Forkable Regular Expressions

Martin Sulzmann[1] and Peter Thiemann[2]

[1] Faculty of Computer Science and Business Information Systems
Karlsruhe University of Applied Sciences
Moltkestrasse 30, 76133 Karlsruhe, Germany
martin.sulzmann@hs-karlsruhe.de
[2] Faculty of Engineering, University of Freiburg
Georges-Köhler-Allee 079, 79110 Freiburg, Germany
thiemann@acm.org

Abstract. We consider *forkable regular expressions*, which enrich regular expressions with a fork operator, to establish a formal basis for static and dynamic analysis of the communication behavior of concurrent programs. We define a novel compositional semantics for forkable expressions, establish their fundamental properties, and define derivatives for them as a basis for the generation of automata, for matching, and for language containment tests.
Forkable expressions may give rise to non-regular languages, in general, but we identify sufficient conditions on expressions that guarantee finiteness of the automata construction via derivatives.

Keywords: automata and logic, forkable expressions, derivatives

1 Introduction

Languages like Concurrent ML and Go come with built-in support for fine-grained concurrency, dynamic thread creation, and channel-based communication. Analyzing the communication behavior of programs in these languages may be done by an *effect system*. Such a system computes an abstraction of the sequences of events (i.e., the communication traces with events like communication actions or synchronizations) that a program may exhibit.

Effect systems for concurrent programs have been explored by the Nielsons [12], who proposed to model event traces with "behaviors" which are regular expressions extended with a fork operator that encapsulates the behavior of a newly created thread. While their work enables the analysis of finiteness properties of the communication topology, it stops short of providing a semantics of behaviors in terms of effect traces. Subsequent work by the same authors [1, 2, 11] concentrates on subtyping and automatic inference of effects.

We take up the Nielsons' notion of behavior and tackle the problem of defining a compositional semantics for behaviors in terms of effect traces. Our novel definition yields a semantic basis for the static and dynamic analysis of concurrent languages with dynamic thread creation and other communication

effects. We show that, in general, a behavior may give rise to a *non-regular* trace language. This observation is in line with previous work on concurrent regular expressions [4], flow expressions [13], and shuffle expressions [6] all of which augment regular expressions with (at least) shuffle and shuffle closure operators. The shuffle closure is also referred to as the iterated shuffle [7].

We explore two application areas for forkable expressions. In a run-time verification setting (aka dynamic analysis), we are interested in matching traces against behaviors, either at run time or post-mortem. To this end, we extend Brzozowski derivatives to forkable expressions. Although Brzozowski's construction no longer gives rise to a finite automaton, in general, derivatives can still be used to solve instances of the word problem (which is hence decidable).

For static analysis, we are interested in approximation and testing language containment in a specification. For this use case, we give a decidable criterium that guarantees finiteness of (our extension of) Brzozowski's automaton construction. This criterium essentially requires a finite communication topology, that is, it forbids that new communicating threads are created in loops. We conjecture that this property can be established with the Nielsons' original analysis [12].

In summary our contributions are:

- In Section 3, we define a novel trace semantics of behaviors (i.e., regular expressions with fork), and establish their fundamental properties.
- Section 4 extends Brzozowski's derivative operation to behaviors.
- In Section 5, we characterize a class of behaviors with regular trace languages. For these behaviors, Brzozowski's construction yields finite automata.

Related work is discussed in Section 6.

The online version of this paper contains an appendix with all proofs. [3]

2 Preliminaries

For a set X, we write $\sharp X$ for the cardinality of X and $\wp(X)$ for its powerset. If $F : \wp(X) \to \wp(X)$ is a monotone function (i.e., $X \subseteq Y$ implies $F(X) \subseteq F(Y)$), then we write μF for the least fixpoint of this function, which is uniquely defined due to Tarski's theorem. We write $\mu X.e$ for the fixpoint $\mu(\lambda X.e)$ where e is a set-valued expression composed of monotone functions assuming that X is a set with the same type of elements. (The scope of the μ operator extends as far to the right as possible.) We will employ this operator to define the meaning of forkable Kleene star behaviors [10].

Let Σ be a finite set, the *alphabet of primitive events*. We write Σ^* for the set of finite words over Σ and denote with $v \cdot w$ the concatenation of words $v, w \in \Sigma^*$. For languages $L, M \subseteq \wp(\Sigma^*)$, we write $L \cdot M = \{v \cdot w \mid v \in L, w \in M\}$ for the set of all pairwise concatenations. We write $v \cdot M$ as a shorthand for $\{v\} \cdot M$. The (asynchronous) shuffle operation $v \| w \subseteq \Sigma^*$ on words is the set of all interleavings of words v and w. It is defined inductively by

$$\varepsilon \| w = \{w\} \qquad v \| \varepsilon = \{v\} \qquad xv \| yw = \{x\} \cdot (v \| yw) \cup \{y\} \cdot (xv \| w)$$

[3] http://arxiv.org/abs/1510.07293

$$L(r) = L(r, \{\varepsilon\}) \qquad \begin{array}{l} L(\phi, K) = \emptyset \\ L(\varepsilon, K) = K \\ L(x, K) = \{x\} \cdot K \end{array} \qquad \begin{array}{r} L(r + s, K) = L(r, K) \cup L(s, K) \\ L(r \cdot s, K) = L(r, L(s, K)) \\ L(r^*, K) = \mu X.L(r, X) \cup K \\ L(\mathit{Fork}(r), K) = L(r) \| K \end{array}$$

Fig. 1. Trace language of a behavior

The shuffle operation is lifted to languages by $L \| M = \bigcup \{v \| w \mid v \in L, w \in M\}$.

We write $L_1 \backslash L_2$ to denote the left quotient of L_2 with L_1 where $L_1 \backslash L_2 = \{w \mid \exists v \in L_1 . v \cdot w \in L_2\}$. We write $x \backslash L$ as a shorthand for $\{x\} \backslash L$.

3 Behaviors

Recall that Σ is the alphabet of primitive events. Intuitively, a primitive event $x \in \Sigma$ is a globally visible side effect like sending or receiving a message. A *behavior* is a regular expression over Σ extended with a new fork operator.

$$r, s, t ::= \phi \mid \varepsilon \mid x \mid r + s \mid r \cdot s \mid r^* \mid \mathit{Fork}(r) \mid (r)$$

As usual, we assume that $\cdot$ binds tighter than $+$.

The semantics of a behavior r is going to be a *trace language* $L(r) \subseteq \Sigma^*$. However, due to the presence of the fork operator, its definition is not a simple extension of the standard semantics $[\![\cdot]\!]$ of a regular expression.

Definition 1. *Figure 1 defines, for a behavior r, the trace languages $L(r) \subseteq \Sigma^*$ and $L(r, K) \subseteq \Sigma^*$ with respect to a continuation language $K \subseteq \Sigma^*$.*

By induction on r, we can show that the mapping $K \mapsto L(r, K)$ in $\wp(\Sigma^*) \to \wp(\Sigma^*)$ is monotone, so that L is well-defined. For fork-free behaviors that do not make use of the $\mathit{Fork}(r)$ operator, the trace language is regular and coincides with the standard semantics $[\![r]\!]$ of a regular expression.

Theorem 2. *If r is fork-free, then $L(r)$ is regular and $L(r) = [\![r]\!]$.*

It is known that the regular languages are closed under the shuffle operation [5]. However, for forkable expressions the semantics of $\mathit{Fork}(r)$ is defined by *shuffling with the continuation language* so that the language defined by a behavior need not be regular as the following example shows.

Example 3. Consider the behavior $\mathit{Fork}(s)^*$ for a plain regular expression s where s only consists of the standard regular expression operators. Its semantics is the shuffle closure of $[\![s]\!]$ as demonstrated by the following calculation

$$\begin{aligned} L(\mathit{Fork}(s)^*, \{\varepsilon\}) &= \mu X.L(\mathit{Fork}(s), X) \cup \{\varepsilon\} = \mu X.L(s) \| X \cup \{\varepsilon\} \\ &= \{\varepsilon\} \cup L(s) \cup L(s) \| L(s) \cup \ldots \end{aligned} \tag{1}$$

where we assume that $\|$ binds tighter than $\cup$.

$$
\begin{array}{llll}
\mathcal{C}(\phi) & = \phi & \mathcal{S}(\phi) & = \phi \\
\mathcal{C}(\varepsilon) & = \varepsilon & \mathcal{S}(\varepsilon) & = \phi \\
\mathcal{C}(x) & = \phi & \mathcal{S}(x) & = x \\
\mathcal{C}(r+s) & = \mathcal{C}(r) + \mathcal{C}(s) & \mathcal{S}(r+s) & = \mathcal{S}(r) + \mathcal{S}(s) \\
\mathcal{C}(r \cdot s) & = \mathcal{C}(r) \cdot \mathcal{C}(s) & \mathcal{S}(r \cdot s) & = \mathcal{S}(r) \cdot s + \mathcal{C}(r) \cdot \mathcal{S}(s) \\
\mathcal{C}(r^*) & = \mathcal{C}(r)^* & \mathcal{S}(r^*) & = \mathcal{C}(r)^* \cdot \mathcal{S}(r) \cdot r^* \\
\mathcal{C}(\mathit{Fork}(r)) & = \mathit{Fork}(r) & \mathcal{S}(\mathit{Fork}(r)) & = \phi
\end{array}
$$

Fig. 2. Concurrent and sequential part of a behavior

In general, the shuffle closure is not regular [7] as the following concrete instance shows. Consider the behavior $r = \mathit{Fork}(x \cdot y + y \cdot x)^*$. By the calculation in (1), $L(r)$ is the shuffle closure of $\{x \cdot y, y \cdot x\}$ which happens to be the context-free language $\{w \in \{x, y\}^* \mid \sharp(x, w) = \sharp(y, w)\}$ of words that contain the same number of xs and ys. This language is not regular.

Some of our proofs rely on semantic equivalence and employ identities from Kleene algebra [9] that hold for standard regular expressions. Hence, we need to establish that forkable expressions also form a Kleene algebra.

Definition 4 (Semantic equality and containment).

