Derivatives in Program Analysis

Albert-Ludwigs-Universität Freiburg

Peter Thiemann[1] Martin Sulzmann[2]

[1]University of Freiburg

[2]Karlsruhe University of Applied Sciences

16 Dec 2015

Setting

Concurrent ML (CML)

- higher-order programming language
- concurrency (dynamic process creation: `fork`)
- dynamically created, typed channels t CHAN
- high-level synchronization primitives

Objective

Analyze communication behavior of CML programs

- adherence to protocols
- deadlock detection

Objective

Analyze communication behavior of CML programs

- adherence to protocols
- deadlock detection

Static and dynamic analysis

Starting point

Effect system by Nielson and Nielson [POPL 1994]

- abstracts communication behavior to (sort of) regular expression
- alphabet = events
  - $r!t$ send value of type t across channel r
  - $r?t$ receive value of type t across channel r

Syntax of effects [NN94]

$$\begin{aligned} b ::= \quad & \varepsilon \mid r!t \mid r?t \mid t \text{ CHAN } r \mid \\ & \text{FORK } b \mid b \cdot b \mid b + b \mid \text{REC } \beta.b \mid \beta \end{aligned}$$

Example

Behavior

$$\mathsf{REC}\ \beta.t\ \mathsf{CHAN}\ r + \mathsf{FORK}(r?t; \beta)$$

Example

Behavior

$$\text{REC } \beta.t \text{ CHAN } r + \text{FORK}(r?t; \beta)$$

Term

```
e = choose [send (ch1, 7),
            wrap (receive ch2, fn x => 1)]
```

$$\begin{array}{l} \texttt{ch1} : \mathit{int} \text{ CHAN } r_1, \texttt{ch2} : \mathit{bool} \text{ CHAN } r_2 \\ \vdash \texttt{e} : \mathit{int} \text{ COM } (r_1!\mathit{int} + r_2?\mathit{bool}) \end{array}$$

Question

Given

- a typed term $e : t; b_{term}$
- a behavior specification b_{spec}

Question

Given

- a typed term $e : t; b_{term}$
- a behavior specification b_{spec}

Does the term's behavior adhere to the specification?

- statically: $b_{term} \sqsubseteq b_{spec}$?
- dynamically: is a trace π of e admissible for b_{spec}?

Approach

Turn into a language problem

- define $[\![b]\!]$ as a set of traces
- define $b_{term} \sqsubseteq b_{spec}$ semantically by $[\![b_{term}]\!] \subseteq [\![b_{spec}]\!]$
- define "π admissible for b_{spec}" by $\pi \in [\![b_{spec}]\!]$
- find decision procedures for inclusion problem and word problem

Behavior $\rightsquigarrow$ set of traces

Start with a simpler set of behaviors ...

$$b ::= \varepsilon \mid x \mid b \cdot b \mid b + b \mid b^*$$

- loops instead of recursion; no FORK
- regular expressions $\rightsquigarrow$ regular trace languages

Behavior ⇝ set of traces

Start with a simpler set of behaviors ...

$$b ::= \varepsilon \mid x \mid b \cdot b \mid b + b \mid b^*$$

- loops instead of recursion; no FORK
- regular expressions ⇝ regular trace languages

Compositional definition for trace language

$$\begin{aligned}
[\![\varepsilon]\!] &= \{\varepsilon\} \\
[\![x]\!] &= \{x\} \\
[\![b_1 \cdot b_2]\!] &= [\![b_1]\!] \cdot [\![b_2]\!] \\
[\![b_1 + b_2]\!] &= [\![b_1]\!] \cup [\![b_2]\!] \\
[\![b^*]\!] &= \mu X.\{\varepsilon\} \cup [\![b]\!] \cdot X
\end{aligned}$$

Behavior $\rightsquigarrow$ set of traces, II

Adding FORK

- FORK b starts an independent thread, which generates events according to its behavior b
- events of forked thread occur *interleaved* with events of main thread: use *asynchronous shuffle operator* $\|$

$$
\begin{aligned}
[\![\mathrm{FORK}(x) \cdot (y \cdot z)]\!] = \{xyz, yxz, yzx\} &= [\![x]\!] \| [\![y \cdot z]\!] \\
&= [\![(\mathrm{FORK}(x) \cdot y) \cdot z]\!] \\
&= \underbrace{[\![(\mathrm{FORK}(x) \cdot y)]\!]}_{=\{xy, yx\}} \;{\color{red}??}\; [\![z]\!]
\end{aligned}
$$

