

# 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,  
New Orleans, USA, April 11 - April 14, 2007



# Outline

- 1 The Calculus  $\lambda(\text{fut})$
- 2 Observational Semantics
- 3 Correctness Proofs
- 4 Results & Future work



# The Calculus $\lambda(\text{fut})$

## Properties and Features of $\lambda(\text{fut})$

- 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 $\sim$

- 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, Hirschhoff, and Sangiorgi 2005, TCS: (other must-convergence)  
On the representation of McCarthy's amb in the  $\pi$ -calculus



# Two-Level-Syntax of $\lambda(\text{fut})$

## Layer of Processes

$$p \in \text{Process} ::= p_1 \mid p_2 \mid (\nu x)p \mid x \leftarrow e \mid x \xleftarrow{\text{susp}} e \mid x \text{ c } v \mid y \text{ h } x \mid y \text{ h } \bullet$$

## Layer of $\lambda$ -Expressions

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

$$c \in \text{Const} ::= \mathbf{unit} \mid \mathbf{cell} \mid \mathbf{thread} \mid \mathbf{handle} \mid \mathbf{lazy}$$

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



# Two-Level-Syntax of $\lambda(\text{fut})$

## Layer of Processes

$$p \in \text{Process} ::= p_1 \mid p_2 \mid (\nu x)p \mid x \leftarrow e \mid x \xleftarrow{\text{susp}} e \mid x \text{ c } v \mid y \text{ h } x \mid y \text{ h } \bullet$$

## Layer of $\lambda$ -Expressions

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

$$c \in \text{Const} ::= \mathbf{unit} \mid \mathbf{cell} \mid \mathbf{thread} \mid \mathbf{handle} \mid \mathbf{lazy}$$

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

## Structural Congruence

$$p_1 \mid p_2 \equiv p_2 \mid p_1$$

$$(p_1 \mid p_2) \mid p_3 \equiv p_1 \mid (p_2 \mid p_3)$$

$$(\nu x)(\nu y)p \equiv (\nu y)(\nu x)p$$

$$(\nu x)(p_1) \mid p_2 \equiv (\nu x)(p_1 \mid p_2) \quad \text{if } x \notin \text{fv}(p_2)$$



# Evaluation Relation

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

$\beta\text{-CBV}_L(\text{ev})$	$E[(\lambda y.e) v] \rightarrow (\nu y)(E[e] \mid y \leftarrow v)$
$\text{THREAD.NEW}(\text{ev})$	$E[\mathbf{thread} v] \rightarrow (\nu z)(E[z] \mid z \leftarrow v z)$
$\text{LAZY.NEW}(\text{ev})$	$E[\mathbf{lazy} v] \rightarrow (\nu z)(E[z] \mid z \xleftarrow{\text{susp}} v z)$
$\text{LAZY.TRIGGER}(\text{ev})$	$F[x] \mid x \xleftarrow{\text{susp}} e \rightarrow F[x] \mid x \leftarrow e$
$\text{FUT.DEREF}(\text{ev})$	$F[x] \mid x \leftarrow v \rightarrow F[v] \mid x \leftarrow v$
$\text{CELL.NEW}(\text{ev})$	$E[\mathbf{cell} v] \rightarrow (\nu z)(E[z] \mid z c v)$
$\text{CELL.EXCH}(\text{ev})$	$E[\mathbf{exch}(z, v_1)] \mid z c v_2 \rightarrow E[v_2] \mid z c v_1$
$\text{HANDLE.NEW}(\text{ev})$	$E[\mathbf{handle} v] \rightarrow (\nu y)(\nu x)(E[v y x] \mid x h y)$
$\text{HANDLE.BIND}(\text{ev})$	$E[x v] \mid x h y \rightarrow E[\mathbf{unit}] \mid y \leftarrow v \mid x h \bullet$

$$E ::= x \leftarrow \tilde{E} \quad \tilde{E} ::= [] \mid \tilde{E} e \mid v \tilde{E} \mid \mathbf{exch}(\tilde{E}, e) \mid \mathbf{exch}(v, \tilde{E}) \quad F ::= x \leftarrow \tilde{E} [[] v] \mid x \leftarrow \tilde{E}[\mathbf{exch}([], v)]$$





