Observational Semantics for a Concurrent Lambda Calculus with Reference Cells and Futures

David Sabel

J. W. Goethe-University
Frankfurt, Germany

joint work with:
Joachim Niehren (Lille, France),
Manfred Schmidt-Schauß (Frankfurt, Germany),
Jan Schwinghammer (Saarbrücken, Germany)

MFPS XXIII, 2007
Outline

1. The Calculus $\lambda(fut)$
2. Observational Semantics
3. Correctness Proofs
4. Results & Future work
Properties and Features of $\lambda$(fut)

- proposed by Niehren, Schwinghammer, Smolka 2006, TCS
- core-language of Alice ML
- call-by-value $\lambda$-calculus
- concurrency (a collection of threads / processes)
- synchronisation via handles and futures (eager as well as lazy)
- reference cells (value exchange between threads)
Our Contribution

Observational Semantics

- based on may- and must-convergence

- sensible notion for
  - equivalence of processes
  - equivalence of expressions

- \( \sim \) implies equivalent behavior,
  e.g. distinguishes erroneous and error-free programs

Program Transformations

- investigate correctness of program transformations

- proof techniques for reasoning about correctness of transformations
Related Work

- **Ong 1993, LICS:** (may & must-convergence)
  Non-determinism in a functional setting

- **Kutzner, Schmidt-Schauß 1998, ICFP:** (diagrams)
  A Non-Deterministic Call-by-Need Lambda Calculus

- **Moran, Sands, Carlsson 1999, COORDINATION:** (context lemma)
  Erratic Fudgets: A semantic theory for an embedded coordination language

- **Pitts 2002, Applied Semantics:** (context. equiv. for ML with local state)
  Operational Semantics and Program Equivalence

- **Carayol, Hirschkoff, and Sangiorgi 2005, TCS:** (other must-convergence)
  On the representation of McCarthy’s amb in the \( \pi \)-calculus
Two-Level-Syntax of $\lambda(fut)$

Layer of Processes

\[ p \in \text{Process} ::= p_1 | p_2 | (\nu x)p | x \leftarrow e | x \overset{susp}{\leftarrow} e | x \cdot c \cdot v | y \cdot h \cdot x | y \cdot h \cdot \bullet \]

Layer of $\lambda$-Expressions

\[ e \in \text{Exp} ::= x | c | \lambda x.e | e_1 e_2 | \text{exch}(e_1, e_2) \]

\[ c \in \text{Const} ::= \text{unit} | \text{cell} | \text{thread} | \text{handle} | \text{lazy} \]

\[ v \in \text{Val} ::= x | c | \lambda x.e \quad x, y, z \in \text{Var} \]
Two-Level-Syntax of \( \lambda(\text{fut}) \)

### Layer of Processes

\[
p \in \text{Process} \quad ::= \quad p_1 \mid p_2 \mid (\nu x)p \mid x \leftarrow e \mid x \rightleftharpoons e \mid x \leftarrow c \mid y \leftarrow h \mid x \mid y \leftarrow h \bullet
\]

### Layer of \( \lambda \)-Expressions

\[
e \in \text{Exp} \quad ::= \quad x \mid c \mid \lambda x.e \mid e_1 \mid e_2 \mid \text{exch}(e_1, e_2)
\]

\[
c \in \text{Const} \quad ::= \quad \text{unit} \mid \text{cell} \mid \text{thread} \mid \text{handle} \mid \text{lazy}
\]

\[
v \in \text{Val} \quad ::= \quad x \mid c \mid \lambda x.e \quad \quad x, y, z \in \text{Var}
\]

### Structural Congruence

\[
p_1 \mid p_2 \equiv p_2 \mid p_1 \\
(\nu x)(\nu y)p \equiv (\nu y)(\nu x)p \\
(p_1 \mid p_2) \mid p_3 \equiv p_1 \mid (p_2 \mid p_3) \\
(\nu x)(p_1) \mid p_2 \equiv (\nu x)(p_1 \mid p_2) \quad \text{if } x \not\in \text{fv}(p_2)
\]
### Evaluation Relation

