Semantics of a Call-by-Need Lambda Calculus with McCarthy's amb for Program Equivalence

David Sabel

Goethe-Universität, Frankfurt

19. August 2008

Übersicht

1. Motivation und Einleitung
2. Der $\Lambda_{\mathtt{amb}}^{\mathtt{let}}$-Kalkül
3. Kontextuelle Gleichheit und Korrektheit von Programmtransformationen
4. Standardisierung und weitere Beweismethoden
5. Die nebenläufige abstrakte Maschine
6. Fazit & Ausblick

Motivation

Compiler-Korrektheit

- Korrektheit: Transformation ohne Änderung der Programmsemantik
- Benötigt formales Modell zur einheitlichen Beschreibung der Semantik
- Techniken für Korrektheitsnachweise von Programmtransformationen

Welche Programmiersprachen?

- Parallele / Nebenläufige Programmierkonstrukte:
  Hardware: Multicore-Architekturen, Software: Multiuser-, Multithreaded-Systeme
- Funktionale Programmiersprachen (FP):
  deklarativ, modularer Entwurf, mathematisch einfach(er) handhabbar, Typisierung
- Verzögert-auswertende FP:
  erlauben Parallelisierung, Umgang mit unendlichen Datenstrukturen (z.B. Ströme)

Untersuchung

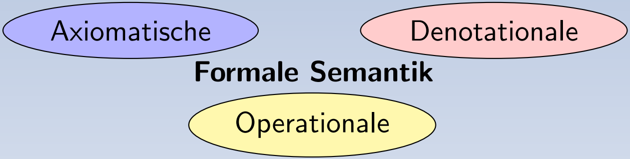
Modell

- Kernsprache einer verzögert auswertenden FP mit
  - `letrec`, Datenkonstruktoren, `case`-Ausdrücken, `seq`
  - McCarthy's [McCarthy, 1963] `amb`-Operator:
    - *`amb` s t wählt s oder t, aber muss divergentes Argument vermeiden*
    - *nebenläufige Auswertung von s und t unter Beachtung von Fairness*
    - *viele nichtdeterministische Operatoren kodierbar*

Programmtransformationen

- Transformationen / Optimierung innerhalb derselben Sprache
  z.B. Prozedureinsetzung, partielles Auswerten, ...
- Übersetzung von einer Sprache in eine (maschinennähere) Sprache.

Welche Semantik?



- axiomatische Semantik: eher für imperative PS
- denotationale Semantik: "schwer" im nichtdeterministischen Fall, `amb`: keine brauchbaren denotationalen Modelle

Gewählter Ansatz

- Operationale small-step Semantik
- kontextuelle Äquivalenz bezüglich may- und must-Konvergenz
- kanonischen Gleichheitsbegriff mit maximaler Menge an Gleichheiten

Related work

Schmidt-Schauß '03, TR
FUNDIO-calculus
Diagramme, faire must, Kontextlemma

Carayol et al. '05, TCS
encoding amb in π-calculus
amb, faire must

Moran '98, Diss.
call-by-name, call-by-need and amb
amb, may/must

Ong '93, LICS
nondeterminism functional setting
erste may/must

Moran et al. '03, SciCo
erratic fudgets
Kontextlemma, may/must, Anwendungen

Kutzner&Schmidt-Schauß '98, ICFP
nondet. call-by-need lambda-calculus
Diagramme, may/must

Der $\Lambda^{\mathtt{let}}_{\mathtt{amb}}$-Kalkül

Syntax

$E ::= V$	(*Variable*)
$\mid (\lambda V.E)$	(*Abstraktion*)
$\mid (E_1\ E_2)$	(*Applikation*)
$\mid (\mathtt{letrec}\ V_1 = E_1, \ldots V_n = E_n\ \mathtt{in}\ E)$	(letrec-*Ausdruck*)
$\mid (\mathtt{case}_T\ E\ Alt_1 \ldots Alt_{\lvert T \rvert})$	(case-*Ausdruck*)
$\mid (c_{T,i}\ E_1 \ldots E_{\mathrm{ar}(c_{T,i})})$	(*Konstruktorapplikation*)
$\mid (\mathtt{seq}\ E_1\ E_2)$	(seq-*Ausdruck*)
$\mid (\mathtt{amb}\ E_1\ E_2)$	(amb-*Ausdruck*)
$Alt ::= ((c_{T,i}\ V_1 \ldots V_{\mathrm{ar}(c_{T,i})}) \ \rightarrow\ E)$	(case-*Alternative*)

