is reduced to being independent of the trace length again
(Theorem 7).

Consider an arbitrary formula $F_1 \cdot F_2$. The state evaluation rule for concate-
nation is given by

$$eval\langle\!\langle F_1 \cdot F_2, s\rangle\!\rangle = \underline{\text{if}}\ value\langle\!\langle F_1\rangle\!\rangle = \textbf{True}\ \underline{\text{then}}\ (eval\langle\!\langle F_1, s\rangle\!\rangle \cdot F_2) \vee eval\langle\!\langle F_2, s\rangle\!\rangle$$
$$\underline{\text{else}}\ eval\langle\!\langle F_1, s\rangle\!\rangle \cdot F_2$$

In the worst case scenario of evaluating concatenation, a non-deterministic cut
is made at each state of a trace. This can be enforced by the formula $\textbf{True} \cdot F_2$.
On an arbitrary trace σ, $\textbf{True} \cdot F_2$ is evaluated as

$$value\langle\!\langle eval\langle\!\langle \ldots eval\langle\!\langle eval\langle\!\langle init\langle\!\langle \textbf{True} \cdot F_2, \texttt{null}, \texttt{null}\rangle\!\rangle, s_1\rangle\!\rangle, s_2\rangle\!\rangle \ldots, s_{|\sigma|}\rangle\!\rangle\rangle\!\rangle$$

By straightforward applications of rules of $\text{EAGLE}_{[]}$'s calculus, the evaluation
can be unfolded as

$$value\langle\!\langle \bigvee_{n=1}^{|\sigma|} eval\langle\!\langle \ldots eval\langle\!\langle eval\langle\!\langle init\langle\!\langle F_2, \texttt{null}, \texttt{null}\rangle\!\rangle, s_n\rangle\!\rangle, s_{n+1}\rangle\!\rangle \ldots, s_{|\sigma|}\rangle\!\rangle\rangle\!\rangle$$

Relative to the evaluation of F_2, the formula requires $(|\sigma|^2 + |\sigma|)/2$ appli-
cations of $eval\langle\!\langle \ldots\rangle\!\rangle$. This argumentation can be carried forward to sequential
composition as well, and additionally, to all mixfix operators with restrictions
on the right operand.

Theorem 6. *For a given trace σ, the operators $F_1 \cdot F_2$, $F_1\,;\,F_2$, $F_1 \cdot \lfloor F_2 \rfloor$, $F_1 \cdot$
$\lceil F_2 \rceil$, $F_1\,;\,\lfloor F_2 \rfloor$ and $F_1\,;\,\lceil F_2 \rceil$ require up to $O(|\sigma|^2)$ applications of $eval\langle\!\langle \ldots\rangle\!\rangle$ in
addition to the applications required to evaluate F_1 and F_2.*

When we consider a mixfix formula with deterministic restrictions on the left
operand, e.g. $\lfloor F_1 \rfloor \cdot F_2$, then we can show that we only need linear-space for its
evaluation – relative to the space required to evaluate the operands. By taking
the state-evaluation rule for $\lfloor F_1 \rfloor \cdot F_2$, i.e.

$$eval\langle\!\langle \lfloor F_1 \rfloor \cdot F_2, s\rangle\!\rangle = \underline{\text{if}}\ value\langle\!\langle F_1\rangle\!\rangle = \textbf{True}\ \underline{\text{then}}\ eval\langle\!\langle F_2, s\rangle\!\rangle$$
$$\underline{\text{else}}\ \lfloor eval\langle\!\langle F_1, s\rangle\!\rangle \rfloor \cdot F_2$$

one can immediately see that the non-deterministic choice of concatenation (i.e.
$(eval\langle\!\langle F_1, s\rangle\!\rangle \cdot F_2) \vee eval\langle\!\langle F_2, s\rangle\!\rangle$) is replaced by a single application of $eval\langle\!\langle \ldots\rangle\!\rangle$.
We can carry this forward to all mixfix operators with restrictions on the left
operand, so that we obtain the following result:

Theorem 7. *For a given trace σ, the left mixfix operators $\lfloor F_1 \rfloor \cdot F_2$, $\lceil F_1 \rceil \cdot F_2$,
$\lfloor F_1 \rfloor\,;\,F_2$ and $\lceil F_1 \rceil\,;\,F_2$ require only $O(|\sigma|)$ applications of $eval\langle\!\langle \ldots\rangle\!\rangle$ in addition
to the applications required to evaluate F_1 and F_2.*

6   Conclusion

For the runtime verification logic EAGLE, we have shown that concatenation and sequential composition are equally expressive. Furthermore, mixfix operators were introduced, which limit the possible number of cuts of sequential composition and concatenation. The new operators restrict the lengths of the sequential composition/concatenation sub-traces, such that the corresponding sub-formula is satisfied on a sub-trace of minimal or maximal length. Since the cut is then uniquely defined on the trace, the mixfix variants of sequential composition and concatenation are deterministic counterparts of their corresponding non-mixfix operators. We further showed that the semantics of the mixfix operators are already definable in unextended EAGLE. For all mixfix operators, semantically equivalent mixfix operator free formulæ were presented. We then extended EAGLE's on-line monitoring calculus with rules for the new operators, where we could show that right-hand side restricted mixfix operators evaluate as efficiently as their non-deterministic counterparts, and left-hand side restricted mixfix operators can be evaluated more efficiently.

References

[BGHS04a] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Program monitoring with LTL in EAGLE. In *18th International Parallel and Distributed Processing Symposium, IPDPS 2004*. IEEE Computer Society, 2004.

[BGHS04b] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In *Verification, Model Checking, and Abstract Interpretation: 5th International Conference, VMCAI 2004*, volume 2937 of *LNCS*, pages 44–57. Springer-Verlag, 2004.

[CHMP81] A. Chandra, J. Halpern, A. Meyer, and R. Parikh. Equations between regular terms and an application to process logic. In *STOC '81: Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 384–390. ACM Press, 1981.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Min61] M.L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *The Annals of Mathematics*, 74(3):437–455, 1961.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.