Behavior $\rightsquigarrow$ set of traces, II

Adding FORK

- FORK b starts an independent thread, which generates events according to its behavior b
- events of forked thread occur *interleaved* with events of main thread: use *asynchronous shuffle operator* $\|$

$$
\begin{aligned}
[\![\mathsf{FORK}(x) \cdot (y \cdot z)]\!] &= \{xyz, yxz, yzx\} = [\![x]\!] \| [\![y \cdot z]\!] \\
&= [\![(\mathsf{FORK}(x) \cdot y) \cdot z]\!] \\
&= \underbrace{[\![(\mathsf{FORK}(x) \cdot y)]\!]}_{=\{xy, yx\}} \; ?? \; [\![z]\!]
\end{aligned}
$$

- no obvious compositional description

Behavior $\rightsquigarrow$ set of traces, III

Solution

- Parameterize language definition by a *continuation language*
- FORK interleaves with the continuation language

Behavior $\rightsquigarrow$ set of traces, IV

Semantics of behaviors, $K \subseteq \Sigma^*$

$$
\begin{aligned}
[\![\varepsilon]\!]K &= K \\
[\![x]\!]K &= \{x\} \cdot K \\
[\![b_1 \cdot b_2]\!]K &= [\![b_1]\!]([\![b_2]\!]K) \\
[\![b_1 + b_2]\!]K &= [\![b_1]\!]K \cup [\![b_2]\!]K \\
[\![b^*]\!]K &= \mu X.K \cup [\![b]\!]X \\
[\![\mathrm{FORK}\ b]\!]K &= K \| [\![b]\!]\{\varepsilon\}
\end{aligned}
$$

Behavior $\rightsquigarrow$ set of traces, IV

Semantics of behaviors, $K \subseteq \Sigma^*$

$$
\begin{aligned}
[\![\varepsilon]\!]K &= K \\
[\![x]\!]K &= \{x\} \cdot K \\
[\![b_1 \cdot b_2]\!]K &= [\![b_1]\!]([\![b_2]\!]K) \\
[\![b_1 + b_2]\!]K &= [\![b_1]\!]K \cup [\![b_2]\!]K \\
[\![b^*]\!]K &= \mu X . K \cup [\![b]\!]X \\
[\![\mathrm{FORK}\ b]\!]K &= K \| [\![b]\!]\{\varepsilon\}
\end{aligned}
$$

Theorem

With this definition, forkable expressions form a Kleene algebra.

Properties

Revisiting the example

$$
\begin{aligned}
[\![\mathrm{FORK}(x) \cdot (y \cdot z)]\!] K &= [\![\mathrm{FORK}(x)]\!]([\![y \cdot z]\!] K) \\
&= \{x\} \| [\![y \cdot z]\!] K \\
&= \{x\} \| [\![y]\!]([\![z]\!] K) \\
&= [\![\mathrm{FORK}(x)]\!]([\![y]\!]([\![z]\!] K)) \\
&= [\![\mathrm{FORK}(x) \cdot y]\!]([\![z]\!] K) \\
&= [\![(\mathrm{FORK}(x) \cdot y) \cdot z]\!] K
\end{aligned}
$$

Inclusion problem

Is $[\![b_1]\!]\{\varepsilon\} \subseteq [\![b_2]\!]\{\varepsilon\}$ decidable?

Inclusion problem

Is $[\![b_1]\!]\{\varepsilon\} \subseteq [\![b_2]\!]\{\varepsilon\}$ decidable?

Unfortunately ...