1. *Behaviors r and s are equal, $r \equiv s$, if $L(r, K) = L(s, K)$, for all K.*
2. *Behaviors r and s are contained, $r \leqq s$, if $L(r, K) \subseteq L(s, K)$, for all K.*

Theorem 5. *The set of forkable expressions with semantic equality and containment is a Kleene algebra.*

Each behavior r can be decomposed into a sequential part $\mathcal{S}(r)$ and a concurrent part $\mathcal{C}(r)$, which are defined by induction on r in Figure 2. The intuition is that the sequential part of a behavior describes what must happen next, inevitably, whereas the concurrent part describes behavior that happens eventually and concurrent to the sequential behavior. For example, in case of concatenation $r \cdot s$, the sequential part must either start with $\mathcal{S}(r)$, or must end with $\mathcal{S}(s)$. For Kleene star r^* it is similar, we simply consider the possible unrolling of the underlying expression r.

Our decomposition theorem proves that every behavior is semantically equivalent to the union of its concurrent part and its sequential part. Its proof requires the Kleene identity $r^* \equiv \varepsilon + r \cdot r^*$.[4]

Theorem 6. *For all r, $r \equiv \mathcal{C}(r) + \mathcal{S}(r)$.*

The next lemma establishes some algebraic properties of the functions $\mathcal{C}()$ and $\mathcal{S}()$ that we need in subsequent proofs.

Lemma 7. *For all r: 1. $\mathcal{C}(\mathcal{C}(r)) = \mathcal{C}(r)$ (syntactic equality); 2. $\mathcal{C}(\mathcal{S}(r)) \equiv \phi$; 3. $\mathcal{S}(\mathcal{C}(r)) \equiv \phi$; 4. $\mathcal{S}(\mathcal{S}(r)) \equiv \mathcal{S}(r)$.*

[4] We generally write $r = s$ for *syntactic equality* of expressions and use other symbols like $r \equiv s$ for equivalences where some additional reasoning may be involved.

Proof. The proof for part 1 is by trivial induction on r. See the online version for the remaining parts; they are not needed in the rest of this paper. □

Lemma 8. *For all r, $\varepsilon \in L(r)$ iff $\varepsilon \leqq \mathcal{C}(r)$.*

4 Derivatives

We want to use Brzozowski's derivative operation [3] to translate behaviors to automata and to create algorithms for checking language containment and matching. To this end, we extend derivatives to forkable expressions. The derivative of r w.r.t. some symbol x, written $d_x(r)$, yields the new behavior after consumption of the leading symbol x. The derivative operation for behaviors is defined by structural induction. In addition to the regular operators, the derivative needs to deal with $\mathit{Fork}(r)$ expressions and the case of concatenated expressions $r \cdot s$ requires special attention.

Definition 9 (Derivatives). *The derivative of behavior r w.r.t. some symbol x is defined inductively as follows:*

$$
\begin{array}{ll}
d_x(\phi) = \phi & d_x(r+s) = d_x(r) + d_x(s) \\
d_x(\varepsilon) = \phi & d_x(r \cdot s) = d_x(r) \cdot s + \mathcal{C}(r) \cdot d_x(s) \\
d_x(y) = \begin{cases} \varepsilon \text{ if } x = y \\ \phi \text{ otherwise} \end{cases} & d_x(r^*) = d_x(r) \cdot r^* \\
 & d_x(\mathit{Fork}(r)) = \mathit{Fork}(d_x(r))
\end{array}
$$

We just explain the cases that differ from Brzozowski's definition. The derivative of a fork, $\mathit{Fork}(r)$, is simply pushed down to the underlying expression. The derivative of $r \cdot s$ consists of two components. The first one, $d_x(r) \cdot s$, is identical to the standard definition: it computes the derivative of r and continues with s. The second one covers symbols that may reach s. In a fork-free regular expression, a symbol in s can only be consumed if r is nullable, i.e. $\varepsilon \in L(r)$. For forkable behaviors, a symbol in s can also be consumed if r exhibits concurrent behavior. Hence, we extract the concurrent behavior $\mathcal{C}(r)$ and concatenate it with the derivative of s. The concurrent behavior generalizes nullability in the sense that $\varepsilon \in \mathcal{C}(r)$ iff $\varepsilon \in L(r)$. See Lemma 8.

Next, we verify that the derivative operation is correct in the sense that the resulting expression $d_x(r)$ denotes the left quotient of r by x.

Theorem 10 (Left Quotients). *Let r be a behavior and x be a symbol. Then, we have that $L(d_x(r)) = x\backslash L(r)$.*

Proof. To obtain a viable inductive hypothesis for the proof, we need to expand the definition of $L(r) = L(r, \{\varepsilon\})$ and to generalize the statement to an arbitrary continuation language $K \subseteq \Sigma^*$. That is, we set out to prove, by induction on r:

$$\forall r.\ \forall K.\ L(d_x(r), K) \cup L(\mathcal{C}(r)) \| (x\backslash K) = x\backslash L(r, K) \tag{2}$$

The original statement follows from the generalized hypothesis (2) by setting $K = \{\varepsilon\}$ (recall that $L\|\emptyset = \emptyset$):

$$\begin{aligned} L(d_x(r)) &= L(d_x(r)) \cup L(\mathcal{C}(r))\|\emptyset \\ &= L(d_x(r), \{\varepsilon\}) \cup L(\mathcal{C}(r))\|(x\backslash\{\varepsilon\}) \\ &\overset{(2)}{=} x\backslash L(r, \{\varepsilon\}) = x\backslash L(r) \end{aligned}$$

The proof of (2) proceeds by induction on r. □

Like in the standard regular expression case, we can conclude (based on the above result) that each behavior can be represented as a sum of its derivatives.

Theorem 11 (Representation). *For any behavior r, we have $L(r) = (\varepsilon \in L(r) \implies \{\varepsilon\}) \cup \bigcup_{x\in\Sigma} x \cdot L(d_x(r))$.*

Expression $(\varepsilon \in L(r) \implies \{\varepsilon\})$ denotes $\{\}$ if $\varepsilon \in L(r)$, otherwise, $\{\}$.

The representation theorem is the basis for solving the word problem with derivatives. Here, we extend the derivative operation to words as usual by $d_\varepsilon(r) = r$ and $d_{aw}(r) = d_w(d_a(r))$.

Corollary 12. *For a behavior r and $w \in \Sigma^*$, $w \in L(r)$ iff $\varepsilon \in L(d_w(r))$.*

This corollary implies decidability of the word problem for forkable expressions: the derivative is computable and the nullability test $\varepsilon \in L(d_w(r))$ is a syntactic test as for standard regular expressions. Full details how to compute all dissimilar derivatives can be found in the online version.

To construct an automaton from an expression r, Brzozowski repeatedly takes the derivative with respect to all symbols $x \in \Sigma$. We call these derivatives *descendants*.

Definition 13 (Descendants). *A* descendant *s of a behavior r is either r itself, a derivative of r, or the derivative of a descendant. We write $s \sqsubset r$, if s is* a direct descendant *of r, that is, if $s = d_x(r)$, for some x. The "is descendant of" relation is the reflexive, transitive closure of the direct descendant relation: $s \preceq r = s \sqsubset^* r$. The "is a true descendant of" relation is the transitive closure of the direct descendant relation: $s \prec r = s \sqsubset^+ r$. We define $d(r) = \{s \mid s \preceq r\}$ as the set of descendants of r.*

For standard regular expressions, Brzozowski showed that the set of descendants of an expression is finite up to *similarity*. Two expressions are similar if they are equal modulo associativity, commutativity, and idempotence. This result no longer holds in our setting.

In the following, we write $r \xrightarrow{x} s$ if $s = d_x(r)$. Subterms on which the derivation operation is applied are underlined.

$$\text{(Refl, Trans, Sym, Comp)}\quad r \simeq r \qquad \frac{r \simeq s \quad s \simeq t}{r \simeq t} \qquad \frac{s \simeq t}{t \simeq s} \qquad \frac{s \simeq t}{E[s] \simeq E[t]}$$

$$\text{(Assoc, Comm)}\qquad r + (s + t) \simeq (r + s) + t \qquad r + s \simeq s + r$$

$$\text{(Idem, Unit)}\qquad r + r \simeq r \qquad r + \phi \simeq r \qquad \phi + r \simeq r$$

$$\text{(Empty Word)}\qquad \varepsilon \cdot r \simeq r \qquad r \cdot \varepsilon \simeq r \qquad \varepsilon^* \simeq \varepsilon \qquad \mathit{Fork}(\varepsilon) \simeq \varepsilon$$

$$\text{(Empty Language)}\qquad \phi \cdot r \simeq \phi \qquad r \cdot \phi \simeq \phi \qquad \phi^* \simeq \varepsilon \qquad \mathit{Fork}(\phi) \simeq \phi$$

$$\text{(Regular Contexts)}\qquad E ::= [] \mid E^* \mid E \cdot s \mid r \cdot E \mid E + s \mid r + E \mid \mathit{Fork}(E)$$

Fig. 3. Rules and axioms for similarity

Example 14. Let $r = (\mathit{Fork}(x \cdot y))^*$ and take the derivative by x repeatedly.

$$\begin{aligned}
&(\mathit{Fork}(x \cdot y))^* \\
\xrightarrow{x}\ &\underline{\mathit{Fork}(y)} \cdot r \\
\xrightarrow{x}\ &\mathit{Fork}(\phi) \cdot r + \underline{\mathit{Fork}(y)} \cdot \underline{\mathit{Fork}(y)} \cdot r \\
\xrightarrow{x}\ &\cdots + \mathit{Fork}(\phi) \cdot \mathit{Fork}(y) \cdot r + \mathit{Fork}(y) \cdot (\mathit{Fork}(\phi) \cdot r + \mathit{Fork}(y) \cdot \mathit{Fork}(y) \cdot r) \\
\xrightarrow{x}\ &\ldots
\end{aligned}$$

Here we omit parentheses (assuming associativity) and apply equivalences such as $\mathcal{C}(\mathcal{C}(r)) = \mathcal{C}(r)$ (Lemma 7). Clearly, we obtain an increasing sequence of behaviors of the form $\mathit{Fork}(y) \cdot \ldots \cdot \mathit{Fork}(y) \cdot r$. Hence, the set of descendants of r is infinite even if we consider behaviors equal modulo associativity, commutativity, and idempotence of alternatives.