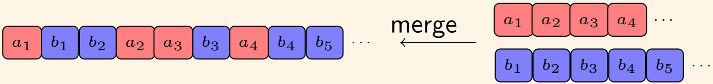
klassischer Lambdakalkül
+rekursive let-Ausdrücke
+case & Konstruktoren
+sequentielle Auswertungen
+nichtdeterministisches amb

if s_1 then s_2 else s_3

$\mathtt{case}_{\mathtt{Bool}}\ s_1\ (\mathtt{True} \rightarrow s_2)\ (\mathtt{False} \rightarrow s_3)$

Kodierungen mittels amb

- par $\equiv \lambda x.\lambda y.\mathtt{amb}\ (\mathtt{seq}\ x\ y)\ y$
- spar $\equiv \lambda x.\lambda y.\mathtt{amb}\ (\mathtt{seq}\ x\ (\mathtt{seq}\ y\ (\mathtt{Pair}\ x\ y)))\ (\mathtt{seq}\ y\ (\mathtt{seq}\ x\ (\mathtt{Pair}\ x\ y)))$
- choice $\equiv \lambda x.\lambda y.((\mathtt{amb}\ \lambda z_1.x\ \lambda z_2.y)\ \mathtt{True})$
- dchoice $\equiv \lambda x.\lambda y.\mathtt{amb}\ (\mathtt{seq}\ x\ (\mathtt{seq}\ y\ x))\ (\mathtt{seq}\ y\ (\mathtt{seq}\ x\ y))$
- por $\equiv \lambda x.\lambda y.\mathtt{amb}\ (\mathtt{if}\ x\ \mathtt{then}\ \mathtt{True}\ \mathtt{else}\ y)\ (\mathtt{if}\ y\ \mathtt{then}\ \mathtt{True}\ \mathtt{else}\ x)$
- pconv $\equiv \lambda x.\lambda y.\lambda z.\mathtt{if}\ (\mathtt{amb}\ (\mathtt{seq}\ x\ \mathtt{True})\ (\mathtt{seq}\ y\ \mathtt{True}))\ \mathtt{then}\ z\ \mathtt{else}\ z$



- Divergenz-vermeidendes Merge (kodierbar): Falls ein Eingabestrom endlich oder partiell, dann erscheinen alle Elemente des anderen
- Unendlich-faires Merge (kodierbar): Wenn ein Eingabestrom unendlich, dann erscheinen alle Elemente des anderen
- Faires Merge: amb kann faires Merge nicht kodieren [Panangaden, 1988]

Semantik

Kontexte $C \in \mathcal{C}$: Ausdrücke mit einem **Loch** $[\cdot]$
Werte: Abstraktionen $\lambda x.s$ und Konstruktorapplikationen $(c_{T,i}\ s_1\ \ldots\ s_{\text{ar}(c_{T,i})})$

Deterministische Reduktionsregeln

(lbeta) $(\lambda x.s)\ r \longrightarrow (\texttt{letrec}\ x = r\ \texttt{in}\ s)$

(cp) $\texttt{letrec}\ x_1 = \lambda x.s, x_2 = x_1 \ldots, x_m = x_{m-1} \ldots C[x_m] \ldots$
$\longrightarrow \texttt{letrec}\ x_1 = \lambda x.s, x_2 = x_1, \ldots x_m = x_{m-1} \ldots C[\lambda x.s] \ldots$

…Regeln zur case- und seq-Auswertung, let-Verschiebungen

Nichtdeterministische Reduktionsregeln: (amb)

(amb-l)
- $\text{amb}\ v\ t \longrightarrow v$ (v Wert)
- $\texttt{letrec}\ x_1 = v, x_2 = x_1 \ldots, x_m = x_{m-1} \ldots C[\text{amb}\ x_m\ t] \ldots$ (v Wert)
  $\longrightarrow \texttt{letrec}\ x_1 = v, x_2 = x_1, \ldots x_m = x_{m-1} \ldots C[x_m] \ldots$