<table>
<thead>
<tr>
<th>Small-Step Reduction</th>
<th>Ev</th>
<th>(local)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\beta\text{-CBV}_L(ev))</td>
<td>(E[(\lambda y.e) \ v] \rightarrow (\nu y)(E[e] \mid y \leftarrow v))</td>
<td></td>
</tr>
<tr>
<td>THREAD.NEW(ev)</td>
<td>(E[\text{thread} \ v] \rightarrow (\nu z)(E[z] \mid z \leftarrow v \ z))</td>
<td></td>
</tr>
<tr>
<td>LAZY.NEW(ev)</td>
<td>(E[\text{lazy} \ v] \rightarrow (\nu z)(E[z] \mid z \leftarrow v \ z))</td>
<td></td>
</tr>
<tr>
<td>LAZY.TRIGGER(ev)</td>
<td>(F[x] \mid x \leftarrow e \rightarrow F[x] \mid x \leftarrow e)</td>
<td></td>
</tr>
<tr>
<td>FUT.DEREF(ev)</td>
<td>(F[x] \mid x \leftarrow v \rightarrow F[v] \mid x \leftarrow v)</td>
<td></td>
</tr>
<tr>
<td>CELL.NEW(ev)</td>
<td>(E[\text{cell} \ v] \rightarrow (\nu z)(E[z] \mid z \ c \ v))</td>
<td></td>
</tr>
<tr>
<td>CELL.EXCH(ev)</td>
<td>(E[\text{exch}(z, v_1)] \mid z \ c \ v_2 \rightarrow E[v_2] \mid z \ c \ v_1)</td>
<td></td>
</tr>
<tr>
<td>HANDLE.NEW(ev)</td>
<td>(E[\text{handle} \ v] \rightarrow (\nu y)(\nu x)(E[v \ y \ x] \mid x \ h \ y))</td>
<td></td>
</tr>
<tr>
<td>HANDLE.BIND(ev)</td>
<td>(E[x \ v] \mid x \ h \ y \rightarrow E[\text{unit}] \mid y \leftarrow v \mid x \ h \bullet)</td>
<td></td>
</tr>
</tbody>
</table>

\(E::=x \leftarrow \tilde{E} \mid \tilde{E} e \mid v \tilde{E} \mid \text{exch}(\tilde{E}, e) \mid \text{exch}(v, \tilde{E})\quad F::=x \leftarrow \tilde{E}[[] \ v] \mid x \leftarrow \tilde{E}[\text{exch}([], v)]\)
### Evaluation Relation

<table>
<thead>
<tr>
<th>Small-Step Reduction $\xrightarrow{\text{ev}}$ (D-closed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta$-CBV$_L$ (ev)</td>
</tr>
<tr>
<td>$D[E[(\lambda y.e) v]] \rightarrow D[(\nu y)(E[e]</td>
</tr>
<tr>
<td>THREAD.NEW (ev)</td>
</tr>
<tr>
<td>$D[E[\text{thread } v]] \rightarrow D[(\nu z)(E[z]</td>
</tr>
<tr>
<td>LAZY.NEW (ev)</td>
</tr>
<tr>
<td>$D[E[\text{lazy } v]] \rightarrow D[(\nu z)(E[z]</td>
</tr>
<tr>
<td>LAZY.TRIGGER (ev)</td>
</tr>
<tr>
<td>$D[F[x]</td>
</tr>
<tr>
<td>FUT.DEREF (ev)</td>
</tr>
<tr>
<td>$D[F[x]</td>
</tr>
<tr>
<td>CELL.NEW (ev)</td>
</tr>
<tr>
<td>$D[E[\text{cell } v]] \rightarrow D[(\nu z)(E[z]</td>
</tr>
<tr>
<td>CELL.EXCH (ev)</td>
</tr>
<tr>
<td>$D[E[\text{exch}(z, v_1)]</td>
</tr>
<tr>
<td>HANDLE.NEW (ev)</td>
</tr>
<tr>
<td>$D[E[\text{handle } v]] \rightarrow D[(\nu y)(\nu x)(E[v _y x]</td>
</tr>
<tr>
<td>HANDLE.BIND (ev)</td>
</tr>
<tr>
<td>$D[E[x v]</td>
</tr>
</tbody>
</table>

**process contexts $D::=\square | p | D | D | p | (\nu x)D$**
Successful Processes

A process \( p \) is **successful**

iff

\( p \) well-formed

and \( \forall x \leftarrow e : x \) is **bound** to a constant, abstraction, cell, lazy future, handle or handled future.