This observation is no surprise, given that behaviors may give rise to non-regular languages (cf. Example 3). In general, there is no hope to retain Brzozowski's result, but it turns out that we can find a well-behavedness condition for behaviors that is sufficient to retain finiteness of descendants.

5 Well-Behaved Behaviors

In this section, we develop a criterion to guarantee that a forkable expression only gives rise to a finite set of dissimilar descendants. To start with, we adapt Brzozowski's notion of similarity to our setting. In addition to associativity, commutativity, and idempotence we introduce simplification rules that implement further Kleene identities and that deal with forks.

Definition 15 (Similarity). *Behaviors r and s are* similar*, if $r \simeq s$ is derivable using the rules and axioms in Figure 3.*

The compatibility rule (Comp) uses regular contexts E, which are regular expressions with a single hole []. In the rule, we write $E[t]$ to denote the expression with the hole replaced by t.

We establish some basic results for similar behaviors, all with straightforward inductive proofs: Similarity implies semantic equivalence, it is complete for recognizing ε and ϕ, and it is compatible with derivatives and extraction of concurrent parts.

Lemma 16. *If $r \approx s$, then $r \equiv s$.*

Lemma 17. *1. If $L(r) = \{\varepsilon\}$ then $r \approx \varepsilon$. 2. If $L(r) = \{\}$ then $r \approx \phi$.*

Lemma 18. *1. If $r \approx r'$, then $d_x(r) \approx d_x(r')$, for all $x \in \Sigma$.*
2. If $r \approx r'$, then $\mathcal{C}(r) \approx \mathcal{C}(r')$.

Similarity is an equivalence relation. We write $[s] = \{t \mid t \approx s\}$ to denote the equivalence class of all expressions similar to s. If R is a set of behaviors, we write $R/\approx = \{[r] \mid r \in R\}$ for the set of equivalence classes of elements of R.

To identify the set of well-behaved behaviors, we need to characterize the set of dissimilar descendants. First, we establish that each composition of derivatives and applications of $\mathcal{C}()$ that finishes in some $\mathcal{C}(r)$ may be compressed to the composition of the derivatives applied to the remaining $\mathcal{C}(r)$.

Lemma 19. *For a behavior r and symbol x, $\mathcal{C}(d_x(\mathcal{C}(r))) = d_x(\mathcal{C}(r))$, syntactically.*

The above result makes it easier to classify the forms of dissimilar descendants.

The Kleene star case is clearly highly relevant. The following statement confirms the observation in Example 14.

Lemma 20. *For $w \in \Sigma^+$, $d_w(r^*) \approx d_w(r) \cdot r^* + t$ where t is a possibly empty sum of terms of the form $s_1 \cdot \ldots \cdot s_n \cdot r' \cdot r^*$ where $r' \prec r$, $n \geq 1$, and for each s_i, $s_i \preceq \mathcal{C}(s)$ for some descendant $s \prec r$.*

Proof. Induction on w.

Case x: $d_x(r^*) = d_x(r) \cdot r^* \approx d_x(r) \cdot r^* + \phi$.

Case wx: $d_{wx}(r^*) = d_x(d_w(r^*))$. By induction for w, $d_w(r^*) \approx d_w(r) \cdot r^* + t$ where each summand of t has the form $s_1 \cdot \ldots \cdot s_n \cdot r' \cdot r^*$. First, observe that $d_x(d_w(r) \cdot r^* + t) = d_{wx}(r) \cdot r^* + \mathcal{C}(d_w(r)) \cdot d_x(r) \cdot r^* + d_x(t)$. We show by auxiliary induction on n that the derivative of t is a sum of terms of the desired form.

Case 0:
$$\begin{aligned} d_x(r' \cdot r^*) &= d_x(r') \cdot r^* + \mathcal{C}(r') \cdot d_x(r^*) \\ &= d_x(r') \cdot r^* + \mathcal{C}(r') \cdot d_x(r) \cdot r^* \end{aligned}$$

which has the desired format.

Case $n > 0$:
$$\begin{aligned} & d_x(s_1 \cdot \ldots \cdot s_n \cdot r' \cdot r^*) \\ &= d_x(s_1) \cdot s_2 \cdot \ldots \cdot s_n \cdot r' \cdot r^* + \mathcal{C}(s_1) \cdot d_x(s_2 \cdot \ldots \cdot s_n \cdot r' \cdot r^*) \end{aligned}$$

The first summand has the desired form. By induction (on n), each summand of $d_x(s_2 \cdot \ldots \cdot s_n \cdot r' \cdot r^*)$ has the desired form and multiplying with $\mathcal{C}(s_1)$ from the left retains this form: By Lemma 19, $\mathcal{C}(s_1)$ is still a descendant of $\mathcal{C}(s)$ for some descendant s of r. □

To obtain finiteness it appears that a sufficient condition is to ensure that the subterms s_i are trivial (either ε or ϕ). Via similarity, the explosion of terms derived from Kleene star can then be avoided. To verify this claim we also characterize the descendants of concatenated behaviors.

Lemma 21. *For $w \in \Sigma^+$, $d_w(r \cdot s)$ has the form*

$$d_w(r \cdot s) \approx d_w(r) \cdot s + \mathcal{C}(r) \cdot d_w(s) + t$$

where t is a sum of terms of the form $r' \cdot s'$ where s' is a descendant of s and r' is a descendant of $\mathcal{C}(r'')$ and r'' is a descendant of r.

Proof. By induction on w.

Case x: Immediate from the definition of $d_x(r \cdot s)$ with $t = \phi$.

Case wx: By induction

$$\begin{aligned} d_x(d_w(r \cdot s)) &\approx d_x(d_w(r) \cdot s + \mathcal{C}(r) \cdot d_w(s) + t) \\ &\approx \underline{d_{wx}(r) \cdot s} + \mathcal{C}(d_w(r)) \cdot d_x(s) \\ &\quad + d_x(\mathcal{C}(r)) \cdot d_w(s) + \underline{\mathcal{C}(r) \cdot d_{wx}(s)} \\ &\quad + d_x(t) \end{aligned}$$

The underlined summands have the expected forms. The newly created summands have a form corresponding to $r' \cdot s'$. It remains to observe that the derivative of a summand in t has the expected form by Lemma 19 and Lemma 7.

$$d_x(r' \cdot s') = d_x(r') \cdot s' + \mathcal{C}(r') \cdot d_x(s') \qquad \square$$

Definition 22 (Well-behaved Behaviors). *A behavior t is* well-behaved *if all subterms of the form r^* have the property that $\mathcal{C}(d_w(r)) \leqq \varepsilon$, for all $w \in \Sigma^*$.*

The intuition for this definition is simple: Well-behaved behaviors do not fork processes with non-trivial communication behavior in a loop (i.e., under a star). Indeed, we have a simple decidable sufficient condition for well-behavedness.

Lemma 23. *If $r \approx r'$ and r' is fork-free, then $\mathcal{C}(d_w(r)) \leqq \varepsilon$, for all $w \in \Sigma^*$.*

Thus, a behavior is also well-behaved if, for all subterms of the form r^*, r is similar to a fork-free expression.

Recall that $d(r)$ is the set of all descendants of r and $d(r)/(\approx)$ denotes the set of equivalence classes of descendants of r. If we pick a representative from each of these equivalence classes, we obtain the dissimilar descendants of r. In a practical implementation, we may want to compute the *canonical* representative of each equivalence class. See the online version for further details. For the purpose of this paper, an *arbitrary* representative is sufficient.

Definition 24 (Dissimilar Descendants). *We define the set of* dissimilar descendants of r, $d_{\approx}(r)$, *as a complete set of arbitrarily chosen representative behaviors for the equivalence classes $d(r)/(\approx)$.*

We extend the function $\mathcal{C}()$ on behaviors pointwise to sets of behaviors and relation $\approx$ to sets of behaviors by

$$R \approx S \text{ iff } (\forall r \in R. \exists s \in S. r \approx s) \wedge (\forall s \in S. \exists r \in R. r \approx s)$$

Lemma 25. *For any behavior r, $\mathcal{C}(d_{\approx}(\mathcal{C}(d_{\approx}(r)))) \approx d_{\approx}(\mathcal{C}(d_{\approx}(r)))$.*

Proof. Follows from Lemma 19 and Lemma 18. □

Lemma 26. *For any behavior r, $d(r) \approx d_{\approx}(r)$.*

Proof. We need to verify that for each $t_1 \in d(r)$ there exists $t_2 \in d_{\approx}(r)$ such that $t_1 \approx t_2$. We prove this property by induction on the number of derivative steps.

Case $w = \varepsilon$: Then, $t_1 = r$. Clearly, there exists $t_2 \in d_{\approx}(r)$ such that $t_1 \approx t_2$.

Case $w = x \cdot w'$: Then, $t_1 = d_x(d_{w'}(r))$. By the IH, $d_{w'}(r) \approx t_2$ where $t_2 \in d_{\approx}(r)$. By Lemma 18, $t_1 \equiv d_x(t_2)$ where $d_x(t_2) \approx t_3$ for some $t_3 \in d_{\approx}(r)$. Thus, we are done. □

The next result can be verified via similar reasoning.

Lemma 27. *For any behavior r, $d(d_{\approx}(r)) \approx d_{\approx}(r)$.*

Theorem 28 (Finiteness of Well-Behaved Dissimilar Descendants). *Let t be a well-behaved behavior. Then, $\sharp d_{\approx}(t) < \infty$.*

Proof. We need to generalize the statement to obtain the result: If t is well-behaved then $\sharp d_t^i < \infty$ for all $i \geq 0$ where $d_t^0 = d_{\approx}(t)$ and $d_t^{n+1} = d_{\approx}(\mathcal{C}(d_t^n))$.

Based on Lemmas 27 and 25 we find that $d_t^{n+1} = d_t^n$ for $n \geq 1$. That is, in the induction step it is sufficient to establish that d_t^0 and d_t^1 are finite.

We proceed by induction on t. For brevity, we only consider the case of concatenation.

Case $r \cdot s$: By the IH, d_r^i and d_s^i are finite for any $i \geq 0$. We first show that $d_{r \cdot s}^0$ is finite.

