JOHANN WOLFGANG GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

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

The Calculus λ(fut)

Properties and Features of λ(fut)

- proposed by Niehren, Schwinghammer, Smolka 2006, TCS
- core-language of Alice ML
- call-by-value λ-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 $\sim$

- 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 π-calculus

Two-Level-Syntax of λ(fut)

Layer of Processes

$$p \in \textit{Process} \quad ::= \quad p_1 \mid p_2 \mid (\nu x)p \mid x \Leftarrow e \mid x \stackrel{\textsf{susp}}{\Longleftarrow} e \mid x \,\mathsf{c}\, v \mid y \,\mathsf{h}\, x \mid y \,\mathsf{h}\, \bullet$$

Layer of λ-Expressions

$$e \in \textit{Exp} \quad ::= \quad x \mid c \mid \lambda x.e \mid e_1\ e_2 \mid \textbf{exch}(e_1, e_2)$$

$$c \in \textit{Const} \quad ::= \quad \textbf{unit} \mid \textbf{cell} \mid \textbf{thread} \mid \textbf{handle} \mid \textbf{lazy}$$

$$v \in \textit{Val} \quad ::= \quad x \mid c \mid \lambda x.e \qquad\qquad x, y, z \in \textit{Var}$$

Two-Level-Syntax of λ(fut)

Layer of Processes

$$p \in \mathit{Process} \quad ::= \quad p_1 \,|\, p_2 \mid (\nu x)p \mid x \Leftarrow e \mid x \overset{\textsf{susp}}{\Longleftarrow} e \mid x \,\mathsf{c}\, v \mid y \,\mathsf{h}\, x \mid y \,\mathsf{h}\, \bullet$$

Layer of λ-Expressions

$$e \in \mathit{Exp} \quad ::= \quad x \mid c \mid \lambda x.e \mid e_1\ e_2 \mid \textbf{exch}(e_1, e_2)$$

$$c \in \mathit{Const} \quad ::= \quad \textbf{unit} \mid \textbf{cell} \mid \textbf{thread} \mid \textbf{handle} \mid \textbf{lazy}$$

$$v \in \mathit{Val} \quad ::= \quad x \mid c \mid \lambda x.e \qquad\qquad x, y, z \in \mathit{Var}$$

Structural Congruence

$$p_1 \,|\, p_2 \equiv p_2 \,|\, p_1 \qquad\qquad (p_1 \,|\, p_2) \,|\, p_3 \equiv p_1 \,|\, (p_2 \,|\, p_3)$$

$$(\nu x)(\nu y)p \equiv (\nu y)(\nu x)p \qquad\qquad (\nu x)(p_1) \,|\, p_2 \equiv (\nu x)(p_1 \,|\, p_2) \quad \text{if } x \notin \mathsf{fv}(p_2)$$

Evaluation Relation

Small-Step Reduction $\xrightarrow{ev}$ (local)

β-CBV$_L$(ev)	$E[(\lambda y.e)\,v] \rightarrow (\nu y)(E[e] \mid y \Leftarrow v)$
THREAD.NEW(ev)	$E[\textbf{thread}\,v] \rightarrow (\nu z)(E[z] \mid z \Leftarrow v\,z)$
LAZY.NEW(ev)	$E[\textbf{lazy}\,v] \rightarrow (\nu z)(E[z] \mid z \overset{\textsf{susp}}{\Longleftarrow} v\,z)$
LAZY.TRIGGER(ev)	$F[x] \mid x \overset{\textsf{susp}}{\Longleftarrow} e \rightarrow F[x] \mid x \Leftarrow e$
FUT.DEREF(ev)	$F[x] \mid x \Leftarrow v \rightarrow F[v] \mid x \Leftarrow v$
CELL.NEW(ev)	$E[\textbf{cell}\,v] \rightarrow (\nu z)(E[z] \mid z\,\mathsf{c}\,v)$
CELL.EXCH(ev)	$E[\textbf{exch}(z, v_1)] \mid z\,\mathsf{c}\,v_2 \rightarrow E[v_2] \mid z\,\mathsf{c}\,v_1$
HANDLE.NEW(ev)	$E[\textbf{handle}\,v] \rightarrow (\nu y)(\nu x)(E[v\,y\,x] \mid x\,\mathsf{h}\,y)$
HANDLE.BIND(ev)	$E[x\,v] \mid x\,\mathsf{h}\,y \rightarrow E[\textbf{unit}] \mid y \Leftarrow v \mid x\,\mathsf{h}\,\bullet$