“**bound**” includes chains of indirections

\( x \leftarrow x_1\ |\ x_1 \leftarrow x_2\ |\ \ldots\ |\ x_{n-1} \leftarrow x_n \)

Examples: successful

\( x \leftarrow \lambda y.y \)

\( x \leftarrow y\ |\ y \leftarrow z\ |\ z \leftarrow \text{unit} \)

Examples: not successful

\( x \leftarrow x \)

\( x \leftarrow yx\ |\ y \leftarrow xy \)
Successful Processes

A process $p$ is **successful** iff

$p$ well-formed

and $\forall x \leftarrow e : x$ is **bound** to a constant, abstraction, cell, lazy future, handle or handled future.

“**bound**” includes chains of indirections

$x \leftarrow x_1 \mid x_1 \leftarrow x_2 \mid \ldots \mid x_{n-1} \leftarrow x_n$

**Examples: successful**

- $x \leftarrow \lambda y.y$
- $x \leftarrow y \mid y \leftarrow z \mid z \in \text{unit}$

**Examples: not successful**

- $x \leftarrow x$
- $x \leftarrow yx \mid y \leftarrow xy$
May- and Must-Convergence

**May-Convergence**

\[ p \downarrow \iff \exists p' : p \xrightarrow{\text{ev}}^* p' \land p' \text{successful} \]

**Must-Convergence**

\[ p \downarrow \iff \forall p' : p \xrightarrow{\text{ev}}^* p' \implies p' \downarrow \]

...includes **weak divergences**, i.e. processes that have an infinite evaluation, but all successors w.r.t. \( \xrightarrow{\text{ev}} \) are may-convergent.

**Must-Divergence**

\[ p \uparrow \iff \neg p \downarrow \]

**May-Divergence**

\[ p \uparrow \iff \neg p \downarrow \]
Contextual Equivalence (of Processes)

Two Contextual Preorders

based on may-convergence

\[ p_1 \leq \downarrow p_2 \quad \text{iff} \quad \forall D : D[p_1] \downarrow \implies D[p_2] \downarrow \]

based on must-convergence

\[ p_1 \leq \downarrow\downarrow p_2 \quad \text{iff} \quad \forall D : D[p_1] \downarrow\downarrow \implies D[p_2] \downarrow\downarrow \]

tests may- / must- convergence in all contexts

Contextual Preorder / Contextual Equivalence

\[ \leq = \leq \downarrow \cap \leq \downarrow \downarrow \quad \sim = \leq \cap \geq \]
Why Weak Divergences are Included in Must-Convergence?

**Former Approach**

total must-convergence:

\[ p \downarrow^{\text{total}} \text{ iff all evaluations of } p \text{ terminate successfully.} \]

**Example of Carayol, Hirschkoff, Sangiorgi, 2005**

\[ Y \lambda f. (\text{choice } I \ f) \]

\[ I \]

\[ I \]

\[ I \]

\[ I \]
Why Weak Divergences are Included in Must-Convergence?

**Former Approach**

**total** must-convergence:

\[ p \downarrow_{total} \text{ iff all evaluations of } p \text{ terminate successfully.} \]

**Example of Carayol, Hirschkoff, Sangiorgi, 2005**

\[ Y \lambda f. (\text{choice } I\ f) \]

with **total** must-convergence:

\[ I \not\sim_{total} Y \lambda f. (\text{choice } I\ f) \sim_{total} \text{choice } I \perp \]

with **our** must-convergence:

\[ I \sim Y \lambda f. (\text{choice } I\ f) \not\sim \text{choice } I \perp \]
Why Weak Divergences are Included in Must-Convergence?

**Fairness**

fair evaluation: every possible redex will be reduced eventually

**With our must-convergence**

convergence predicates unchanged if evaluations are restricted to fairness

\[ \downarrow_{\text{fair}} = \downarrow \text{ and } \downarrow_{\text{fair}} = \downarrow \]

Hence: \( \sim_{\text{fair}} = \sim \)
Let \( t \) be a (D-closed) transformation on processes.

**Correctness**

\( t \) is correct iff \( t \subseteq \sim \)

**Proof Plan (show \( t \subseteq \sim \))**

Show for all \( p_1, p_2 \) with \( p_1 \xrightarrow{t} p_2 \):

- \( t \) preserves may-convergence:
  - \( p_1 \downarrow \implies p_2 \downarrow \)
  - \( p_2 \downarrow \implies p_1 \downarrow \)

- \( t \) preserves must-convergence:
  - \( p_1 \downarrow \implies p_2 \downarrow \)
  - \( p_2 \downarrow \implies p_1 \downarrow \)
Forking and Commuting Diagrams for Transformation $t$

Diagrams are **meta-rewriting rules**

$$t' \subseteq t \quad r, r' \subseteq ev \quad f \text{ relations on processes.}$$