# Evaluation Relation

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

$\beta\text{-CBV}_L(\text{ev})$	$D[E[(\lambda y.e) v]] \rightarrow D[(\nu y)(E[e] \mid y \leftarrow v)]$
$\text{THREAD.NEW}(\text{ev})$	$D[E[\mathbf{thread} v]] \rightarrow D[(\nu z)(E[z] \mid z \leftarrow v z)]$
$\text{LAZY.NEW}(\text{ev})$	$D[E[\mathbf{lazy} v]] \rightarrow D[(\nu z)(E[z] \mid z \xleftarrow{\text{susp}} v z)]$
$\text{LAZY.TRIGGER}(\text{ev})$	$D[F[x] \mid x \xleftarrow{\text{susp}} e] \rightarrow D[F[x] \mid x \leftarrow e]$
$\text{FUT.DEREF}(\text{ev})$	$D[F[x] \mid x \leftarrow v] \rightarrow D[F[v] \mid x \leftarrow v]$
$\text{CELL.NEW}(\text{ev})$	$D[E[\mathbf{cell} v]] \rightarrow D[(\nu z)(E[z] \mid z c v)]$
$\text{CELL.EXCH}(\text{ev})$	$D[E[\mathbf{exch}(z, v_1)] \mid z c v_2] \rightarrow D[E[v_2] \mid z c v_1]$
$\text{HANDLE.NEW}(\text{ev})$	$D[E[\mathbf{handle} v]] \rightarrow D[(\nu y)(\nu x)(E[v y x] \mid x h y)]$
$\text{HANDLE.BIND}(\text{ev})$	$D[E[x v] \mid x h y] \rightarrow D[E[\mathbf{unit}] \mid y \leftarrow v \mid x h \bullet]$

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



# Successful Processes

## 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

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

### Examples: successful

$$x \leftarrow \lambda y. y$$

$$x \leftarrow y \mid y \leftarrow z \mid z \text{ c unit}$$

### Examples: not successful

$$x \leftarrow x$$

$$x \leftarrow yx \mid y \leftarrow xy$$



# Successful Processes

## 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

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

## Examples: successful

$$x \leftarrow \lambda y. y$$

$$x \leftarrow y \mid y \leftarrow z \mid z \text{ c unit}$$

## 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-Convergence

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

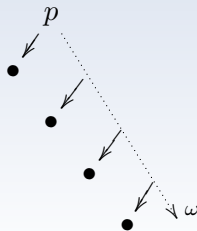
...includes **weak divergences**,  
i.e. processes that have an infinite evaluation, but all successors w.r.t.  $\xrightarrow{\text{ev}}$  are may-convergent

## 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

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

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

## Contextual Preorder / Contextual Equivalence

$$\leq = \leq_{\downarrow} \cup \leq_{\Downarrow}$$

$$\sim = \leq \cap \geq$$



# Why Weak Divergences are Included in Must-Convergence?

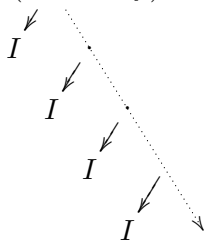
## Former Approach

**total** must-convergence:

$p \Downarrow_{\text{total}}$  iff all evaluations of  $p$  terminate successfully.

## Example of Carayol, Hirschhoff, Sangiorgi, 2005

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





# Why Weak Divergences are Included in Must-Convergence?

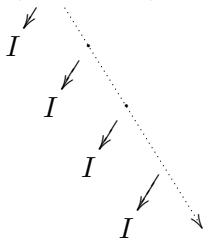
## Former Approach

**total** must-convergence:

$p \Downarrow_{\text{total}}$  iff all evaluations of  $p$  terminate successfully.

