Forkable Regular Expressions

Martin Sulzmann[1] Peter Thiemann[2]

[1]University of Freiburg

[2]Karlsruhe University of Applied Sciences

LATA 2016, Prague

Objective

Analysis of concurrent programs

- Threads + primitive events.
- Abstraction in terms of event traces.
- Run-time verification $\Rightarrow$ word problem.
- Static analysis $\Rightarrow$ inclusion problem.

$$\mathsf{while}(e)\{\mathsf{thread}\ x; \mathsf{thread}\ y\}$$

Event traces:

- $x \cdot y \cdot x \cdot y \cdot x \cdot y \ldots$
- $x \cdot x \cdot x \cdot y \cdot y \cdot y \ldots$
- ...

Earlier Works: Garg and Ragunath

Concurrent regular expressions

- Iterated shuffle $r^{\|}$ (a.k.a. shuffle closure).
  - $x \cdot y \| z = x \cdot y \cdot z + x \cdot z \cdot y + z \cdot x \cdot y$.
  - $r^{\|} = \epsilon + r\|r + r\|r\|r + ...$
- Syntactic connection to program largely lost.

  $\text{while}(e)\{\text{thread } x; \text{thread } y\}$ versus $(x \cdot y + y \cdot x)^{\|}$

Earlier Works: Nielson and Nielson

Type and behavior reconstruction

- Type and effect system based on forkable behaviors.
- Lack of semantic interpretation.

Forkable Regular Expressions

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

Forkable Regular Expressions

Natural abstraction of concurrent programs

while$(e)\{$thread $x;$ thread $y\}$ $\Rightarrow$ $(Fork(x) \cdot Fork(y))^*$

Semantics

Mapping L from forkable expressions to languages.

As expressive as shuffle expressions

$$r \| s \;=\; Fork(r) \cdot s \quad r^{\|} \;=\; Fork(r)^*$$

Analysis method

Establish the notion of Brzozowski's derivatives.

Language Semantics?

Challenge

- $L((Fork(x) \cdot y)^*) = \{\epsilon, x \cdot y, y \cdot x, y \cdot y \cdot x \cdot x, ...\}$

Language Semantics?

Challenge

- $L((Fork(x) \cdot y)^*) = \{\epsilon, x \cdot y, y \cdot x, y \cdot y \cdot x \cdot x, ...\}$
- Compositional semantics?

$$L(Fork(x) \cdot y) = L(Fork(x)) \; ? \; L(y)$$

Language Semantics?

Challenge

- $L((Fork(x) \cdot y)^*) = \{\epsilon, x \cdot y, y \cdot x, y \cdot y \cdot x \cdot x, ...\}$
- Compositional semantics?

$$L(Fork(x) \cdot y) = L(Fork(x)) \boxed{?} L(y)$$

$$L(Fork(x) \cdot y) \neq L(Fork(x)) \cdot L(y)$$
$$L(Fork(x) \cdot y) = L(Fork(x)) \| L(y)$$

Solution

$L(r, K)$ for Fork and Kleene star

- Parameterize by a continuation language K.
- Fork interleaves with continuation:

$$L(Fork(r), K) = L(r) \| K$$

- Kleene star via fixpoint:

$$L(r^*, K) = \mu X. L(r, X) \cup K$$

Solution

$L(r, K)$ for Fork and Kleene star

- Parameterize by a continuation language K.
- Fork interleaves with continuation:

$$L(Fork(r), K) = L(r) \| K$$

- Kleene star via fixpoint:

$$L(r^*, K) = \mu X.L(r, X) \cup K$$

$$\begin{aligned}
&L((Fork(x) \cdot y)^*, \{\epsilon\}) = \{\epsilon\} \cup X_1 \cup X_2 \cup \ldots \\
&X_1 = L(Fork(x) \cdot y, \{\epsilon\}) = \{x \cdot y, y \cdot x\} \\
&X_2 = L(Fork(x) \cdot y, X_1) = \{x \cdot y \cdot x \cdot y, y \cdot x \cdot x \cdot y, \ldots\}
\end{aligned}$$

Solution (2)

Remaining cases $L(r, K)$

$$
\begin{aligned}
L(\phi, K) &= \emptyset \\
L(\varepsilon, K) &= K \\
L(x, K) &= \{x\} \cdot K
\end{aligned}
\qquad
\begin{aligned}
L(r + s, K) &= L(r, K) \cup L(s, K) \\
L(r \cdot s, K) &= L(r, L(s, K))
\end{aligned}
$$

Analysis of Concurrent Programs

Approach

- Given program p and specification s.
- Run-time verification (word problem):
  - Check traces w resulting from running p against s.
  - $w \in L(s)$.
- Static analysis (inclusion problem):
  - p's traces included in s.
  - Abstraction of $p \Longrightarrow r$.
  - Verify $L(r) \subseteq L(s)$.