1. By Lemma 21, the elements of $d(r \cdot s)$ are drawn from the set
$$d(r) \cdot s + \mathcal{C}(r) \cdot d(s) + \sum d(\mathcal{C}(d(r))) \cdot d(s)$$
2. By Lemma 26 the above is similar to
$$d_{\approx}(r) \cdot s + \mathcal{C}(r) \cdot d_{\approx}(s) + \sum d_{\approx}(\mathcal{C}(d_{\approx}(r))) \cdot d_{\approx}(s)$$
3. Immediately, we can conclude that $d_{r \cdot s}^0$ is finite.

Next, we consider $d_{r \cdot s}^1$.

1. From above, the (dissimilar) descendants of $r \cdot s$ are drawn from
$$\underbrace{d(r) \cdot s}_{t_1} + \underbrace{\mathcal{C}(r) \cdot d(s)}_{t_2} + \sum \underbrace{d(\mathcal{C}(d(r))) \cdot d_{\approx}(s)}_{t_3}$$

For each t_i we will show that $d_{t_i}^1$ is finite and thus follows the desired result.

2. By Lemma 21, descendants of $\mathcal{C}(t_1)$ are of the form
$$d(\mathcal{C}(d(r))) \cdot \mathcal{C}(s) + \mathcal{C}(d(r)) \cdot d(\mathcal{C}(s)) + \sum d(\mathcal{C}(d(\mathcal{C}(d(r))))) \cdot d(\mathcal{C}(s))$$
3. As we know by the IH, d_r^i and d_s^i are finite for any $i \geq 0$. Hence, via similar reasoning as above we can conclude that the above is similar to expressions of the form
$$d_r^1 \cdot \mathcal{C}(s) + \mathcal{C}(d_r^0) \cdot d_s^1 + \sum d_r^2 \cdot d_s^1$$
As all sub-components are finite, we conclude that $d_{t_1}^1$ is finite.
4. We consider the descendants of $\mathcal{C}(t_2)$ which are of the following form
$$d(\mathcal{C}(r)) \cdot \mathcal{C}(d(s)) + \mathcal{C}(r) \cdot d(\mathcal{C}(d(s))) + \sum d(\mathcal{C}(d(\mathcal{C}(r)))) \cdot d(\mathcal{C}(d(s)))$$
5. The above is similar to
$$d_r^1 \cdot \mathcal{C}(d_s^0) + \mathcal{C}(r) \cdot d_r^1 + \sum d_r^2 \cdot d_s^1$$
Thus, we find that $d_{t_2}^1$ is finite.
6. Finally, we observe that shape of descendants of $\mathcal{C}(t_3)$
$$\begin{aligned} &d(\mathcal{C}(d(\mathcal{C}(d(r))))) \cdot \mathcal{C}(d(s)) + \mathcal{C}(d(\mathcal{C}(d(r)))) \cdot d(\mathcal{C}(d(s))) \\ &+ \sum d(\mathcal{C}(d(\mathcal{C}(d(\mathcal{C}(d(r))))))) \cdot d(\mathcal{C}(d(s))) \end{aligned}$$
7. The above is similar to
$$d_r^2 \cdot \mathcal{C}(d_s^0) + \mathcal{C}(d_r^1) \cdot d_s^1 + \sum d_r^3 \cdot d_s^1$$
8. Then, $d_{t_3}^1$ is finite which concludes the proof for this case.

□

The result no longer holds if we replace the assumption $\mathcal{C}(d_w(r)) \leqq \varepsilon$, for $w \in \Sigma^*$ by a simpler assumption like $\mathcal{C}(r) \leqq \varepsilon$. For example, consider the behavior $(x \cdot \mathit{Fork}(y))^*$ where $\mathcal{C}(x \cdot \mathit{Fork}(y)) = \phi \leqq \varepsilon$. However, the set of dissimilar descendants of $(x \cdot \mathit{Fork}(y))^*$ is infinite as shown by the calculation

$$\begin{aligned} & (x \cdot \mathit{Fork}(y))^* \\ \xrightarrow{x}\ & (\varepsilon \cdot \mathit{Fork}(y)) \cdot (x \cdot \mathit{Fork}(y))^* \\ \approx\ & \mathit{Fork}(y) \cdot (x \cdot \mathit{Fork}(y))^* \\ \xrightarrow{x}\ & \mathit{Fork}(\phi) \cdot (x \cdot \mathit{Fork}(y))^* + \mathit{Fork}(y) \cdot \mathit{Fork}(y) \cdot (x \cdot \mathit{Fork}(y))^* \\ \approx\ & \mathit{Fork}(y) \cdot \mathit{Fork}(y) \cdot (x \cdot \mathit{Fork}(y))^* \\ & ... \end{aligned}$$

The example also shows that the assumption $\mathcal{C}(d_w(r)) \leqq \varepsilon$, for $w \in \Sigma^*$ is necessary and cannot be weakened to words w of a fixed length.

As an example, consider the behavior $t = (x_1 \cdot ... \cdot x_n \cdot x_{n+1}\mathit{Fork}(y))^*$ where for all $w \in \Sigma^*$ with length less or equal n we find that $\mathcal{C}(d_w(x_1 \cdot ... \cdot x_n \cdot x_{n+1}\mathit{Fork}(y))) \leqq \varepsilon$. Via a similar calculation as above, we can show that the set of dissimilar descendants of t is infinite.

6 Related Work

Shuffle expressions are regular expressions with operators for shuffle and shuffle closure. Shaw [13] proposes to describe the behavior of software using flow expressions, which extend shuffle expressions with further operators. Gischer [6] shows that shuffle expression generate context-sensitive languages and proposes a connection to Petri net languages.

The latter connection is made precise by Garg and Ragunath [4], who study concurrent regular expressions (CRE), which are shuffle expressions extended with synchronous composition. They show that the class of CRE languages is equal to the class of Petri net languages. The proof requires the presence of synchronous composition. Forkable expressions do not support synchronous composition, but they are equivalent to *unit expressions*, which are also defined by Garg and Ragunath and shown to be strictly less powerful than CREs.

Warmuth and Haussler [14] present more refined complexity results for the languages generated by shuffle expressions. Jedrzejowicz [8] shows that the nesting of iterated closure operators matters.

Acknowledgments

We thank the reviewers for their comments.

References

1. Amtoft, T., Nielson, F., Nielson, H.R.: Type and behaviour reconstruction for higher-order concurrent programs. Journal of Functional Programming 7(3), 321–347 (May 1997)
2. Amtoft, T., Nielson, H.R., Nielson, F.: Behavior analysis for validating communication patterns. STTT 2(1), 13–28 (1998)
3. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM 11(4), 481–494 (1964)
4. Garg, V.K., Ragunath, M.T.: Concurrent regular expressions and their relationship to Petri nets. Theor. Comput. Sci. 96(2), 285–304 (Apr 1992)
5. Ginsburg, S.: The Mathematical Theory of Context-Free Languages. McGraw-Hill, Inc., New York, NY, USA (1966)
6. Gischer, J.: Shuffle languages, Petri nets, and context-sensitive grammars. Commun. ACM 24(9), 597–605 (Sep 1981)
7. Jantzen, M.: Extending regular expressions with iterated shuffle. Theor. Comput. Sci. 38, 223–247 (1985)
8. Jedrzejowicz, J.: Nesting of shuffle closure is important. Inf. Process. Lett. 25(6), 363–368 (1987)
9. Kozen, D.: On Kleene algebras and closed semirings. In: Proceedings of the Mathematical Foundations of Computer Science 1990. pp. 26–47. MFCS '90, Springer-Verlag, London, UK, UK (1990)
10. Leiß, H.: Towards Kleene algebra with recursion. In: Proc. of CSL'91. LNCS, vol. 626, pp. 242–256. Springer (1992)
11. Nielson, H.R., Amtoft, T., Nielson, F.: Behaviour analysis and safety conditions: A case study in CML. In: FASE. pp. 255–269 (1998)

12. Nielson, H.R., Nielson, F.: Higher-order concurrent programs with finite communication topology. In: Proc. of POPL'94. pp. 84–97. ACM Press (Jan 1994)
13. Shaw, A.C.: Software descriptions with flow expressions. IEEE Trans. Software Eng. 4(3), 242–254 (1978)
14. Warmuth, M.K., Haussler, D.: On the complexity of iterated shuffle. J. Comput. Syst. Sci. 28(3), 345–358 (1984)

A Supplementary Material

A.1 Well-definedness of L

For each r, the mapping $K \mapsto L(r, K)$ is a monotone mapping in $\wp(\Sigma^*) \to \wp(\Sigma^*)$. The proof is by induction on r.

Case ϕ: $K \mapsto L(\phi, K) = \emptyset$ is monotone.

Case ε: $K \mapsto L(\varepsilon, K) = K$ is monotone.

Case x: $K \mapsto L(x, K) = x \cdot K$ is monotone.

Case $r + s$: $K \mapsto L(r + s, K) = L(r, K) \cup L(s, K)$ is monotone by induction.

Case $r \cdot s$: $K \mapsto L(r \cdot s, K) = L(r, L(s, K))$ is monotone by induction.

Case r^*: $K \mapsto L(r^*, K) = \mu X.L(r, X) \cup K$. By induction, $f = X \mapsto L(r, X)$ is monotone. Hence, $g_K = X \mapsto L(r, X) \cup K$ is monotone, for all K. Furthermore, $f \leq g_K$ and $g_K \leq g_L$ in the pointwise ordering of set-valued functions, whenever $K \subseteq L$. It follows that $\mu g_K \leq \mu g_L$, which proves that $K \mapsto L(r^*, K)$ is monotone.

Case $\mathit{Fork}(r)$: $K \mapsto L(\mathit{Fork}(r), K) = L(r) \| K$. Immediate by induction and because $\|$ is monotone.

A.2 Proof of Theorem 6

Proof. By induction on r.

Case ϕ**,** ε**,** x**,** $\mathit{Fork}(r)$**:** immediate.

Case $r + s$**:** immediate by induction.