**forking diagram**

**commuting diagram**
Forking and Commuting Diagrams for Transformation $t$

Diagrams are meta-rewriting rules

$t' \subseteq t \quad r, r' \subseteq ev \quad f$ relations on processes.

set of forking diagrams
is complete iff for every

$p_1 \xrightarrow{t} p_3$

there is an applicable diagram

set of commuting diagrams
is complete iff for every

$p_1 \xrightarrow{t} p_2$

there is an applicable diagram
Preservation of May-Convergence

\[ \text{prove } p_1 \Downarrow \quad \iff \quad p_2 \Downarrow \]

\[ p_1 \xrightarrow{t} p_2 \]

\[ \begin{array}{c}
\text{ev} \\
\downarrow \\
\vdots \\
\text{ev} \\
\downarrow \\
\vdots \\
\text{ev} \\
\downarrow \\
p'_1
\end{array} \]

\[ \text{successful} \]
Preservation of May-Convergence

\[ \text{prove } p_1 \downarrow \implies p_2 \downarrow \]

\[ p_1 \xrightarrow{t} p_2 \]

\[ \text{ev} \]

\[ \text{ev} \]

\[ t \]

\[ \text{ev} \]

\[ p'_1 \]

\[ \text{successful} \]
Preservation of May-Convergence

\[ \text{prove } p_1 \Downarrow \iff p_2 \Downarrow \]

\[
\begin{array}{c}
 p_1 \xrightarrow{t} p_2 \\
 \Downarrow \quad \quad \Downarrow \\
 \quad \quad \quad \quad t \quad \quad \quad \quad \quad \quad \quad t \\
 \quad \quad \quad \quad \Downarrow \quad \quad \quad \quad \Downarrow \\
 \quad \quad \quad \quad \Downarrow \quad \quad \quad \quad \Downarrow \\
 \quad \quad \quad \quad \Downarrow \quad \quad \quad \quad \Downarrow \\
 p_1' \xrightarrow{t} p_2' \\
\end{array}
\]

successful  successful
Preservation of May-Convergence

prove $p_1 \downarrow \implies p_2 \downarrow$

prove $p_2 \downarrow \implies p_1 \downarrow$
Preservation of May-Convergence

\[ \text{prove } p_1 \downarrow \implies p_2 \downarrow \]

\[ p_1 \xrightarrow{t} p_2 \]

\[ p_1 \xrightarrow{t} p_2 \]

\[ \text{ev} \]

\[ \text{ev} \]

\[ \text{successful} \quad \text{successful} \]

\[ \text{prove } p_2 \downarrow \implies p_1 \downarrow \]

\[ p_1 \xrightarrow{t} p_2 \]

\[ p_1 \xrightarrow{t} p_2 \]

\[ \text{ev} \]

\[ \text{ev} \]

\[ \text{successful} \quad \text{successful} \]
Proving Correctness of a Transformation

Proof Plan

Show for all $p_1, p_2$ with $p_1 \xrightarrow{t} p_2$:

- $t$ preserves may-convergence:
  - $p_1 \downarrow \implies p_2 \downarrow$
  - $p_2 \downarrow \implies p_1 \downarrow$
  - $\checkmark$

- $t$ preserves must-convergence:
  - $p_1 \downarrow \implies p_2 \downarrow$
  - $p_2 \downarrow \implies p_1 \downarrow$
  - $?\,$
Proving Correctness of a Transformation

Proof Plan

Show for all \( p_1, p_2 \) with \( p_1 \xrightarrow{t} p_2 \):

- \( t \) preserves may-convergence:
  - \( p_1 \Downarrow \implies p_2 \Downarrow \)
  - \( p_2 \Downarrow \implies p_1 \Downarrow \)

- \( t \) preserves must-convergence (= preserves may-divergence):
  - \( p_1 \Downarrow \implies p_2 \Downarrow \) equivalent to \( p_2 \Uparrow \implies p_1 \Uparrow \)
  - \( p_2 \Downarrow \implies p_1 \Downarrow \) equivalent to \( p_1 \Uparrow \implies p_2 \Uparrow \)

\( p \Uparrow \) equivalent to \( \exists p' : p \xrightarrow{\text{ev}^*} p' \land p' \Uparrow \)
Preservation of Must-Convergence

prove $p_1 \uparrow \implies p_2 \uparrow$

prove $p_2 \uparrow \implies p_1 \uparrow$

$\begin{array}{c}
\text{must-divergent} \\
\text{must-divergent}
\end{array}$

$\begin{array}{c}
\text{must-divergent} \\
\text{must-divergent}
\end{array}$
Diagram Method

Diagrams may be more complex . . .