## Example of Carayol, Hirschkoﬀ, Sangiorgi, 2005

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



with **total** must-convergence:

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

with **our** must-convergence:

$I \sim Y \lambda f.(\text{choice } I f) \not\Downarrow \text{choice } I \perp$



# 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

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

**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  $p_1 \xrightarrow{t} p_2$ :

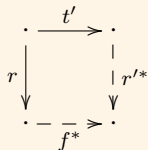
- $t$  preserves may-convergence:
  - $p_1 \downarrow \implies p_2 \downarrow$
  - $p_2 \downarrow \implies p_1 \downarrow$
- $t$  preserves must-convergence:
  - $p_1 \Downarrow \implies p_2 \Downarrow$
  - $p_2 \Downarrow \implies p_1 \Downarrow$



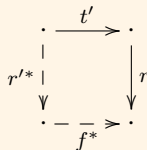
# Forking and Commuting Diagrams for Transformation $t$

Diagrams are **meta-rewriting rules**

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



*forking diagram*



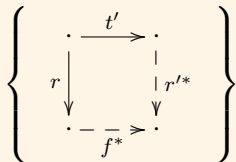
*commuting diagram*



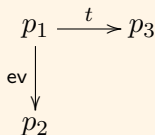
# Forking and Commuting Diagrams for Transformation $t$

Diagrams are **meta-rewriting rules**

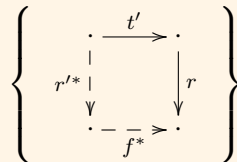
$t' \subseteq t$       $r, r' \subseteq \text{ev}$       $f$  relations on processes.



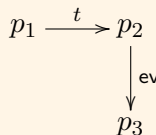
**set** of forking diagrams  
is **complete** iff for every



there is an applicable diagram



**set** of commuting diagrams  
is **complete** iff for every

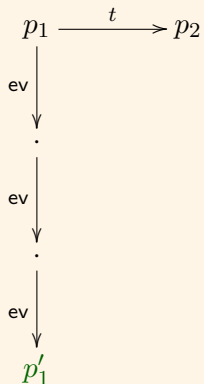


there is an applicable diagram



# Preservation of May-Convergence

prove  $p_1 \Downarrow \implies p_2 \Downarrow$

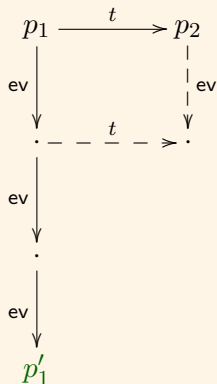


successful



# Preservation of May-Convergence

prove  $p_1 \Downarrow \implies p_2 \Downarrow$

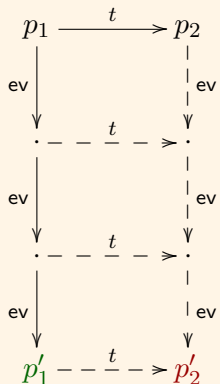


successful



# Preservation of May-Convergence

prove  $p_1 \Downarrow \implies p_2 \Downarrow$

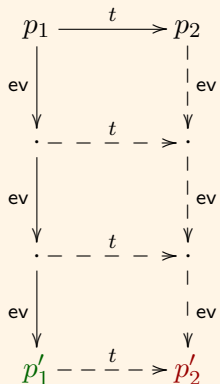


successful successful



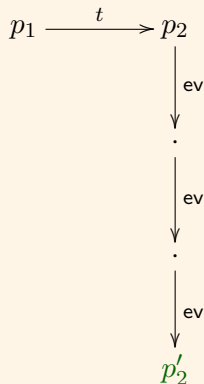
# Preservation of May-Convergence

prove  $p_1 \Downarrow \implies p_2 \Downarrow$



successful successful

prove  $p_2 \Downarrow \implies p_1 \Downarrow$

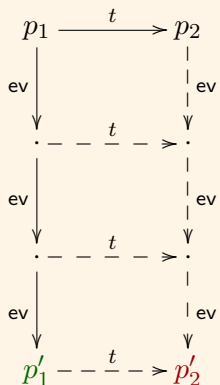


successful



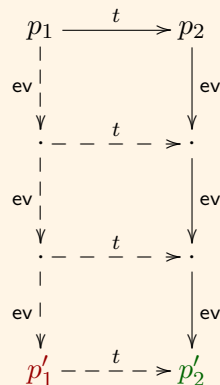
# Preservation of May-Convergence

prove  $p_1 \Downarrow \implies p_2 \Downarrow$



successful successful

prove  $p_2 \Downarrow \implies p_1 \Downarrow$



successful 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:
  - $p_1 \downarrow \implies p_2 \downarrow$  ✓
  - $p_2 \downarrow \implies p_1 \downarrow$
- $t$  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**     $p_2 \Uparrow \implies p_1 \Uparrow$