Case $r \cdot s$**:**
$$\begin{aligned}
\mathcal{C}(r \cdot s) + \mathcal{S}(r \cdot s) &= \mathcal{C}(r) \cdot \mathcal{C}(s) + \mathcal{C}(r) \cdot \mathcal{S}(s) + \mathcal{S}(r) \cdot s \\
&\equiv \mathcal{C}(r) \cdot (\mathcal{C}(s) + \mathcal{S}(s)) + \mathcal{S}(r) \cdot s \\
&\overset{IH}{\equiv} \mathcal{C}(r) \cdot s + \mathcal{S}(r) \cdot s \\
&\equiv (\mathcal{C}(r) + \mathcal{S}(r)) \cdot s \\
&\overset{IH}{\equiv} r \cdot s
\end{aligned}$$

Case r^***:**
$$\begin{aligned}
\mathcal{C}(r^*) + \mathcal{S}(r^*) &= \mathcal{C}(r)^* + \mathcal{C}(r)^* \cdot \mathcal{S}(r) \cdot r^* \\
&\overset{IH}{\equiv} \mathcal{C}(r)^* + \mathcal{C}(r)^* \cdot \mathcal{S}(r) \cdot (\mathcal{C}(r) + \mathcal{S}(r))^* \\
&\equiv (\mathcal{C}(r) + \mathcal{S}(r))^* \\
&\overset{IH}{\equiv} r^*
\end{aligned}$$

□

A.3 Proof of Theorem 2

Proof. Induction on r where we generalize the statement to $L(r, K) = [\![r]\!] \cdot K$, for all $K \subseteq \Sigma^*$, and use Arden's lemma for the case r^*. The theorem follows by setting $K = \{\varepsilon\}$.

Cases ϕ, ε, x: immediate.

Case $r + s$: immediate by IH.

Case $r \cdot s$: $L(r \cdot s, K) = L(r, L(s, K)) = [\![r]\!] \cdot ([\![s]\!] \cdot K) = [\![r \cdot s]\!] \cdot K$.
Case r^*:

$$\begin{aligned} & L(r^*, K) \\ = & \text{ (by definition)} \\ & \mu X.L(r, X) \cup K \\ = & \text{ (by IH, } X \text{ must be in } \Sigma^*) \\ & = \mu X.[\![r]\!] \cdot X \cup K \\ & \text{(by Arden's lemma)} \\ & = [\![r^*]\!] \cdot K \end{aligned}$$

□

A.4 Proof of Theorem 5

Proof. Following [9] a Kleene algebra satisfies the axioms of an idempotent semiring and the following axioms:

1. $\varepsilon + r \cdot r^* \leqq r^*$
2. $\varepsilon + r^* \cdot r \leqq r^*$
3. If $r \cdot s \leqq s$ then $r^* \cdot s \leqq s$
4. If $s \cdot r \leqq s$ then $s \cdot r^* \leqq s$

It is straightforward to verify that behaviors form on idempotent semiring.

Axioms (1) and (2) follow by calculation from the semantics.

Consider (3). Suppose $r \cdot s \leqq s$. We will show that by induction for any n we have that $r^n \cdot s \leqq s$ where $r^0 = \varepsilon$ and $r^{n+1} = r \cdot r^n$. Case $n = 0$ holds immediately. For case $n + 1$ we perform the following calculations.

$$\begin{aligned} & r^{n+1} \cdot s \\ = & \ r \cdot r^n \cdot s \\ & \text{(IH)} \\ \leqq & \ r \cdot s \\ \leqq & \ s \end{aligned}$$

Thus, we find that $r^* \cdot s \leqq s$. Case (4) can be verified similarly. □

A.5 Proof of Lemma 7

Proof. 1. Trivial induction.

2. Induction on r. The cases for ϕ, ε, l, $r + s$, $\mathit{Fork}(r)$ are trivial or immediate by the inductive hypothesis.

$$\begin{aligned} \mathcal{C}(\mathcal{S}(r \cdot s)) &= \mathcal{C}(\mathcal{S}(r) \cdot s + \mathcal{C}(r) \cdot \mathcal{S}(s)) \\ &= \mathcal{C}(\mathcal{S}(r) \cdot s) + \mathcal{C}(\mathcal{C}(r) \cdot \mathcal{S}(s)) \\ &= \mathcal{C}(\mathcal{S}(r)) \cdot \mathcal{C}(s) + \mathcal{C}(\mathcal{C}(r)) \cdot \mathcal{C}(\mathcal{S}(s)) \\ &\equiv \phi \cdot \mathcal{C}(s) + \mathcal{C}(r) \cdot \phi \\ &\equiv \phi \end{aligned}$$

$$\begin{aligned}\mathcal{C}(\mathcal{S}(r^*)) &= \mathcal{C}(\mathcal{C}(r)^* \cdot \mathcal{S}(r) \cdot r^*) \\ &= \mathcal{C}(\mathcal{C}(r)^*) \cdot \mathcal{C}(\mathcal{S}(r)) \cdot \mathcal{C}(r^*) \\ &\equiv \mathcal{C}(\mathcal{C}(r))^* \cdot \phi \cdot \mathcal{C}(r)^* \\ &\equiv \phi\end{aligned}$$

3. Induction on r. The cases for ϕ, ε, l, $r + s$, $\mathit{Fork}(r)$ are trivial or immediate by inductive hypothesis.

$$\begin{aligned}\mathcal{S}(\mathcal{C}(r \cdot s)) &= \mathcal{S}(\mathcal{C}(r) \cdot \mathcal{C}(s)) \\ &= \mathcal{S}(\mathcal{C}(r)) \cdot \mathcal{C}(s) + \mathcal{C}(\mathcal{C}(r)) \cdot \mathcal{S}(\mathcal{C}(s)) \\ &\equiv \phi \cdot \mathcal{C}(s) + \mathcal{C}(r) \cdot \phi \\ &\equiv \phi\end{aligned}$$

$$\begin{aligned}\mathcal{S}(\mathcal{C}(r^*)) &= \mathcal{S}(\mathcal{C}(r)^*) \\ &= \mathcal{C}(\mathcal{C}(r))^* \cdot \mathcal{S}(\mathcal{C}(r)) \cdot \mathcal{C}(r)^* \\ &\equiv \mathcal{C}(r)^* \cdot \phi \cdot \mathcal{C}(r)^* \\ &\equiv \phi\end{aligned}$$

4. Induction on r. The cases for ϕ, ε, l, $r + s$, $\mathit{Fork}(r)$ are trivial or immediate by inductive hypothesis.

$$\begin{aligned}\mathcal{S}(\mathcal{S}(r \cdot s)) &= \mathcal{S}(\mathcal{S}(r) \cdot s + \mathcal{C}(r) \cdot \mathcal{S}(s)) \\ &= \mathcal{S}(\mathcal{S}(r) \cdot s) + \mathcal{S}(\mathcal{C}(r) \cdot \mathcal{S}(s)) \\ &= \mathcal{S}(\mathcal{S}(r)) \cdot s + \mathcal{C}(\mathcal{S}(r)) \cdot \mathcal{S}(s) + \mathcal{S}(\mathcal{C}(r)) \cdot \mathcal{S}(s) + \mathcal{C}(\mathcal{C}(r)) \cdot \mathcal{S}(\mathcal{S}(s)) \\ &\{\text{by 1., 2., 3., and the inductive hypothesis}\} \\ &\equiv \mathcal{S}(r) \cdot s + \mathcal{C}(r) \cdot \mathcal{S}(s)\end{aligned}$$

$$\begin{aligned}\mathcal{S}(\mathcal{S}(r^*)) &= \mathcal{S}(\mathcal{C}(r)^* \cdot \mathcal{S}(r) \cdot r^*) \\ &= \mathcal{S}(\mathcal{C}(r)^*) \cdot \mathcal{S}(r) \cdot r^* + \mathcal{C}(\mathcal{C}(r)^*) \cdot \mathcal{S}(\mathcal{S}(r) \cdot r^*) \\ &= \mathcal{S}(\mathcal{C}(r)^*) \cdot \mathcal{S}(r) \cdot r^* + \mathcal{C}(r)^* \cdot (\mathcal{S}(\mathcal{S}(r)) \cdot r^* + \mathcal{C}(\mathcal{S}(r)) \cdot \mathcal{S}(r^*)) \\ &= \mathcal{S}(\mathcal{C}(r)^*) \cdot \mathcal{S}(r) \cdot r^* + \mathcal{C}(r)^* \cdot (\mathcal{S}(r) \cdot r^* + \phi \cdot \mathcal{S}(r^*)) \\ &= \mathcal{S}(\mathcal{C}(r)^*) \cdot \mathcal{S}(r) \cdot r^* + \mathcal{C}(r)^* \cdot (\mathcal{S}(r) \cdot r^*) \\ &= \mathcal{C}(\mathcal{C}(r))^* \cdot \mathcal{S}(\mathcal{C}(r)) \cdot \mathcal{C}(r)^* \cdot \mathcal{S}(r) \cdot r^* + \mathcal{C}(r)^* \cdot (\mathcal{S}(r) \cdot r^*) \\ &\equiv \mathcal{C}(r)^* \cdot \phi \cdot \mathcal{C}(r)^* \cdot \mathcal{S}(r) \cdot r^* + \mathcal{C}(r)^* \cdot (\mathcal{S}(r) \cdot r^*) \\ &\equiv \mathcal{C}(r)^* \cdot (\mathcal{S}(r) \cdot r^*)\end{aligned}$$

□

A.6 Proof of Lemma 8

Proof. If $\varepsilon \leqq \mathcal{C}(r)$ and by decomposition $r \equiv \mathcal{S}(r) + \mathcal{C}(r)$ we find that $\varepsilon \in L(r)$.

The other direction requires induction on r. For brevity, we only consider some cases.

$$\begin{aligned} & \varepsilon \in L(r \cdot s) \\ \implies & \varepsilon \in L(r) \wedge \varepsilon \in L(s) \\ \overset{\text{IH}}{\implies} & \varepsilon \leqq \mathcal{C}(r) \wedge \varepsilon \leqq \mathcal{C}(s) \\ \implies & \varepsilon \leqq \mathcal{C}(r) \cdot \mathcal{C}(s) = \mathcal{C}(r \cdot s) \end{aligned}$$

For Kleene star, by definition $\mathcal{C}(r^*) = \mathcal{C}(r)^*$ and $\varepsilon \leqq \mathcal{C}(r)^*$ is a derived property in a Kleene algebra. Thus, $\varepsilon \leqq \mathcal{C}(r^*)$ follows immediately. □

A.7 Auxiliary Statements

Lemma 29. *Let r, s be behaviors. Then $L(\mathcal{C}(r)) \| L(\mathcal{C}(s)) = L(\mathcal{C}(r \cdot s))$.*

Proof.

$$\begin{aligned} L(\mathcal{C}(r)) \| L(\mathcal{C}(s)) &= L(\mathcal{C}(r), L(\mathcal{C}(s))) \\ &= L(\mathcal{C}(r), L(\mathcal{C}(s), \{\varepsilon\})) \\ &= L(\mathcal{C}(r \cdot s), \{\varepsilon\}) \\ &= L(\mathcal{C}(r \cdot s)) \end{aligned}$$

□

Lemma 30. *Let r be a behavior and K a language. Then $L(\mathcal{C}(r), K) = L(\mathcal{C}(r)) \| K$*

Proof. Induction on f.