Word Problem $w \in L(s)$

Derivative-based decision procedure

- Build derivative $d_x(r)$ by taking away the leading symbol x.
- $x \cdot w \in L(r)$ iff $w \in L(d_x(r))$.
- For word $w = x_1 \cdot \ldots \cdot x_n$:
  - Build $d_w(r)$.
  - Check if nullable, i.e. $\epsilon \in L(d_w(r))$.

Brzozowski's Derivatives

$d_x(r)$

$$
\begin{array}{lcl}
d_x(\phi) & = & \phi \\
d_x(\varepsilon) & = & \phi \\
d_x(y) & = & \begin{cases} \varepsilon & \text{if } x = y \\ \phi & \text{otherwise} \end{cases} \\
d_x(r+s) & = & d_x(r) + d_x(s) \\
d_x(r^*) & = & d_x(r) \cdot r^* \\
d_x(r \cdot s) & = & \begin{cases} d_x(r) \cdot s & \text{if } \epsilon \notin L(r) \\ d_x(r) \cdot s + d_x(s) & \text{otherwise} \end{cases}
\end{array}
$$

Extension to Forkable Expressions

Challenge

- $d_x(Fork(r) \cdot s) = ???$
- We would expect

$$d_x(Fork(r) \cdot s) = d_x(Fork(r)) \cdot s + Fork(r) \cdot d_x(s)$$

- How to distinguish 'sequential' from 'concurrent' parts?

Observations

Concurrent and sequential parts

- $r = \mathcal{C}(r) + \mathcal{S}(r)$.
- $\mathcal{S}(r)$ what happens next.
- $\mathcal{C}(r)$ what happens eventually and concurrently.
- $\mathcal{C}(r)$, $\mathcal{S}(r)$ can be computed syntactically.

$$\underbrace{(Fork(x)+y)^*}_{r}$$
$$=$$
$$\underbrace{(Fork(x))^*}_{\mathcal{C}(r)} + \underbrace{(Fork(x))^* \cdot y \cdot (Fork(x)+y)^*}_{\mathcal{S}(r)}$$

Concurrent Parts

$\mathcal{C}(r)$

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

Sequential Parts

$\mathcal{S}(r)$

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

Derivatives for Forkable Expressions

Adjust $r \cdot s$ and include $Fork(r)$

$$
\begin{array}{rcl}
d_x(r \cdot s) & = & d_x(r) \cdot s + \mathcal{C}(r) \cdot d_x(s) \\
\\
d_x(Fork(r)) & = & Fork(d_x(r))
\end{array}
$$

- $\epsilon \in L(r)$ iff $\epsilon \in L(\mathcal{C}(r))$.
- $x \cdot w \in L(r)$ iff $w \in L(d_x(r))$.
- See paper for formal results.

Inclusion Problem $L(r) \subseteq L(s)$

Derivative-based proof method

- Grabmeyer's coinductive proof systems $r \leq s$
  - Reduce $r \leq s$ to $d_x(r) \leq d_x(s)$.

Inclusion Problem $L(r) \subseteq L(s)$

Derivative-based proof method

- Grabmeyer's coinductive proof systems $r \leq s$
  - Reduce $r \leq s$ to $d_x(r) \leq d_x(s)$.
  - Decidable if set of dissimilar derivatives is finite.

$$(\text{Idem}) \;\; r + r \approx r \qquad (\text{Comm}) \;\; r + s \approx s + r$$

$$(\text{Assoc}) \;\; (r + s) + t \approx r + (s + t)$$

$$(\text{Elim1}) \;\; \epsilon \cdot r \approx r \qquad (\text{Elim2}) \;\; \phi \cdot r \approx \phi$$

- $x^* \leq x^* \xrightarrow{x} \epsilon \cdot x^* \leq \epsilon \cdot x^* \xrightarrow{\approx} x^* \leq x^*$

Issue

Loss of finiteness of dissimilar derivatives

- Take $r = (Fork(x \cdot y))^*$:

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

- Forkable expressions non-regular!

Well-Behaved Forkable Expressions

No non-trivial concurrent behavior under Kleene star

- For all subexpressions r^*: $\forall w. L(\mathcal{C}(d_w(r))) \subseteq \{\epsilon\}$.
- Guarantees that set of dissmilar derivatives is finite.
- Cannot be weakened to words w of a fixed length.
- See paper for formal result + examples.

Conclusion

- Forkable expressions to describe concurrent programs.
  - Expressiveness?
- Decidable word problem.
- Decidable inclusion problem for well-behaved expressions.
  - Finite approximation of derivatives of ill-behaved expressions?

Expressiveness

Context-free

$Fork(x \cdot y + y \cdot x)^*$ is the shuffle closure $\{x \cdot y, y \cdot x\}$ 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.

Context-sensitive

$Fork(x \cdot y \cdot z)^* = (x \cdot y \cdot z)^{\|}$ where $Fork(x \cdot y \cdot z)^* \cap (x^* \cdot y^* \cdot z^*) = \{x^n \cdot y^n \cdot z^n\}$ which is not context-free. So, $Fork(x \cdot y \cdot z)^*$ cannot be context-free either.