$E ::= x \Leftarrow \widetilde{E} \quad \widetilde{E} ::= [\,] \mid \widetilde{E}\,e \mid v\,\widetilde{E} \mid \textbf{exch}(\widetilde{E}, e) \mid \textbf{exch}(v, \widetilde{E}) \quad F ::= x \Leftarrow \widetilde{E}[[\,]\,v] \mid x \Leftarrow \widetilde{E}[\textbf{exch}([\,], v)]$

Evaluation Relation

Small-Step Reduction $\xrightarrow{ev}$ (D-closed)

β-CBV$_L$(ev)	$D[E[(\lambda y.e)\, v]] \to D[(\nu y)(E[e] \mid y \Leftarrow v)]$
THREAD.NEW(ev)	$D[E[\textbf{thread}\, v]] \to D[(\nu z)(E[z] \mid z \Leftarrow v\, z)]$
LAZY.NEW(ev)	$D[E[\textbf{lazy}\, v]] \to D[(\nu z)(E[z] \mid z \overset{susp}{\Longleftarrow} v\, z)]$
LAZY.TRIGGER(ev)	$D[F[x] \mid x \overset{susp}{\Longleftarrow} e] \to D[F[x] \mid x \Leftarrow e]$
FUT.DEREF(ev)	$D[F[x] \mid x \Leftarrow v] \to D[F[v] \mid x \Leftarrow v]$
CELL.NEW(ev)	$D[E[\textbf{cell}\, v]] \to D[(\nu z)(E[z] \mid z\, \mathsf{c}\, v)]$
CELL.EXCH(ev)	$D[E[\textbf{exch}(z, v_1)] \mid z\, \mathsf{c}\, v_2] \to D[E[v_2] \mid z\, \mathsf{c}\, v_1]$
HANDLE.NEW(ev)	$D[E[\textbf{handle}\, v]] \to D[(\nu y)(\nu x)(E[v\, y\, x] \mid x\, \mathsf{h}\, y)]$
HANDLE.BIND(ev)	$D[E[x\, v] \mid x\, \mathsf{h}\, y] \to D[E[\textbf{unit}] \mid y \Leftarrow v \mid x\, \mathsf{h}\, \bullet]$

process contexts $D ::= [\,] \mid p \mid D \mid D \mid p \mid (\nu x) D$

Successful Processes

Succesful 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

$x \Leftarrow \lambda y.y$

$x \Leftarrow y \mid y \Leftarrow z \mid z\,\mathbf{c}\,\mathbf{unit}$

Examples: not successful

$x \Leftarrow x$

$x \Leftarrow yx \mid y \Leftarrow xy$

Successful Processes

Succesful 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

$x \Leftarrow \lambda y.y$

$x \Leftarrow y \mid y \Leftarrow z \mid z$ c **unit**

Examples: not successful

$x \Leftarrow x$

$x \Leftarrow yx \mid y \Leftarrow xy$

May- and Must-Convergence

May-Convergence

$p{\downarrow}$ iff $\exists p' : p \xrightarrow{\text{ev}}{}^{*} p' \wedge p'$successful

Must-Divergence

$p{\Uparrow}$ iff $\neg p{\downarrow}$

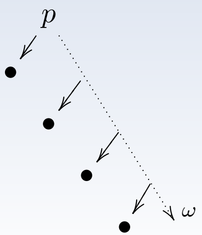
Must-Convergence

$p{\Downarrow}$ iff $\forall p' : p \xrightarrow{\text{ev}}{}^{*} p' \implies p'{\downarrow}$