- Consider

$$\begin{aligned} L &= [\![(\text{FORK } (xyz))^*]\!]\{\varepsilon\} \\ &= \mu X.\{\varepsilon\} \cup \{xyz\} \| X \\ &= \{\varepsilon\} \cup \{xyz\} \cup \{xyz\} \| \{xyz\} \cup \ldots \\ &= \{xyz\}^{\|} \end{aligned}$$

  the iterated shuffle (shuffle closure)
- Clearly $L \cap x^* y^* z^* = \{x^n y^n z^n\}$ which is not even context-free, so L cannot be context-free, either
- inclusion is undecidable

Word problem

Recall Brzozowski: Derivatives for regular expressions

$$
\begin{aligned}
d_y(\varepsilon) &= \emptyset & N(\varepsilon) &= \varepsilon \\
d_y(x) &= \begin{cases} \varepsilon & x = y \\ \emptyset & x \neq y \end{cases} & N(x) &= \emptyset \\
d_y(b_1 \cdot b_2) &= d_y(b_1) \cdot b_2 & N(b_1 \cdot b_2) &= N(b_1) \cdot N(b_2) \\
&\quad + N(b_1) \cdot d_y(b_2) & & \\
d_y(b_1 + b_2) &= d_y(b_1) \cup d_y(b_2) & N(b_1 + b_2) &= N(b_1) \cup N(b_2) \\
d_y(b^*) &= d_y(b) \cdot b^* & N(b^*) &= \varepsilon
\end{aligned}
$$

Correctness

$$[\![d_y(b)]\!] = y^{-1}[\![b]\!] = \{ w \mid yw \in [\![b]\!] \}$$

Derivatives for forkable expressions?

Two changes wrt regular expressions

$$d_y(\mathrm{FORK}\ b) = \mathrm{FORK}\ (d_y(b))$$
$$d_y(b_1 \cdot b_2) = d_y(b_1) \cdot b_2 + C(b_1) \cdot d_y(b_2)$$

- $C(b)$ is the *concurrent part* of b
- intuition: there are two possibilities
  1. the derivative takes the first symbol of b_1 or
  2. the derivative takes the first symbol of b_2 if b_1 can somehow be skipped; for instance if there is a path through b_1 that consumes no symbols, but may fork new processes

Concurrent and sequential part of a forkable expression
Definition

- $C(b)$ concurrent part
- $S(b)$ sequential part

$$
\begin{aligned}
C(\varepsilon) &= \varepsilon & S(\varepsilon) &= \emptyset \\
C(x) &= \emptyset & S(x) &= x \\
C(b_1 \cdot b_2) &= C(b_1) \cdot C(b_2) & S(b_1 \cdot b_2) &= S(b_1) \cdot b_2 + C(b_1) \cdot S(b_2) \\
C(b_1 + b_2) &= C(b_1) + C(b_2) & S(b_1 + b_2) &= S(b_1) + S(b_2) \\
C(b^*) &= C(b)^* & S(b^*) &= C(b)^* \cdot S(b) \cdot b^* \\
C(\mathrm{FORK}\ b) &= \mathrm{FORK}\ b & S(\mathrm{FORK}\ b) &= \emptyset
\end{aligned}
$$

Concurrent and sequential part of a forkable expression

Properties

- $b \equiv C(b) + S(b)$
- $C(C(b)) = C(b)$
- $C(S(b)) = \emptyset$
- $S(C(b)) = \emptyset$
- $S(S(b)) = S(b)$

Correctness of derivatives

Theorem

$$[\![d_y(b)]\!]\{\varepsilon\} = y^{-1}[\![b]\!]\{\varepsilon\}$$

Correctness of derivatives

Theorem

$$[\![d_y(b)]\!]\{\varepsilon\} = y^{-1}[\![b]\!]\{\varepsilon\}$$

Proof

By induction using the generalized hypothesis

$$\forall b.\ \forall K.\ [\![d_y(b)]\!]K \cup [\![C(b)]\!]\{\varepsilon\} \| (y^{-1}K) = y^{-1}[\![b]\!]K$$

Back to the word problem

UNI FREIBURG

Decide the word problem by ...

- the derivative $d_y(b)$ is effectively computable
- $\varepsilon \in [\![b]\!]\{\varepsilon\}$ is effectively checkable
- to test $w \in [\![b]\!]\{\varepsilon\}$ check $\varepsilon \in [\![d_w(b)]\!]\{\varepsilon\}$