Case ϕ.
Trivial as $\mathcal{C}(\phi) = \phi$, $L(\phi, K) = \emptyset = \emptyset \| K$.
Case ε.
$\mathcal{C}(\varepsilon) = \varepsilon$. $L(\varepsilon, K) = K = \{\varepsilon\} \| K$.
Case x.
$\mathcal{C}(x) = \phi$. $L(\phi, K) = \emptyset = \emptyset \| K$.
Case $r + s$.

$$\begin{aligned} L(\mathcal{C}(r + s), K) &= L(\mathcal{C}(r), K) \cup L(\mathcal{C}(s), K) \\ &= L(\mathcal{C}(r)) \| K \cup L(\mathcal{C}(s)) \| K \\ &= L(\mathcal{C}(r + s)) \| K \end{aligned}$$

Case $r \cdot s$. (requires Lemma 29)

$$\begin{aligned} L(\mathcal{C}(r \cdot s), K) &= L(\mathcal{C}(r) \cdot \mathcal{C}(s), K) \\ &= L(\mathcal{C}(r), L(\mathcal{C}(s), K)) \\ &= L(\mathcal{C}(r)) \| L(\mathcal{C}(s)) \| K \\ &= L(\mathcal{C}(r \cdot s)) \| K \end{aligned}$$

Case r^*.

$$\begin{aligned}
L(\mathcal{C}(r^*), K) &= L(\mathcal{C}(r)^*, K) \\
&= \mu X.(L(\mathcal{C}(r), X) \cup K) \\
&\{\text{IH}\} \\
&= \mu X.(L(\mathcal{C}(r))\|X \cup K) \\
&= L(\mathcal{C}(r))^\sharp\|K \\
&= (\mu X.L(\mathcal{C}(r))\|X \cup \{\varepsilon\})\|K \\
&\{\text{IH}\} \\
&= (\mu X.L(\mathcal{C}(r), X) \cup \{\varepsilon\})\|K \\
&= L(\mathcal{C}(r)^*, \{\varepsilon\})\|K \\
&= L(\mathcal{C}(r^*))\|K
\end{aligned}$$

Case $\mathit{Fork}(r)$.

$$\begin{aligned}
L(\mathcal{C}(\mathit{Fork}(r)), K) &= L(\mathit{Fork}(r), K) \\
&= L(r)\|K \\
&= L(r)\|\{\varepsilon\}\|K \\
&= L(\mathit{Fork}(r), \{\varepsilon\})\|K \\
&= L(\mathcal{C}(\mathit{Fork}(r)))\|K
\end{aligned}$$

□

A.8 Remaining cases of Theorem 10

Proof. **Case** $\mathit{Fork}(r)$.

$$\begin{aligned}
&L(d_x(\mathit{Fork}(r)), K) \cup L(\mathcal{C}(\mathit{Fork}(r)))\|x\backslash K \\
&= L(\mathit{Fork}(d_x(r)), K) \cup L(\mathit{Fork}(r))\|x\backslash K \\
&= L(d_x(r))\|K \cup L(r)\|x\backslash K \\
&\{\text{IH}\} \\
&= x\backslash L(r)\|K \cup L(r)\|x\backslash K \\
&= x\backslash(L(r)\|K) \\
&= x\backslash(L(\mathit{Fork}(r), K))
\end{aligned}$$

Case ϕ. $\mathcal{C}(\phi) = \phi$.
$L(d_x(\phi), K) = L(\phi, K) = \emptyset = x\backslash L(\phi, K)$
Case ε. $\mathcal{C}(\varepsilon) = \varepsilon$.
$L(d_x(\varepsilon), K) \cup L(\mathcal{C}(\varepsilon))\|(x\backslash K) = L(\phi, K) \cup x\backslash K = x\backslash K = x\backslash L(\varepsilon, K)$

Case x. $\mathcal{C}(x) = \phi$.
$L(d_x(x), K) = L(\varepsilon, K) = K = x\backslash L(x, K)$
Case $y \neq x$.
$L(d_x(y), K) = L(\phi, K) = \emptyset = x\backslash L(y, K)$
Case $r + s$. $\mathcal{C}(r + s) = \mathcal{C}(r) + \mathcal{C}(s)$.

$$\begin{aligned}
&L(d_x(r+x), K) \cup L(\mathcal{C}(r+s))\|(x\backslash K) \\
&= L(d_x(r), K) \cup L(\mathcal{C}(r))\|(x\backslash K) \cup L(d_x(s), K) \cup L(\mathcal{C}(s))\|(x\backslash K) \\
&\{\text{IH}\} \\
&= x\backslash L(r, K) \cup x\backslash L(s, K) \\
&= x\backslash L(r+s, K)
\end{aligned}$$

Case $r \cdot s$. $\mathcal{C}(r \cdot s) = \mathcal{C}(r) \cdot \mathcal{C}(s)$.

$$\begin{aligned}
&L(d_x(r \cdot s), K) \cup L(\mathcal{C}(r \cdot s))\|(x\backslash K) \\
&= L(d_x(r) \cdot s, K) \cup L(\mathcal{C}(r) \cdot d_x(s), K) \cup L(\mathcal{C}(s))\|L(\mathcal{C}(s))\|(x\backslash K) \\
&= L(d_x(r), L(s, K)) \cup L(\mathcal{C}(r), L(d_x(s), K)) \cup L(\mathcal{C}(r))\|L(\mathcal{C}(s))\|(x\backslash K) \\
&= L(d_x(r), L(s, K)) \cup L(\mathcal{C}(r))\|L(d_x(s), K) \cup L(\mathcal{C}(r))\|L(\mathcal{C}(s))\|(x\backslash K) \\
&= L(d_x(r), L(s, K)) \cup L(\mathcal{C}(r))\|\big(L(d_x(s), K) \cup L(\mathcal{C}(s))\|(x\backslash K)\big) \\
&\{\text{IH}\} \\
&= L(d_x(r), L(s, K)) \cup L(\mathcal{C}(r))\|(x\backslash L(s, K)) \\
&\{\text{IH}\} \\
&= x\backslash L(r, L(s, K)) \\
&= x\backslash L(r \cdot s, K)
\end{aligned}$$

Case r^*.
We verify for all $n \leq 0$ that

$$x\backslash L(r^n, K) = L(d_x(r^n), K) \cup L(\mathcal{C}(r^n))\|(x\backslash K)$$

where $r^0 = \varepsilon$ and $r^{n+1} = r \cdot r^n$.
Case $n = 0$**:** Straightforward
Case $n \implies n + 1$**:**

$$\begin{aligned}
&L(d_x(r^{n+1}), K) \cup L(\mathcal{C}(r^{n+1}))\|(x\backslash K) \\
&= L(d_x(r) \cdot r^n + \mathcal{C}(r) \cdot d_x(r^n), K) \cup L(\mathcal{C}(r^{n+1}))\|(x\backslash K) \\
&= L(d_x(r), L(r^n, K)) \cup L(\mathcal{C}(r), L(d_x(r^n), K)) \cup L(\mathcal{C}(r^{n+1}))\|(x\backslash K) \\
&= L(d_x(r), L(r^n, K)) \cup L(\mathcal{C}(r), L(d_x(r^n), K)) \cup L(\mathcal{C}(r), L(\mathcal{C}(r^n)))\|(x\backslash K) \\
&\quad \text{(Lemma 30)} \\
&= L(d_x(r), L(r^n, K)) \cup L(\mathcal{C}(r))\|L(d_x(r^n), K) \cup L(\mathcal{C}(r)\|L(\mathcal{C}(r^n))\|(x\backslash K) \\
&= L(d_x(r), L(r^n, K)) \cup L(\mathcal{C}(r))\|(L(d_x(r^n), K) \cup L(\mathcal{C}(r^n))\|(x\backslash K)) \\
&\quad \text{(IH)} \\
&= L(d_x(r), L(r^n, K)) \cup L(\mathcal{C}(r))\|(x\backslash(L(r^n, K))) \\
&\quad \text{(IH)} \\
&= x\backslash L(r, L(r^n, K)) \\
&= x\backslash L(r^{n+1}, K)
\end{aligned}$$

Then, the actual statement

$$l\backslash L(r^*, K) = L(d_x(r^*), K) \cup L(\mathcal{C}(r^*))\|(x\backslash K)$$

follows from the fact that

$$(x\backslash L(r^n, K)) \leq (x\backslash L(b^*, K))$$

$$L(d_x(r^n), K) \cup L(\mathcal{C}(r^n))\|(x\backslash K) \leq L(d_x(r^*), K) \cup L(\mathcal{C}(r^*))\|(x\backslash K)$$

and by showing that for all finite words $w \in \Sigma^*$ we have that $w \in (x\backslash L(r^*, K))$ iff $w \in L(d_x(r^*), K) \cup L(\mathcal{C}(r^*))\|(x\backslash K)$.

If $w \in (x\backslash L(r^*, K))$ then $w \in (x\backslash L(b^n, K))$ for some $n \leq 0$. From above, we conclude that $w \in L(d_x(r^n), K) \cup L(\mathcal{C}(r^n))\|(x\backslash K) \leq L(d_x(r^*), K) \cup L(\mathcal{C}(r^*))\|(x\backslash K)$. The argument is similar for the other direction. □

A.9 Proof of Lemma 17

Proof. By induction on r. Consider $L(r) = \{\varepsilon\}$.

Cases x and ϕ do not apply. Case ε is straightforward.