May-Divergence

$p{\uparrow}$ iff $\neg p{\Downarrow}$

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



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

$$p_1 \leq_{\Downarrow} p_2 \quad \text{iff} \quad \forall D : D[p_1]{\Downarrow} \implies D[p_2]{\Downarrow}$$

tests may- / must- convergence in **all** contexts

Contextual Preorder / Contextual Equivalence

$$\leq \; = \; \leq_{\downarrow} \cap \leq_{\Downarrow} \qquad\qquad \sim \; = \; \leq \cap \geq$$

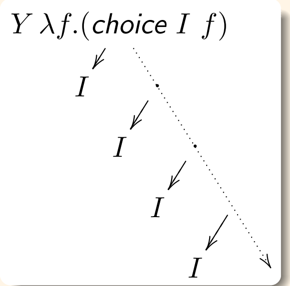
Why Weak Divergences are Included in Must-Convergence?

Former Approach

total must-convergence:

$p{\Downarrow}_{total}$ iff all evaluations of p terminate successfully.

Example of Carayol, Hirschkoff, Sangiorgi, 2005



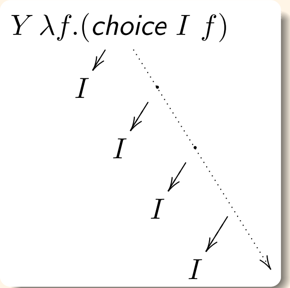
Why Weak Divergences are Included in Must-Convergence?

Former Approach

total must-convergence:

$p{\Downarrow}_{total}$ iff all evaluations of p terminate successfully.

Example of Carayol, Hirschkoff, Sangiorgi, 2005



with total must-convergence:
$I \not\sim_{total} Y\,\lambda f.(choice\ I\ \ f) \sim_{total} choice\ I\ \bot$

with our must-convergence:
$I \sim Y\,\lambda f.(choice\ I\ \ f) \not\sim choice\ I\ \bot$

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

$$\downarrow_{\text{fair}} = \downarrow \quad \text{and} \quad \Downarrow_{\text{fair}} = \Downarrow$$

Hence: $\sim_{\text{fair}} = \sim$

Proving Correctness of a Transformation

Let t be a (D-closed) transformation on processes

Correctness

t is correct iff $t \subseteq \sim$

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

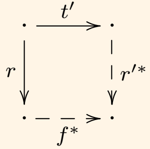
Show for all p_1, p_2 with $p_1 \xrightarrow{t} p_2$:

- t preserves may-convergence:
  - $p_1{\downarrow} \implies p_2{\downarrow}$
  - $p_2{\downarrow} \implies p_1{\downarrow}$
- t preserves must-convergence:
  - $p_1{\Downarrow} \implies p_2{\Downarrow}$
  - $p_2{\Downarrow} \implies p_1{\Downarrow}$

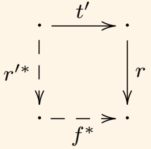
Forking and Commuting Diagrams for Transformation t

Diagrams are meta-rewriting rules

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



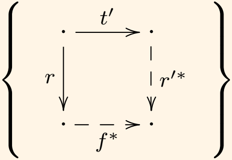
forking diagram



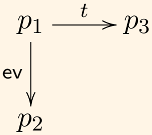
commuting diagram

Forking and Commuting Diagrams for Transformation t

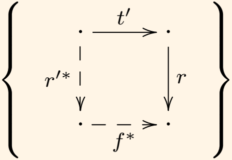
Diagrams are meta-rewriting rules
$t' \subseteq t$ $r, r' \subseteq$ ev f relations on processes.