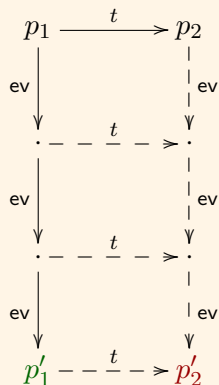
- $p_2 \Downarrow \implies p_1 \Downarrow$     **equivalent to**     $p_1 \Uparrow \implies p_2 \Uparrow$

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



# Preservation of Must-Convergence

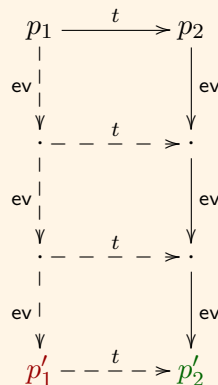
prove  $p_1 \uparrow \implies p_2 \uparrow$



must-  
divergent

must-  
divergent

prove  $p_2 \uparrow \implies p_1 \uparrow$



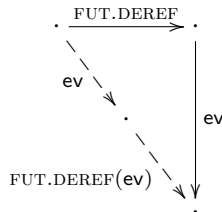
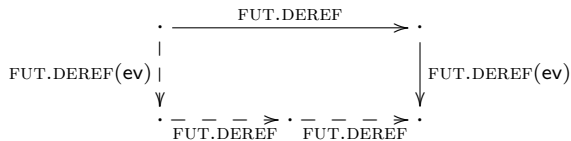
must-  
divergent

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

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

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

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

## Correctness

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

## Context Lemma

$$\begin{aligned} \forall D, E : (D[E[e_1]]) \downarrow \implies D[E[e_2]] \downarrow \quad \wedge \quad D[E[e_1]] \downarrow \implies D[E[e_2]] \downarrow \\ \implies \\ e_1 \leq e_2 \end{aligned}$$

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  $D[E[(\lambda x.e) v]] \rightarrow D[E[e[v/x]]]$



# Results

## Correct Program Transformations

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

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

- arbitrary copying of values

$$C[x] \mid x \leftarrow v \rightarrow C[v] \mid x \leftarrow v \quad \text{if } x \notin \text{bv}(C)$$

- garbage collection

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

if  $p'$  successful &  $y_1, \dots, y_n$  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**( $\neg\text{ev}$ )  
 $E[\mathbf{exch}(z, v_1)] \mid z \text{ c } v_2 \rightarrow E[v_2] \mid z \text{ c } v_1$
- **call-by-name**  $\beta$   
 $(\lambda x.e) e' \rightarrow e[e'/x]$
- **LAZY.TRIGGER**( $\neg\text{ev}$ )  
 $C[x] \mid x \xleftarrow{\text{susp}} e \rightarrow C[x] \mid x \leftarrow e$
- **CELL.NEW**( $\neg\text{ev}$ )  
 $C[\mathbf{cell} v] \rightarrow (\nu z)(C[z] \mid z \text{ c } v)$
- **THREAD.NEW**( $\neg\text{ev}$ )  
 $C[\mathbf{thread} v] \rightarrow (\nu z)(C[z] \mid z \leftarrow v z)$
- **LAZY.NEW**( $\neg\text{ev}$ )  
 $C[\mathbf{lazy} v] \rightarrow (\nu z)(C[z] \mid z \xleftarrow{\text{susp}} v z)$
- **HANDLE.NEW**( $\neg\text{ev}$ )  
 $C[\mathbf{handle} v] \rightarrow (\nu y)(\nu x)(C[v y x] \mid x \text{ h } y)$
- **HANDLE.BIND**( $\neg\text{ev}$ )  
 $C[x v] \mid x \text{ h } y \rightarrow C[\mathbf{unit}] \mid y \leftarrow v \mid x \text{ h } \bullet$

$\neg\text{ev}$  means that context  $C$  is not an  $E$  context



# Conclusion

- we have presented an observational equivalence for  $\lambda(\text{fut})$  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  $\pi$ -calculus