(amb-r)
- $\text{amb}\ t\ v \longrightarrow v$ (v Wert)
- $\texttt{letrec}\ x_1 = v, x_2 = x_1 \ldots, x_m = x_{m-1} \ldots C[\text{amb}\ t\ x_m] \ldots$ (v Wert)
  $\longrightarrow \texttt{letrec}\ x_1 = v, x_2 = x_1, \ldots x_m = x_{m-1} \ldots C[x_m] \ldots$

Normalordnungsreduktion

$\xrightarrow{\textbf{no}}$ = Anwenden einer Reduktionsregel "mithilfe" eines maximalen Reduktionskontextes

Maximalen Reduktionkontext finden: Unwinding-Algorithmus $\mathcal{UW}$

$$\mathtt{uw}((s\ t), R) \to \mathtt{uw}(s, R[([\cdot]\ t)])$$

$$\mathtt{uw}(\mathtt{seq}\ s\ t, R) \to \mathtt{uw}(s, R[\mathtt{seq}\ [\cdot]\ t])$$

$$\mathtt{uw}(\mathtt{case}_T\ s\ alts, R) \to \mathtt{uw}(s, R[\mathtt{case}_T\ [\cdot]\ alts])$$

$$\mathtt{uw}(\mathtt{amb}\ s\ t, R) \to \mathtt{uw}(s, R[\mathtt{amb}\ [\cdot]\ t])\ \textbf{oder}\ \mathtt{uw}(t, R[\mathtt{amb}\ s\ [\cdot]])$$

$$\mathtt{uw}(x, (\mathtt{letrec}\ x = s, Env\ \mathtt{in}\ R^-)) \to \mathtt{uw}(s, (\mathtt{letrec}\ x \doteq [\cdot], Env\ \mathtt{in}\ R^-[x]))$$

$$\mathtt{uw}(x, (\mathtt{letrec}\ y \doteq R^-, x = s, Env\ \mathtt{in}\ t))$$
$$\to \mathtt{uw}(s, (\mathtt{letrec}\ y \doteq R^-[x], x \doteq [\cdot], Env\ \mathtt{in}\ t))$$

$$\mathtt{uw}(s, R) \to (s, R)\ \text{falls keine andere Regel anwendbar}$$

Beispiel: $\mathtt{letrec}\ x_2 = \lambda x.x, x_1 = (x_2\ x_1), x_3 = (\mathtt{amb}\ (x_2\ x_1)\ y)\ \mathtt{in}\ \mathtt{amb}\ x_1\ x_3$

Normalordnungsreduktion

$\xrightarrow{\textbf{no}}$ = Anwenden einer Reduktionsregel "mithilfe" eines maximalen Reduktionskontextes

Maximalen Reduktionkontext finden: Unwinding-Algorithmus $\mathcal{UW}$

$\mathtt{uw}((s\ t), R) \to \mathtt{uw}(s, R[([\cdot]\ t)])$

$\mathtt{uw}(\mathtt{seq}\ s\ t, R) \to \mathtt{uw}(s, R[\mathtt{seq}\ [\cdot]\ t])$

$\mathtt{uw}(\mathtt{case}_T\ s\ alts, R) \to \mathtt{uw}(s, R[\mathtt{case}_T\ [\cdot]\ alts])$

$\mathtt{uw}(\mathtt{amb}\ s\ t, R) \to \mathtt{uw}(s, R[\mathtt{amb}\ [\cdot]\ t])$ **oder** $\mathtt{uw}(t, R[\mathtt{amb}\ s\ [\cdot]])$

$\mathtt{uw}(x, (\mathtt{letrec}\ x = s, Env\ \mathtt{in}\ R^-)) \to \mathtt{uw}(s, (\mathtt{letrec}\ x \stackrel{\bullet}{=} [\cdot], Env\ \mathtt{in}\ R^-[x]))$

$\mathtt{uw}(x, (\mathtt{letrec}\ y \stackrel{\bullet}{=} R^-, x = s, Env\ \mathtt{in}\ t))$
$\qquad \to \mathtt{uw}(s, (\mathtt{letrec}\ y \stackrel{\bullet}{=} R^-[x], x \stackrel{\bullet}{=} [\cdot], Env\ \mathtt{in}\ t))$