Case $r + s$**:** $L(r + s) = \{\varepsilon\}$ implies that $L(r) \subseteq \{\varepsilon\}$ and $L(s) \subseteq \{\varepsilon\}$. There are four subcases to consider. Suppose $L(r) = \{\varepsilon\}$ and $L(s) = \{\}$. By IH, $r \approx \varepsilon$ and either $s \approx \phi$. Then, $r + s \approx \varepsilon$, by (Unit). Other cases are similar.

Case $r \cdot s$**:** $L(r \cdot s) = L(r) \cdot L(s) = \{\varepsilon\}$ implies that $L(r) = \{\varepsilon\}$ and $L(s) = \{\varepsilon\}$. By IH, $r \approx \varepsilon$ and $s \approx \varepsilon$. Hence, $r \cdot s \approx \varepsilon$, by (Empty Word).

Case r^***:** $L(r^*) = \{\varepsilon\}$ implies that $L(r) = \{\varepsilon\}$. By IH, $r \approx \varepsilon$ from which we can conclude that $r^* \approx \varepsilon$.

Case $\mathit{Fork}(r)$**:** Similar to the above.

Consider $L(r) = \{\}$. Cases x, ε and r^* do not apply. Case ϕ is straightforward.

Case $r \cdot s$**:** $L(r \cdot s) = \{\}$ implies that $L(r) = \{\} \vee L(s) = \{\}$. Suppose $L(r) = \{\}$. By IH, $r \approx \phi$ and therefore $r \cdot s \approx \phi$. The other case is similar.

Case $r + s$**:** Similar to the above.

Case $\mathit{Fork}(r)$**:** $L(\mathit{Fork}(r)) = \{\}$ implies $L(r) = \{\}$. Via similar arguments as above we find that $\mathit{Fork}(r) \approx \phi$. □

A.10 Proof of Lemma 19

Proof. **Case** $r = \phi, \varepsilon, x$: trivial as $\mathcal{C}(r) = \phi$.

Case $r + s$: immediate from the IH.

Case $r \cdot s$:

$$\begin{aligned}
&d_x(\mathcal{C}(r \cdot r)) \\
&= d_x(\mathcal{C}(r) \cdot \mathcal{C}(x)) \\
&= d_x(\mathcal{C}(r)) \cdot \mathcal{C}(s) + \mathcal{C}(\mathcal{C}(r)) \cdot d_x(\mathcal{C}(x)) \\
&\qquad \{\text{IH, Lemma 7 part 1}\} \\
&= \mathcal{C}(d_x(\mathcal{C}(r))) \cdot \mathcal{C}(\mathcal{C}(s)) + \mathcal{C}(\mathcal{C}(\mathcal{C}(r))) \cdot \mathcal{C}(d_x(\mathcal{C}(s))) \\
&= \mathcal{C}(d_x(\mathcal{C}(r \cdot s)))
\end{aligned}$$

Case r^*:

$$\begin{aligned}
&d_l(\mathcal{C}(r^*)) \\
&= d_x(\mathcal{C}(r)^*) \\
&= d_x(\mathcal{C}(r)) \cdot \mathcal{C}(r)^* \\
&\quad \{\text{IH, Lemma 7 part 1}\} \\
&= \mathcal{C}(d_x(\mathcal{C}(r))) \cdot \mathcal{C}(\mathcal{C}(r))^* \\
&= \mathcal{C}(d_x(\mathcal{C}(r^*)))
\end{aligned}$$

Case $\mathit{Fork}(r)$:

$$\begin{aligned}
&d_x(\mathcal{C}(\mathit{Fork}(r))) \\
&= d_x(\mathit{Fork}(r)) \\
&= \mathit{Fork}(d_x(r)) \\
&= \mathcal{C}(\mathit{Fork}(d_x(r))) \\
&= \mathcal{C}(d_x(\mathcal{C}(\mathit{Fork}(r))))
\end{aligned}$$

□

A.11 Proof of Theorem 28

Proof. We need to generalize the statement to obtain the result: If t is well-behaved then $\sharp d_t^i < \infty$ for all $i \geq 0$ where $d_t^0 = d_{\approx}(t)$ and $d_t^{n+1} = d_{\approx}(\mathcal{C}(d_t^n))$.

Based on Lemmas 27 and 25 we find that $d_t^{n+1} = d_t^n$ for $n \geq 1$. That is, in the induction step it is sufficient to establish that d_t^0 and d_t^1 are finite. We proceed by induction on t.

Cases ε, x, ϕ**:** Straightforward.

Case $r + s$**:** By the IH, d_r^i, d_s^i are finite for any $i \geq 0$. We know that $d(r + s) = d(r) + d(s)$. The above can be represented by $d_{\approx}(r) + d_{\approx}(s)$. Immediately, we find d_{r+s}^0 is finite.

Via similar reasoning, we can show that the $d(\mathcal{C}(d(r + s)))$ can be represented by $d_{\approx}(\mathcal{C}(d_{\approx}(r))) + d_{\approx}(\mathcal{C}(d_{\approx}(s)))$. Hence, d_{r+s}^1 which concludes the proof for this case.

Case $\mathit{Fork}(r)$: By the IH, d_r^i are finite for any $i \geq 0$.

From $d(\mathit{Fork}(r)) = \mathit{Fork}(d(r))$ and the above we can conclude that $d_{\mathit{Fork}(r)}^0$ is finite.

To establish $d_{\mathit{Fork}(r)}^1$ is finite, we apply the following reasoning.

$$\begin{aligned}
&d_{\approx}(\mathcal{C}(d_{\approx}(\mathit{Fork}(r)))) \\
\approx\ &d_{\approx}(\mathcal{C}(\mathit{Fork}(d_{\approx}(r)))) \\
\approx\ &d_{\approx}(\mathit{Fork}(d_{\approx}(r))) \\
\approx\ &\mathit{Fork}(d_{\approx}(d_{\approx}(r))) \\
\approx\ &\mathit{Fork}(d_{\approx}(r))
\end{aligned}$$

The set $d_{\approx}(r)$ is finite. Hence, $d_{\mathit{Fork}(r)}^1$ is finite.

Case r^***:** By the IH, d_r^i are finite for any $i \geq 0$. We show that $d_{r^*}^0$ is finite.

1. By Lemma 20, descendants of r^* are of the form

$$s_1 \cdot ... \cdot s_n \cdot d(r) \cdot r^*$$

where $s_i \in d(\mathcal{C}(d_w(r)))$ for some $w \in \Sigma^+$.
2. From Lemma 17 and the assumption that all subterms of the form r^* have the property that $\mathcal{C}(d_w(r)) \leq \varepsilon$, for $w \in \Sigma^*$, we conclude that the above is either similar to ϕ or $d_r^0 \cdot r^*$.
3. Hence, $d_{r^*}^0$ is finite.

Next, we show that $d_{r^*}^1$ is finite.

1. We observe the possible forms of (dissimilar) descendants of r^*.
2. For case ϕ we immediately find that $d_{\approx}(\mathcal{C}(\phi))$ is finite.
3. For case $d_r^0 \cdot r^*$, we consider the possible descendants of $\mathcal{C}(d_r^0 \cdot r^*) = \mathcal{C}(d_r^0) \cdot \mathcal{C}(r^*)$.
4. By Lemma 21, the shape of such terms is of the form

$$d(\mathcal{C}(d_r^0)) \cdot \mathcal{C}(r^*) + \mathcal{C}(d_r^0) \cdot d(\mathcal{C}(r^*)) + \sum d(\mathcal{C}(d(\mathcal{C}(d_r^0)))) \cdot d(\mathcal{C}(r^*))$$

5. By Lemma 26 the above is similar to

$$d_r^1 \cdot \mathcal{C}(r^*) + \mathcal{C}(d_r^0) \cdot d(\mathcal{C}(r^*)) + \sum d_r^2 \cdot d(\mathcal{C}(r^*))$$

6. We know that d_r^i are finite. What remains is to show that $d_{\approx}(\mathcal{C}(r^*))$ is finite.
7. Via similar reasoning as above we can argue that the descendants of $\mathcal{C}(r^*) = \mathcal{C}(r)^*$ are of the form

$$t_1 \cdot ... \cdot t_m \cdot d(\mathcal{C}(r)) \cdot \mathcal{C}(r)^*$$

where $t_i \in d(\mathcal{C}(d_w(\mathcal{C}(r))))$ for some $w \in \Sigma^+$.
8. Each t_i is either ϕ or ε based on the following approximation. First, we find that $\mathcal{C}(d_w(\mathcal{C}(r))) \leq \mathcal{C}(d_w(\mathcal{C}(r) + \mathcal{S}(r))) = \mathcal{C}(d_w(r))$. As we know that $\mathcal{C}(d_w(r)) \leq \varepsilon$ for $w \in \Sigma^+$, we can conclude that $t_i \leq \varepsilon$.
9. Hence, descendants of $\mathcal{C}(r)^*$ are either similar to ϕ or terms of the form $d(\mathcal{C}(r)) \cdot \mathcal{C}(r)^*$.
10. As we know d_r^1 is finite and subsumes $d(\mathcal{C}(r))$ we can conclude that $d_{\approx}(\mathcal{C}(r^*))$ is finite. Thus, we are done.

Case $r \cdot s$**:** By the IH, d_r^i and d_s^i are finite for any $i \geq 0$. We show that $d_{r \cdot s}^0$ is finite.

1. By Lemma 21, each derivative $d_w(r \cdot s)$ has the form

$$d(r) \cdot s + \mathcal{C}(r) \cdot d(s) + \sum d(\mathcal{C}(d(r))) \cdot d(s)$$

2. By Lemma 26 the above is similar to

$$d_{\approx}(r) \cdot s + \mathcal{C}(r) \cdot d_{\approx}(s) + \sum d_{\approx}(\mathcal{C}(d_{\approx}(r))) \cdot d_{\approx}(s)$$

3. Immediately, we can conclude that $d^0_{r\cdot s}$ is finite.

Next, we consider $d^1_{r\cdot s}$.