Back to the inclusion problem

Iterated derivatives

$$
\begin{aligned}
& d_{x\ldots}((\mathsf{FORK}(xy))^*) \\
&= d_{x\ldots}(d_x(\mathsf{FORK}(xy)) \cdot (\mathsf{FORK}(xy))^*) \\
&= d_{x\ldots}(\mathsf{FORK}(y) \cdot (\mathsf{FORK}(xy))^*) \\
&= \underbrace{d_{x\ldots}(\mathsf{FORK}(y))}_{=\emptyset} + \underbrace{C(\mathsf{FORK}(y))}_{=\mathsf{FORK}(y)} \cdot d_{x\ldots}((\mathsf{FORK}(xy))^*) \\
&= \mathsf{FORK}(y) \cdot (d_{x\ldots}(\mathsf{FORK}(y) \cdot (\mathsf{FORK}(xy))^*)) \\
&= \mathsf{FORK}(y) \cdots \mathsf{FORK}(y) \cdot d_{x\ldots}((\mathsf{FORK}(xy))^*)
\end{aligned}
$$

Back to the inclusion problem, II

Observation

- The size of an iterated derivative grows without bound.
- Inclusion tests that work by constructing a (bi)simulation do not work.

Back to the inclusion problem, II

Observation

- The size of an iterated derivative grows without bound.
- Inclusion tests that work by constructing a (bi)simulation do not work.

Definition: Well-behavior

b_0 is *well-behaved* if all subterms of the form b^* have the property that, for all $w \in \Sigma^*$, $C(d_w(b)) \leq \varepsilon$.

Back to the inclusion problem, II

Observation

- The size of an iterated derivative grows without bound.
- Inclusion tests that work by constructing a (bi)simulation do not work.

Definition: Well-behavior

b_0 is *well-behaved* if all subterms of the form b^* have the property that, for all $w \in \Sigma^*$, $C(d_w(b)) \leq \varepsilon$.

Lemma

If b is fork=free, then , for all $w \in \Sigma^*$, $C(d_w(b)) \leq \varepsilon$.

A decidable case for inclusion

Definition

Let $\sharp d(b)$ be the number of dissimilar iterated derivatives of b.

A decidable case for inclusion

Definition

Let $\sharp d(b)$ be the number of dissimilar iterated derivatives of b.

Theorem

Let b be well-behaved. Then $\sharp d(b) < \infty$.

A decidable case for inclusion

Definition

Let $\sharp d(b)$ be the number of dissimilar iterated derivatives of b.

Theorem

Let b be well-behaved. Then $\sharp d(b) < \infty$.

Corollary

If b_1 and b_2 are well-behaved, then "$[\![b_1]\!] \subseteq [\![b_2]\!]$?" is decidable.

A decidable case for inclusion

Definition

Let $\sharp d(b)$ be the number of dissimilar iterated derivatives of b.

Theorem

Let b be well-behaved. Then $\sharp d(b) < \infty$.

Corollary

If b_1 and b_2 are well-behaved, then "$[\![b_1]\!] \subseteq [\![b_2]\!]$?" is decidable.

Proof

Since $\sharp d(b_i) < \infty$, we can attempt to construct a bisimulation for $b_1 + b_2 \sim b_2$. This construction stops after finitely many steps.

Open questions

Power of forkable expressions

- Forkable expressions subsume regular shuffle expressions.
- The reverse direction is not known.

Open questions

Power of forkable expressions

- Forkable expressions subsume regular shuffle expressions.
- The reverse direction is not known.

Complexity of word problem?

Open questions

Power of forkable expressions

- Forkable expressions subsume regular shuffle expressions.
- The reverse direction is not known.

Complexity of word problem?

Adding REC

What remains decidable, when we consider the full behavior language of [NN94], e.g, add general recursion?

Open questions

Power of forkable expressions

- Forkable expressions subsume regular shuffle expressions.
- The reverse direction is not known.

Complexity of word problem?

Adding REC

What remains decidable, when we consider the full behavior language of [NN94], e.g, add general recursion?

Synchronizing shuffle

If there are, e.g., matching events like $r!t$ and $r?t$, we want to resolve to event $[r]$. Can we define derivatives for this case?

Conclusion

- Towards a compositional trace semantics for CML
- New flavor of *forkable* regular expressions to describe effect traces
- Generated language is context-sensitive (conjecture: proper subclass)
- Decidable word problem $\Rightarrow$ dynamic analysis possible
- Inclusion decidable in restricted cases $\Rightarrow$ static analysis possible; approximation?

See upcoming paper at LATA2016

`http://arxiv.org/abs/1510.07293`