$\mathtt{uw}(s, R) \to (s, R)$ falls keine andere Regel anwendbar

Beispiel: $\mathtt{letrec}\ x_2 = \lambda x.x, x_1 = (x_2\ x_1), x_3 = (\mathtt{amb}\ (x_2\ x_1)\ y)\ \mathtt{in}\ \mathtt{amb}\ x_1\ x_3$

Normalordnungsreduktion

$\xrightarrow{\textbf{no}}$ = Anwenden einer Reduktionsregel "mithilfe" eines maximalen Reduktionskontextes

Maximalen Reduktionkontext finden: Unwinding-Algorithmus $\mathcal{UW}$

$$\texttt{uw}((s\ t), R) \to \texttt{uw}(s, R[([\cdot]\ t)])$$

$$\texttt{uw}(\texttt{seq}\ s\ t, R) \to \texttt{uw}(s, R[\texttt{seq}\ [\cdot]\ t])$$

$$\texttt{uw}(\texttt{case}_T\ s\ alts, R) \to \texttt{uw}(s, R[\texttt{case}_T\ [\cdot]\ alts])$$

$$\texttt{uw}(\texttt{amb}\ s\ t, R) \to \texttt{uw}(s, R[\texttt{amb}\ [\cdot]\ t])\ \textbf{oder}\ \texttt{uw}(t, R[\texttt{amb}\ s\ [\cdot]])$$

$$\texttt{uw}(x, (\texttt{letrec}\ x = s, Env\ \texttt{in}\ R^-)) \to \texttt{uw}(s, (\texttt{letrec}\ x \doteq [\cdot], Env\ \texttt{in}\ R^-[x]))$$

$$\texttt{uw}(x, (\texttt{letrec}\ y \doteq R^-, x = s, Env\ \texttt{in}\ t))$$
$$\to \texttt{uw}(s, (\texttt{letrec}\ y \doteq R^-[x], x \doteq [\cdot], Env\ \texttt{in}\ t))$$

$$\texttt{uw}(s, R) \to (s, R)\ \text{falls keine andere Regel anwendbar}$$

Beispiel: $\texttt{letrec}\ x_2 = \lambda x.x, x_1 \doteq (x_2\ x_1), x_3 = (\texttt{amb}\ (x_2\ x_1)\ y)\ \texttt{in}\ \texttt{amb}\ x_1\ x_3$

Normalordnungsreduktion

$\xrightarrow{\mathbf{no}}$ = Anwenden einer Reduktionsregel "mithilfe" eines maximalen Reduktionskontextes

Maximalen Reduktionkontext finden: Unwinding-Algorithmus $\mathcal{UW}$

$$\mathtt{uw}((s\ t), R) \to \mathtt{uw}(s, R[([\cdot]\ t)])$$

$$\mathtt{uw}(\mathtt{seq}\ s\ t, R) \to \mathtt{uw}(s, R[\mathtt{seq}\ [\cdot]\ t])$$

$$\mathtt{uw}(\mathtt{case}_T\ s\ alts, R) \to \mathtt{uw}(s, R[\mathtt{case}_T\ [\cdot]\ alts])$$

$$\mathtt{uw}(\mathtt{amb}\ s\ t, R) \to \mathtt{uw}(s, R[\mathtt{amb}\ [\cdot]\ t])\ \textbf{oder}\ \mathtt{uw}(t, R[\mathtt{amb}\ s\ [\cdot]])$$

$$\mathtt{uw}(x, (\mathtt{letrec}\ x = s, Env\ \mathtt{in}\ R^-)) \to \mathtt{uw}(s, (\mathtt{letrec}\ x \stackrel{\bullet}{=} [\cdot], Env\ \mathtt{in}\ R^-[x]))$$

$$\mathtt{uw}(x, (\mathtt{letrec}\ y \stackrel{\bullet}{=} R^-, x = s, Env\ \mathtt{in}\ t))$$
$$\to \mathtt{uw}(s, (\mathtt{letrec}\ y \stackrel{\bullet}{=} R^-[x], x \stackrel{\bullet}{=} [\cdot], Env\ \mathtt{in}\ t))$$

