Derivatives in Program Analysis

Peter Thiemann\textsuperscript{1}  \quad Martin Sulzmann\textsuperscript{2}

\textsuperscript{1}University of Freiburg
\textsuperscript{2}Karlsruhe University of Applied Sciences

16 Dec 2015
**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
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]

$$b ::= \varepsilon \mid r!t \mid r?t \mid t \text{ CHAN } r \mid$$
$$\text{FORK } b \mid b \cdot b \mid b + b \mid \text{REC } \beta.b \mid \beta$$
Example

Behavior

\[ \text{REC } \beta.t \text{ Chan } r + \text{Fork}(r?t; \beta) \]
Example

Behavior

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

Term

\[ e = \text{choose } [\text{send } (\text{ch1}, 7), \text{wrap } (\text{receive } \text{ch2}, \text{fn } x \Rightarrow 1)] \]

\[ \text{ch1} : \text{int CHAN } r_1, \text{ch2} : \text{bool CHAN } r_2 \]

\[ \vdash e : \text{int COM } (r_1!\text{int} + r_2?\text{bool}) \]
Question

Given

- a typed term \( e : t; \ b_{term} \)
- a behavior specification \( \ b_{spec} \)
Question

Given

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

Does the term’s behavior adhere to the specification?

- statically: $b_{\text{term}} \subseteq b_{\text{spec}}$?
- dynamically: is a trace $\pi$ of $e$ admissible for $b_{\text{spec}}$?
Approach

- Turn into a language problem
  - define $[b]$ as a set of traces
  - define $b_{term} \subseteq b_{spec}$ semantically by $[b_{term}] \subseteq [b_{spec}]$
  - define “$\pi$ admissible for $b_{spec}$” by $\pi \in [b_{spec}]$
  - find decision procedures for inclusion problem and word problem
Behavior $\leadsto$ set of traces

Start with a simpler set of behaviors . . .

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

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

Compositional definition for trace language

$$[\varepsilon] = \{\varepsilon\}$$
$$[x] = \{x\}$$
$$[b_1 \cdot b_2] = [b_1] \cdot [b_2]$$
$$[b_1 + b_2] = [b_1] \cup [b_2]$$
$$[b^*] = \mu X.\{\varepsilon\} \cup [b] \cdot X$$
Behavior $\leadsto$ 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* $\parallel$

\[
\llbracket \text{FORK}(x) \cdot (y \cdot z) \rrbracket = \{xyz, yxz, yzx\} = \llbracket x \rrbracket \parallel \llbracket y \cdot z \rrbracket \\
= \llbracket (\text{FORK}(x) \cdot y) \cdot z \rrbracket \\
= \llbracket (\text{FORK}(x) \cdot y) \rrbracket \parallel \llbracket z \rrbracket \\
= \{xy, yx\}
\]
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* $\parallel$

\[
\llbracket \text{FORK}(x) \cdot (y \cdot z) \rrbracket = \{xyz, yxz, yzx\} = \llbracket x \rrbracket \parallel \llbracket y \cdot z \rrbracket = \llbracket (\text{FORK}(x) \cdot y) \cdot z \rrbracket = \llbracket (\text{FORK}(x) \cdot y) \rrbracket \parallel \llbracket z \rrbracket = \{xy, yx\}
\]

- no obvious compositional description
Behavior $\leadsto$ set of traces, III

Solution

- Parameterize language definition by a *continuation language*
- FORK interleaves with the continuation language
Semantics of behaviors, $K \subseteq \Sigma^*$

\[
\begin{align*}
\llbracket \varepsilon \rrbracket K &= K \\
\llbracket \mathit{x} \rrbracket K &= \{ \mathit{x} \} \cdot K \\
\llbracket b_1 \cdot b_2 \rrbracket K &= \llbracket b_1 \rrbracket (\llbracket b_2 \rrbracket K) \\
\llbracket b_1 + b_2 \rrbracket K &= \llbracket b_1 \rrbracket K \cup \llbracket b_2 \rrbracket K \\
\llbracket b^* \rrbracket K &= \mu X.K \cup [b]X \\
\llbracket \text{FORK } b \rrbracket K &= K \parallel [b]\{\varepsilon\}
\end{align*}
\]
Behavior $\leadsto$ set of traces, IV

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

$$
\begin{align*}
\llbracket \varepsilon \rrbracket K &= K \\
\llbracket x \rrbracket K &= \{x\} \cdot K \\
\llbracket b_1 \cdot b_2 \rrbracket K &= \llbracket b_1 \rrbracket (\llbracket b_2 \rrbracket K) \\
\llbracket b_1 + b_2 \rrbracket K &= \llbracket b_1 \rrbracket K \cup \llbracket b_2 \rrbracket K \\
\llbracket b^* \rrbracket K &= \mu X. K \cup \llbracket b \rrbracket X \\
\llbracket \text{FORK } b \rrbracket K &= K \| \llbracket b \rrbracket \{\varepsilon\}
\end{align*}
$$

Theorem

With this definition, forkable expressions form a Kleene algebra.
Revisiting the example

\[
\begin{align*}
\lbrack \text{FORK}(x) \cdot (y \cdot z) \rbrack K &= \lbrack \text{FORK}(x) \rbrack (\lbrack y \cdot z \rbrack K) \\
&= \{x\} \parallel \lbrack y \cdot z \rbrack K \\
&= \{x\} \parallel \lbrack y \rbrack (\lbrack z \rbrack K) \\
&= \lbrack \text{FORK}(x) \rbrack (\lbrack y \rbrack (\lbrack z \rbrack K)) \\
&= \lbrack \text{FORK}(x) \cdot y \rbrack (\lbrack z \rbrack K) \\
&= \lbrack (\text{FORK}(x) \cdot y) \cdot z \rbrack K
\end{align*}
\]
Inclusion problem

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

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

Unfortunately . . .