1. From above, the form of (dissimilar) descendants of $r \cdot s$ is

$$\underbrace{d(r)\cdot s}_{t_1} + \underbrace{\mathcal{C}(r)\cdot d(s)}_{t_2} + \sum \underbrace{d(\mathcal{C}(d(r)))\cdot d_{\approx}(s)}_{t_3}$$

   For each t_i we will show that $d^1_{t_i}$ is finite and thus follows the desired result.
2. By Lemma 21, descendants of $\mathcal{C}(t_1)$ are of the form

$$d(\mathcal{C}(d(r)))\cdot\mathcal{C}(s) + \mathcal{C}(d(r))\cdot d(\mathcal{C}(s)) + \sum d(\mathcal{C}(d(\mathcal{C}(d(r)))))\cdot d(\mathcal{C}(s))$$

3. As we know by the IH, d^i_r and d^i_s are finite for any $i \geq 0$. Hence, via similar reasoning as above we can conclude that the above is similar to expressions of the form

$$d^1_r\cdot\mathcal{C}(s) + \mathcal{C}(d^0_r)\cdot d^1_s + \sum d^2_r\cdot d^1_s$$

   As all sub-components are finite, we can immediately conclude that $d^1_{t_1}$ is finite.
4. We consider the descendants of $\mathcal{C}(t_2)$ which are of the following form

$$d(\mathcal{C}(r))\cdot\mathcal{C}(d(s)) + \mathcal{C}(r)\cdot d(\mathcal{C}(d(s))) + \sum d(\mathcal{C}(d(\mathcal{C}(r))))\cdot d(\mathcal{C}(d(s)))$$

5. The above is similar to

$$d^1_r\cdot\mathcal{C}(d^0_s) + \mathcal{C}(r)\cdot d^1_r + \sum d^2_r\cdot d^1_s$$

   Thus, we find that $d^1_{t_2}$ is finite.
6. Finally, we observe that shape of descendants of $\mathcal{C}(t_3)$

$$\begin{aligned} &d(\mathcal{C}(d(\mathcal{C}(d(r)))))\cdot\mathcal{C}(d(s)) + \mathcal{C}(d(\mathcal{C}(d(r))))\cdot d(\mathcal{C}(d(s))) \\ &+\sum d(\mathcal{C}(d(\mathcal{C}(d(\mathcal{C}(d(r)))))))\cdot d(\mathcal{C}(d(s))) \end{aligned}$$

7. The above is similar to

$$d^2_r\cdot\mathcal{C}(d^0_s) + \mathcal{C}(d^1_r)\cdot d^1_s + \sum d^3_r\cdot d^1_s$$

8. Then, $d^1_{t_3}$ is finite which concludes the proof for this case.

□

A.12 Similarity is decidable

We turn similarity rules from Figure 3 into term rewriting rules. Roughly, the right-hand side of $\approx$ is replaced by the left-hand side. See Figure 4. Rules (EW) and (EL) are straightforward. Alternatives are normalized into right-associative normal form. See rule (Assoc2). To easily enforce idempotence (rule (Idemp)) we sort terms in a sum according to their term size. See rules (Comm) and (Assoc1). We assume that $|r|$ denotes the size of r. We write $r < s$ iff $|r| < |s|$. Finally, rule (U) removes ϕ in a sum. We assume that the term size of behaviors is such that $|\phi| \leq |r|$ for any behavior r. Thus, the single rule (U) is sufficient.

$$\text{(Comm)}\ \frac{s < r}{r + s \to s + r}$$

$$\text{(Assoc1)}\ \frac{s < r}{r + (s + t) \to s + (r + t)}$$

$$\text{(Assoc2)}\ (r + s) + t \to r + (s + t)$$

$$\text{(Idemp)}\ r + r \to r$$

$$\text{(U)}\ \phi + r \to r \qquad \text{(EW)}\ \begin{array}{ll} \varepsilon \cdot r \to r & \phi \cdot r \to \phi \\ r \cdot \varepsilon \to r & r \cdot \phi \to \phi \\ \varepsilon^* \to \varepsilon & \phi^* \to \varepsilon \\ \mathit{Fork}(\varepsilon) \to \varepsilon & \mathit{Fork}(\phi) \to \phi \end{array} \qquad \text{(EL)}$$

Fig. 4. Normalization

Lemma 31. *The rewrite system in Figure 4 is terminating and confluent.*

Proof. Termination is due to the fact that the he size of the left-hand side term becomes smaller than the size of the right-hand side term. The exceptions are rules (Comm) and (Assoc1-2). However, it is clear that there can only be a finite number of applications of these terms. In case of (Comm) and (Assoc1) we shift smaller terms to the 'left'. In case of (Assoc2), the left part of an alternative becomes smaller.

We show confluence by establishing local confluence. That is, all critical pairs are confluence. Given that the term rewrite system is terminating, we obtain confluence by application of Newman's Lemma.

It is a straightforward exercise to verify that all critical pairs are joinable. For example, note the presence of rule (Assoc2). Thus, the critical pair among (Comm) and (Assoc2) becomes joinable. □

Based on the above result we can build canonical normal forms by exhaustive rule application until we reach a fixpoint. We write $r \to^* t$ to denote that t is term on which no further rewrite rules are applicable. We refer to t as the canonical form of r. It is clear that $r \approx t$.

Lemma 32. *Let r and s be behaviors such that $r \approx s$. Then, we find $r \rightarrowtail^* t_1$ and $s \rightarrowtail^* t_2$ for some t_1 and t_2 where t_1 and t_2 are syntactically equivalent.*

Proof. The result follows by induction over the derivation $r \approx s$. For brevity, we only show some cases.

Suppose

$$\text{(Compatibility)}\ \frac{s_1 \approx s_2}{r[s_1] \approx r[s_2]}$$

By the IH, we find $s_1 \rightarrowtail^* t$, $s_2 \rightarrowtail^* t$ for some t. Hence, $r[s_1] \rightarrowtail^n r[t]$ and $r[s_2] \rightarrowtail^m r[t]$. That is, in finite number of rewrite steps (either n or m), we rewrite $r[s_1]$ into $r[t]$ and $r[s_2]$ into $r[t]$. As every term, e.g. $r[t]$, must have a canonical normal form, the result follows by combing the above rewrite steps and from the fact that the rewrite system is confluent.

Suppose in the last derivation step, we apply

$$\text{(Associativity) } r_1 + (r_2 + r_3) \approx (r_1 + r_2) + r_3$$

Suppose $r_1 + (r_2 + r_3) \rightarrowtail^* t$. Then, $(r_1 + r_2) + r_3 \rightarrowtail^{Assoc2} r_1 + (r_2 + r_3)$ and by confluence we know that $r_1 + (r_2 + r_3)$ and $(r_1 + r_2) + r_3$ share the same canonical normal form. □

Based on the above results we obtain a decidable method to test for similarity for some behaviors r and s. We build the canonical normal forms of r and s. If the canonical normal forms are syntactically equivalent, we find that that r and s are similar. Otherwise, r and s are dissimilar.

A.13 Normal form

Define smart constructors for expressions that implement the similarity rules. We assume a total ordering $<$ on forkable expressions. We further assume that arguments of constructors are already in normal form. That means, that the monomials in every sum are sorted in ascending order and that monomials are bracketed to the right.

$$r \oplus s = \begin{cases} r & r = s \vee s = \phi \\ s & r = \phi \\ r' \oplus (r'' \oplus s) & r = r' + r'' \\ (r \oplus s') + s'' & r \neq r' + r'', s = s' + s'', r < \min s'' \\ s + (r \oplus s'') & r \neq r' + r'', s = s' + s'', r \geq \min s'' \\ r + s & r \neq r' + r'', s \neq s' + s'', r < s \\ s + r & r \neq r' + r'', s \neq s' + s'', r > s \end{cases}$$

$$r \odot s = \begin{cases} \phi & r = \phi \vee s = \phi \\ r & s = \varepsilon \\ s & r = \varepsilon \\ r' \odot (r'' \odot s) & r = r' \cdot r'' \\ s \cdot r & r = \mathit{Fork}(r'), s = \mathit{Fork}(s'), s' < r' \\ \mathit{Fork}(s') \cdot (r \odot s'') & r = \mathit{Fork}(r'), s = \mathit{Fork}(s') \cdot s'', s' < r' \\ r \cdot s & r \neq r' \cdot r'' \end{cases}$$

$$r^{\circledast} = \begin{cases} \varepsilon & r = \phi \vee r = \varepsilon \\ r^* & \text{otherwise} \end{cases}$$

$$F(r) = \begin{cases} \phi & r = \phi \\ \varepsilon & r = \varepsilon \\ \mathit{Fork}(r) & \text{otherwise} \end{cases}$$

This normal form exploits the following lemma, which is straightforward to prove from the semantics.

Lemma 33. *$\mathit{Fork}(r) \cdot \mathit{Fork}(s) \equiv \mathit{Fork}(s) \cdot \mathit{Fork}(r)$, for all r and s.*

B Nullable and Emptiness Test

Definition 34 (Nullable Test).

$$\begin{aligned} \mathcal{N}(\phi) &= \mathit{False} & \mathcal{N}(\varepsilon) &= \mathit{True} & \mathcal{N}(x) &= \mathit{True} \\ \mathcal{N}(r+s) &= \mathcal{N}(r) \vee \mathcal{N}(s) & \mathcal{N}(r \cdot s) &= \mathcal{N}(r) \wedge \mathcal{N}(s) & \mathcal{N}(r^*) &= \mathit{True} \\ \mathcal{N}(\mathit{Fork}(r)) &= \mathcal{N}(r) \end{aligned}$$

Lemma 35. *Let r be a behavior. Then, $\varepsilon \in L(r)$ iff $\mathcal{N}(r)$.*

Definition 36 (Emptiness Test).

$$\begin{aligned} \Phi(\phi) &= \mathit{True} & \Phi(\varepsilon) &= \mathit{False} & \Phi(x) &= \mathit{False} \\ \Phi(r+s) &= \Phi(r) \wedge \Phi(s) & \Phi(r \cdot s) &= \Phi(r) \vee \Phi(s) & \Phi(r^*) &= \mathit{False} \\ \Phi(\mathit{Fork}(r)) &= \Phi(r) \end{aligned}$$

Lemma 37. *Let r be a behavior. Then, $L(r) = \{\}$ iff $\Phi(r)$.*

Proof. Fact: $L(r, \{\}) = \{\}$. We verify $L(r, K) \neq \{\}$ iff $\neg\Phi(r)$. The proof proceeds by straightforward induction. □