$\mathtt{uw}(s, R) \to (s, R)$ falls keine andere Regel anwendbar

Beispiel: $\mathtt{letrec}\ x_2 = \lambda x.x, x_1 \stackrel{\bullet}{=} (x_2\ x_1), x_3 = (\mathtt{amb}\ (x_2\ x_1)\ y)\ \mathtt{in}\ \mathtt{amb}\ x_1\ x_3$

Normalordnungsreduktion

$\xrightarrow{\mathbf{no}}$ = Anwenden einer Reduktionsregel "mithilfe" eines maximalen Reduktionskontextes

Maximalen Reduktionkontext finden: Unwinding-Algorithmus $\mathcal{UW}$

$$\mathtt{uw}((s\ t), R) \to \mathtt{uw}(s, R[([\cdot]\ t)])$$

$$\mathtt{uw}(\mathtt{seq}\ s\ t, R) \to \mathtt{uw}(s, R[\mathtt{seq}\ [\cdot]\ t])$$

$$\mathtt{uw}(\mathtt{case}_T\ s\ alts, R) \to \mathtt{uw}(s, R[\mathtt{case}_T\ [\cdot]\ alts])$$

$$\mathtt{uw}(\mathtt{amb}\ s\ t, R) \to \mathtt{uw}(s, R[\mathtt{amb}\ [\cdot]\ t])\ \textbf{oder}\ \mathtt{uw}(t, R[\mathtt{amb}\ s\ [\cdot]])$$

$$\mathtt{uw}(x, (\mathtt{letrec}\ x = s, Env\ \mathtt{in}\ R^-)) \to \mathtt{uw}(s, (\mathtt{letrec}\ x \stackrel{\bullet}{=} [\cdot], Env\ \mathtt{in}\ R^-[x]))$$

$$\mathtt{uw}(x, (\mathtt{letrec}\ y \stackrel{\bullet}{=} R^-, x = s, Env\ \mathtt{in}\ t))$$
$$\to \mathtt{uw}(s, (\mathtt{letrec}\ y \stackrel{\bullet}{=} R^-[x], x \stackrel{\bullet}{=} [\cdot], Env\ \mathtt{in}\ t))$$

$\mathtt{uw}(s, R) \to (s, R)$ falls keine andere Regel anwendbar

Beispiel: $\mathtt{letrec}\ x_2 \stackrel{\bullet}{=} \lambda x.x, x_1 \stackrel{\bullet}{=} (x_2\ x_1), x_3 = (\mathtt{amb}\ (x_2\ x_1)\ y)\ \mathtt{in}\ \mathtt{amb}\ x_1\ x_3$

May- und Must-Konvergenz

Weak Head Normal Form (WHNF)

- Werte (Abstraktionen, Konstruktoranwendungen)
- $\texttt{letrec}\ Env\ \texttt{in}\ v$, wobei v ein Wert ist
- $\texttt{letrec}\ x_1 = (c_{T,i}\ s_1\ \ldots\ s_n), x_2 = x_1, \ldots, x_m = x_{m-1}, Env\ \texttt{in}\ x_m$

May-Konvergenz

$s{\downarrow}$ gdw. $\exists t : s \xrightarrow{\mathbf{no},*} t, t$ WHNF

"reduzibel zu einer WHNF"

Must-Divergenz

$s{\Uparrow}$ gdw. $\neg s{\downarrow}$

"nicht reduzibel zu einer WHNF"

May- und Must-Konvergenz

Weak Head Normal Form (WHNF)

- Werte (Abstraktionen, Konstruktoranwendungen)
- `letrec` Env `in` v, wobei v ein Wert ist
- `letrec` $x_1 = (c_{T,i}\ s_1\ \ldots\ s_n), x_2 = x_1, \ldots, x_m = x_{m-1}, Env$ `in` x_m

May-Konvergenz

$s{\downarrow}$ gdw. $\exists t : s \xrightarrow{\textbf{no},*} t, t$ WHNF

"reduzibel zu einer WHNF"

Must-Divergenz

$s{\Uparrow}$ gdw. $\neg s{\downarrow}$

"nicht reduzibel zu einer WHNF"

Must-Konvergenz

$s{\Downarrow}$ gdw. $\forall t : s \xrightarrow{\textbf{no},*} t \implies t{\downarrow}$