- Consider

\[ L = [[(\text{FORK } (xyz))^\ast]\{\varepsilon\}] \]
\[ = \mu X.\{\varepsilon\} \cup \{xyz\}\|X \]
\[ = \{\varepsilon\} \cup \{xyz\} \cup \{xyz\}\|\{xyz\} \cup \ldots \]
\[ = \{xyz\}\| \]

the iterated shuffle (shuffle closure)

- Clearly $L \cap x^n y^n z^n = \{x^n y^n z^n\}$ which is not even context-free, so $L$ cannot be context-free, either

- inclusion is undecidable
Recall Brzozowski: Derivatives for regular expressions

\[

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

Correctness

\[
[d_y(b)] = y^{-1}[b] = \{w \mid yw \in [b]\}
\]
Derivatives for forkable expressions?

Two changes wrt regular expressions

\[ d_y(\text{FORK } b) = \text{FORK } (d_y(b)) \]
\[ d_y(b_1 \cdot b_2) = d_y(b_1) \cdot b_2 + C(b_1) \cdot d_y(b_2) \]

- \( C(b) \) is the \textit{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

- $C(\varepsilon) = \varepsilon$  
- $S(\varepsilon) = \emptyset$
- $C(x) = \emptyset$  
- $S(x) = x$
- $C(b_1 \cdot b_2) = C(b_1) \cdot C(b_2)$  
- $S(b_1 \cdot b_2) = S(b_1) \cdot b_2 + C(b_1) \cdot S(b_2)$
- $C(b_1 + b_2) = C(b_1) + C(b_2)$  
- $S(b_1 + b_2) = S(b_1) + S(b_2)$
- $C(b^*) = C(b)^*$  
- $S(b^*) = C(b)^* \cdot S(b) \cdot b^*$
- $C(\text{FORK } b) = \text{FORK } b$  
- $S(\text{FORK } b) = \emptyset$
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

$$[dy(b)]\{\varepsilon\} = y^{-1}[b]\{\varepsilon\}$$
Correctness of derivatives

**Theorem**

\[ \llbracket d_y(b) \rrbracket \{\varepsilon\} = y^{-1} \llbracket b \rrbracket \{\varepsilon\} \]

**Proof**

By induction using the generalized hypothesis

\[ \forall b. \forall K. \llbracket d_y(b) \rrbracket K \cup \llbracket C(b) \rrbracket \{\varepsilon\} \parallel (y^{-1} K) = y^{-1} \llbracket b \rrbracket K \]
Back to the word problem

Decide the word problem by . . .

- the derivative $d_y(b)$ is effectively computable
- $\varepsilon \in \llbracket b \rrbracket \{\varepsilon\}$ is effectively checkable
- to test $w \in \llbracket b \rrbracket \{\varepsilon\}$ check $\varepsilon \in \llbracket d_w(b) \rrbracket \{\varepsilon\}$
Back to the inclusion problem

Iterated derivatives

\[ d_{x\ldots}((\text{FORK}(xy))^*) \]
\[ = d_{x\ldots}(d_x(\text{FORK}(xy)) \cdot (\text{FORK}(xy))^*) \]
\[ = d_{x\ldots}(\text{FORK}(y) \cdot (\text{FORK}(xy))^*) \]
\[ = d_{x\ldots}(\text{FORK}(y)) + C(\text{FORK}(y)) \cdot d_{x\ldots}((\text{FORK}(xy))^*) \]
\[ = \emptyset \]
\[ = \text{FORK}(y) \]
\[ = \text{FORK}(y) \cdot (d_{x\ldots}(\text{FORK}(y) \cdot (\text{FORK}(xy))^*)) \]
\[ = \text{FORK}(y) \cdot \ldots \text{FORK}(y) \cdot d_{x\ldots}((\text{FORK}(xy))^*) \]
Observation

- The size of an iterated derivative grows without bound.
- Inclusion tests that work by constructing a (bi)simulation do not work.
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 $w \in \Sigma^*$, $C(d_w(b)) \leq \varepsilon$. 
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 $\#d(b)$ be the number of dissimilar iterated derivatives of $b$. 
A decidable case for inclusion

**Definition**

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

**Theorem**

Let $b$ be well-behaved. Then $\#d(b) < \infty$. 
A decidable case for inclusion

**Definition**
Let \( \#d(b) \) be the number of dissimilar iterated derivatives of \( b \).

**Theorem**
Let \( b \) be well-behaved. Then \( \#d(b) < \infty \).

**Corollary**
If \( b_1 \) and \( b_2 \) are well-behaved, then “\( \lbrack b_1 \rbrack \subseteq \lbrack b_2 \rbrack \)” is decidable.
A decidable case for inclusion

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $♯d(b)$ be the number of dissimilar iterated derivatives of $b$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $b$ be well-behaved. Then $♯d(b) &lt; ∞$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Corollary</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $b_1$ and $b_2$ are well-behaved, then “$[b_1] ⊆ [b_2]$?” is decidable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>Since $♯d(b_i) &lt; ∞$, we can attempt to construct a bisimulation for $b_1 + b_2 ∼ b_2$. This construction stops after finitely many steps.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Power of forkable expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forkable expressions subsume regular shuffle expressions.</td>
</tr>
<tr>
<td>The reverse direction is not known.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Complexity of word problem?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adding REC</th>
</tr>
</thead>
<tbody>
<tr>
<td>What remains decidable, when we consider the full behavior language of [NN94], e.g, add general recursion?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Synchronizing shuffle</th>
</tr>
</thead>
<tbody>
<tr>
<td>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?</td>
</tr>
</tbody>
</table>
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