- does not work: induction on the length of the reduction sequence
- solution: well-founded measure which is decreased by every diagram

J. Niehren, D. Sabel, M. Schmidt-Schauß, J. Schwinghammer
18 Semantics for a Concurrent λ-Calculus with Cells and Futures
Transformations on Expressions

Contextual Equivalence of Expressions

\[ e_1 \leq \downarrow e_2 \iff \forall C : C[e_1] \downarrow \implies C[e_2] \downarrow \]
\[ e_1 \downarrow \leq e_2 \iff \forall C : C[e_1] \downarrow \implies C[e_2] \downarrow \]

\[ \leq = \leq \downarrow \cap \leq \downarrow \cap \sim \leq \cup \geq \]

Correctness

transformation \( t \) on expressions is correct
if \( t \subseteq \sim \)

Context Lemma

\[ \forall D, E : (D[E[e_1]] \downarrow \implies D[E[e_2]] \downarrow \land D[E[e_1]] \downarrow \implies D[E[e_2]] \downarrow) \implies e_1 \leq e_2 \]

restricts the number of contexts needed to be taken into account!

\[ (\lambda x.e) v \xrightarrow{\text{cbv-\( \beta \)}} e[v/x] : \text{prove correctness of } D[E[(\lambda x.e) v]] \rightarrow D[E[e[v/x]]] \]
Results

Correct Program Transformations

- all reductions except for \texttt{CELL.EXCH(ev)}
- deterministic cell exchange
  \[(\nu x)(E[\texttt{exch}(x, v_1)] \mid x \ cout v_2) \rightarrow (\nu x)(E[v_2] \mid x \ cout v_1)\]
- arbitrary copying of values
  \[C[x] \mid x \leftarrow v \rightarrow C[v] \mid x \leftarrow v \quad \text{if } x \notin \texttt{bv}(C)\]
- garbage collection
  \[p \mid (\nu y_1) \ldots (\nu y_n)p' \rightarrow p\]
  \[\text{if } p' \text{ successful & } y_1, \ldots, y_n \text{ contain all process variables of } p'\]
- call-by-value \(\beta\) (without sharing)
  \[(\lambda x.e) v \rightarrow e[v/x]\]
- ...
Results

Incorrect Program Transformations

- **CELL.EXCH**(ev)
  \[ E[\text{exch}(z, v_1)] | z c v_2 \rightarrow E[v_2] | z c v_1 \]

- call-by-name \( \beta \)
  \[ (\lambda x.e) e' \rightarrow e[e'/x] \]

- **LAZY.TRIGGER**(\( \neg ev \))
  \[ C[x] | x \xleftarrow{\text{susp}} e \rightarrow C[x] | x \xleftarrow{e} \]

- **CELL.NEW**(\( \neg ev \))
  \[ C[\text{cell} v] \rightarrow (\nu z)(C[z] | z \xleftarrow{c} v) \]

- **THREAD.NEW**(\( \neg ev \))
  \[ C[\text{thread} v] \rightarrow (\nu z)(C[z] | z \xleftarrow{\text{susp}} v z) \]

- **LAZY.NEW**(\( \neg ev \))
  \[ C[\text{lazy} v] \rightarrow (\nu z)(C[z] | z \xleftarrow{\text{susp}} v z) \]

- **HANDLE.NEW**(\( \neg ev \))
  \[ C[\text{handle} v] \rightarrow (\nu y)(\nu x)(C[v y x] | x h y) \]

- **HANDLE.BIND**(\( \neg ev \))
  \[ C[x v] | x h y \rightarrow C[\text{unit}] | y \xleftarrow{v} | x h \bullet \]

\( \neg ev \) means that context \( C \) is not an \( E \) context
we have presented an observational equivalence for \( \lambda(\text{fut}) \) based on may- as well as must-convergence.

This enables to reason about the **correctness** of transformations of stateful and concurrent computations.

The used proof methods are successful.

In particular, we proved **correctness of partial evaluation** which is used in compilers.
extensions of the calculus: types, case-expressions and data constructors (e.g. lists)

investigate static analyses (e.g. touch-analysis) and prove correctness of the related optimisations

maybe our methods are applicable to other process calculi like the $\pi$-calculus