"jeder Nachfolger ist may-konvergent"

May-Divergenz

$s{\uparrow}$ gdw. $\neg s{\Downarrow}$

$= \exists t : s \xrightarrow{\textbf{no},*} t, t{\Uparrow}$

"reduzibel zu must-divergentem Term"

Normalordungsreduktion ist nicht fair

Fairness

in einer unendlichen Reduktionssequenz, wird jeder Redex nach einer endlichen Anzahl von Schritten reduziert

Normalordnungsreduktion kann einen Redex unendlich oft ignorieren:

$$
\begin{array}{c}
\texttt{amb}\ \Omega_1\ \texttt{True} \\
\downarrow \textbf{no} \\
\texttt{amb}\ \Omega_2\ \texttt{True} \\
\downarrow \textbf{no} \\
\texttt{amb}\ \Omega_3\ \texttt{True} \\
\downarrow \textbf{no} \\
\texttt{amb}\ \Omega_4\ \texttt{True} \\
\downarrow \textbf{no} \\
\texttt{amb}\ \Omega_5\ \texttt{True} \\
\downarrow \textbf{no} \\
\ldots
\end{array}
$$

Normalordungsreduktion ist nicht fair

Fairness

in einer unendlichen Reduktionssequenz, wird jeder Redex nach einer endlichen Anzahl von Schritten reduziert

Normalordnungsreduktion kann einen Redex unendlich oft ignorieren:

$$
\begin{tikzcd}
\texttt{amb}\ \Omega_1\ \texttt{True} \arrow[d, "\mathbf{no}"] \arrow[ddr, "\mathbf{no}"] & \\
\texttt{amb}\ \Omega_2\ \texttt{True} \arrow[d, "\mathbf{no}"] \arrow[dr, "\mathbf{no}"] & \\
\texttt{amb}\ \Omega_3\ \texttt{True} \arrow[d, "\mathbf{no}"] \arrow[r, "\mathbf{no}"] & \texttt{True} \\
\texttt{amb}\ \Omega_4\ \texttt{True} \arrow[d, "\mathbf{no}"] \arrow[ur, "\mathbf{no}"] & \\
\texttt{amb}\ \Omega_5\ \texttt{True} \arrow[d, "\mathbf{no}"] \arrow[uur, "\mathbf{no}"] & \\
\ldots &
\end{tikzcd}
$$

Normalordungsreduktion ist nicht fair

Fairness

in einer unendlichen Reduktionssequenz, wird jeder Redex nach einer endlichen Anzahl von Schritten reduziert

Normalordnungsreduktion kann einen Redex unendlich oft ignorieren:

$$
\begin{tikzcd}
\texttt{amb}\ \Omega_1\ \texttt{True} \arrow[d, "\textbf{no}"] \arrow[ddr, "\textbf{no}"] & \\
\texttt{amb}\ \Omega_2\ \texttt{True} \arrow[d, "\textbf{no}"] \arrow[dr, "\textbf{no}"] & \\
\texttt{amb}\ \Omega_3\ \texttt{True} \arrow[d, "\textbf{no}"] \arrow[r, "\textbf{no}"] & \texttt{True} \\
\texttt{amb}\ \Omega_4\ \texttt{True} \arrow[d, "\textbf{no}"] \arrow[ur, "\textbf{no}"] & \\
\texttt{amb}\ \Omega_5\ \texttt{True} \arrow[d, "\textbf{no}"] \arrow[uur, "\textbf{no}"] & \\
\dots &
\end{tikzcd}
$$

$\Longrightarrow$ verwendet man Normalordnungsreduktion so ist `amb` nicht Divergenz-vermeidend!

Fairness zusichern

Hinzufügen von Ressourcen: Annotierte Varianten

statt $\texttt{amb}\ s\ t$ nun $\texttt{amb}_{\langle m,n\rangle}\ s\ t$ mit $m,n \in \mathbb{N}_0$ (am Anfang alle 0)

Faires Unwinding mit Ressourcen

...

$$\texttt{uwf}((\texttt{amb}_{\langle m+1,n\rangle}\ s\ t), R) \to \texttt{uwf}(s, R[(\texttt{amb}_{\langle m,n\rangle}\ [\cdot]\ t)])$$

$$\texttt{uwf}((\texttt{amb}_{\langle m,n+1\rangle}\ s\ t), R) \to \texttt{uwf}(t, R[(\texttt{amb}_{\langle m,n\rangle}\ s\ [\cdot])])$$

$$\texttt{uwf}((\texttt{amb}_{\langle 0,0\rangle}\ s\ t), R) \to \texttt{uwf}((\texttt{amb}_{\langle m,n\rangle}\ s\ t), R),\ \text{wobei } m,n>0$$

...