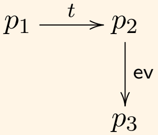
set of forking diagrams is complete iff for every



there is an applicable diagram



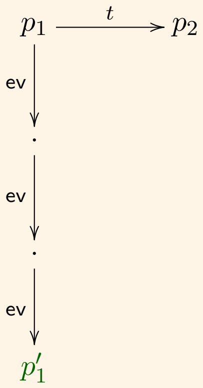
set of commuting diagrams is complete iff for every



there is an applicable diagram

Preservation of May-Convergence

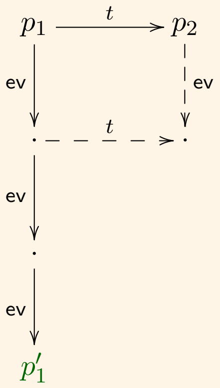
prove $p_1{\downarrow} \implies p_2{\downarrow}$



successful

Preservation of May-Convergence

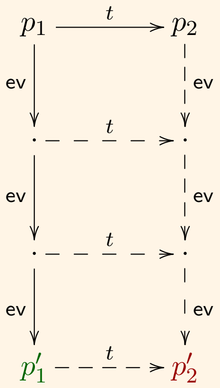
prove $p_1\downarrow \implies p_2\downarrow$



successful

Preservation of May-Convergence

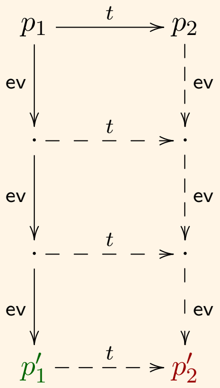
prove $p_1\!\downarrow \implies p_2\!\downarrow$



successful successful

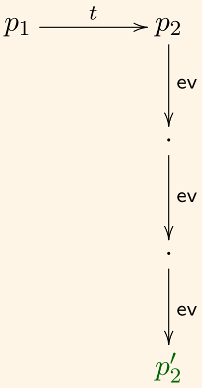
Preservation of May-Convergence

prove $p_1\!\downarrow \implies p_2\!\downarrow$



successful successful

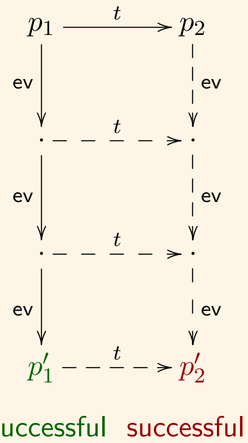
prove $p_2\!\downarrow \implies p_1\!\downarrow$



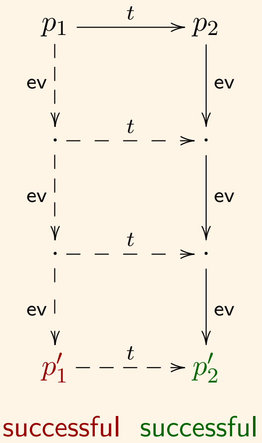
successful

Preservation of May-Convergence

prove $p_1{\downarrow} \implies p_2{\downarrow}$



prove $p_2{\downarrow} \implies p_1{\downarrow}$



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: ?
  - $p_1{\Downarrow} \implies p_2{\Downarrow}$
  - $p_2{\Downarrow} \implies p_1{\Downarrow}$

Proving Correctness of a Transformation

Proof Plan

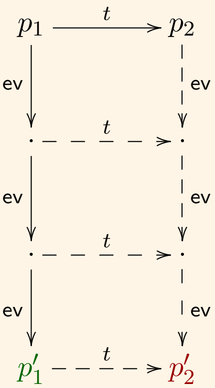
Show for all p_1, p_2 with $p_1 \xrightarrow{t} p_2$:

- 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 $\exists p' : p \xrightarrow{\text{ev}}{}^{*} p' \wedge p'\Uparrow$

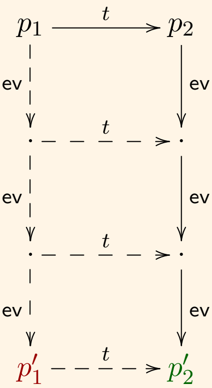
Preservation of Must-Convergence

prove $p_1\uparrow \implies p_2\uparrow$



must-divergent must-divergent

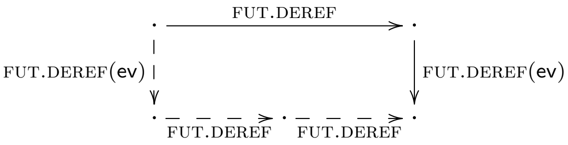
prove $p_2\uparrow \implies p_1\uparrow$

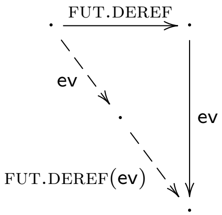


must-divergent must-divergent

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

Transformations on Expressions

Contextual Equivalence of Expressions

$e_1 \leq_{\downarrow} e_2$ iff $\forall C : C[e_1]\downarrow \implies C[e_2]\downarrow$
$e_1 \leq_{\Downarrow} e_2$ iff $\forall C : C[e_1]\Downarrow \implies C[e_2]\Downarrow$

$\leq = \leq_{\downarrow} \cap \leq_{\Downarrow}$ $\qquad \sim = \leq \cap \geq$

Correctness

transformation t on expressions is correct if $t \subseteq \sim$

Context Lemma

$$\forall D, E : (D[E[e_1]]\downarrow \implies D[E[e_2]]\downarrow \;\wedge\; D[E[e_1]]\Downarrow \implies D[E[e_2]]\Downarrow)$$
$$\implies$$
$$e_1 \leq e_2$$

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

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

Results

Correct Program Transformations

- all reductions except for CELL.EXCH(ev)
- deterministic cell exchange

$$(\nu x)(E[\mathbf{exch}(x, v_1)] \mid x \,\mathsf{c}\, v_2) \to (\nu x)(E[v_2] \mid x \,\mathsf{c}\, v_1)$$

- arbitrary copying of values

$$C[x] \mid x \Leftarrow v \to C[v] \mid x \Leftarrow v \qquad \text{if } x \notin \mathsf{bv}(C)$$

- garbage collection

$$p \mid (\nu y_1) \dots (\nu y_n) p' \to p$$

if p' successful & $y_1, \dots, y_n$ contain all process variables of p'

- call-by-value β (without sharing)

$$(\lambda x.e)\ v \ \to\ e[v/x]$$

- ...

Results

Incorrect Program Transformations

- CELL.EXCH(ev)
  $E[\mathbf{exch}(z, v_1)] \mid z\,\mathsf{c}\,v_2 \rightarrow E[v_2] \mid z\,\mathsf{c}\,v_1$
- call-by-name β
  $(\lambda x.e)\ e' \rightarrow e[e'/x]$
- LAZY.TRIGGER(¬ev)
  $C[x] \mid x \stackrel{susp}{\Longleftarrow} e \rightarrow C[x] \mid x \Leftarrow e$
- CELL.NEW(¬ev)
  $C[\mathbf{cell}\ v] \rightarrow (\nu z)(C[z] \mid z\,\mathsf{c}\,v)$
- THREAD.NEW(¬ev)
  $C[\mathbf{thread}\ v] \rightarrow (\nu z)(C[z] \mid z \Leftarrow v\ z)$
- LAZY.NEW(¬ev)
  $C[\mathbf{lazy}\ v] \rightarrow (\nu z)(C[z] \mid z \stackrel{susp}{\Longleftarrow} v\ z)$
- HANDLE.NEW(¬ev)
  $C[\mathbf{handle}\ v] \rightarrow (\nu y)(\nu x)(C[v\ y\ x] \mid x\,\mathsf{h}\,y)$
- HANDLE.BIND(¬ev)
  $C[x\ v] \mid x\,\mathsf{h}\,y \rightarrow C[\mathbf{unit}] \mid y \Leftarrow v \mid x\,\mathsf{h}\,\bullet$

¬ev means that context C is not an E context

Conclusion

- we have presented an observational equivalence for λ(fut) based on may- as well as must-convergence
- 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

Future Work

- 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 π-calculus