$s \xrightarrow{\text{fno}} t$

1. Wende fairen Unwindingalgorithmus auf s an $\Rightarrow (s', R)$
2. Wenn N.O.-Reduktion auf $R[s']$ unter Benutzung von R anwendbar (Ressourcen ignorieren), dann $s \xrightarrow{\text{fno}} t$
3. Andernfalls: Gehe zu 1, mit $R[s']$ für s

Konvergenz-Äquivalenz $\xrightarrow{\mathbf{fno}}$ und $\xrightarrow{\mathbf{no}}$

Beispiel:

$$
\begin{tikzcd}
\mathtt{amb}_{\langle 0,0\rangle}\ \Omega_1\ \mathtt{True} \arrow[d, "\mathbf{fno}"] \arrow[ddr, dashed, "\mathbf{fno}"] & \\
\mathtt{amb}_{\langle m-1,n\rangle}\ \Omega_2\ \mathtt{True} \arrow[d, "\mathbf{fno}"] \arrow[dr, dashed, "\mathbf{fno}"] & \\
\mathtt{amb}_{\langle m-2,n\rangle}\ \Omega_3\ \mathtt{True} \arrow[d, "\mathbf{fno}"] \arrow[r, dashed, "\mathbf{fno}"] & \mathtt{True} \\
\ldots \arrow[d, "\mathbf{fno}"] & \\
\mathtt{amb}_{\langle 0,n\rangle}\ \Omega_{m+1}\ \mathtt{True} \arrow[uur, "\mathbf{fno}"] &
\end{tikzcd}
$$

$m, n > 0$

Faire may- und must-Konvergenz

$s{\downarrow_F}$ gdw. $\exists t : s \xrightarrow{\mathbf{fno},*} t$, t ann. WHNF

$s{\Downarrow_F}$ gdw. $\forall t : s \xrightarrow{\mathbf{fno},*} t \implies t{\downarrow_F}$

Theorem

Für alle Ausdrücke s: $s{\downarrow}$ gdw. $s{\downarrow_F}$ und $s{\Downarrow}$ gdw. $s{\Downarrow_F}$

Prinzip der kontextuellen Gleichheit

Zwei Programme sind gleich gdw.
sie sich in jedem Programmkontext gleich verhalten

Prinzip der kontextuellen Gleichheit

Zwei Programme sind gleich gdw.
sie sich in jedem Programmkontext gleich verhalten

Verhalten = may- und must-Konvergenz

Zwei kontextuelle Präordnungen

für may-Konvergenz

$$s \leq_c^{\downarrow} t \quad \text{gdw.} \quad \forall C \in \mathcal{C}: \; C[s]{\downarrow} \Rightarrow C[t]{\downarrow}$$

für must-Konvergenz

$$s \leq_c^{\Downarrow} t \quad \text{gdw.} \quad \forall C \in \mathcal{C}: \; C[s]{\Downarrow} \Rightarrow C[t]{\Downarrow}$$

Kontextuelle Präordnung / Kontextuelle Äquivalenz

$$\leq_c \; = \; \leq_c^{\downarrow} \cap \leq_c^{\Downarrow} \qquad\qquad \sim_c \; = \; \leq_c \cap \geq_c$$

Korrektheit von Programmtransformationen

- Programmtransformation P = binäre Relation über Ausdrücken
- P ist korrekt gdw. P erhält die kontextuelle Gleichheit $= P \subseteq \sim_c$

Korrektheit zu widerlegen ist einfach...

- finde Gegenbeispiel: Kontext, der die Ausdrücke unterscheidet
- **Beispiel:** $(\texttt{choice}\ \Omega\ \texttt{True}) \not\sim_c \texttt{True}$
  denn für $C = [\cdot]$ gilt $C[\texttt{True}]\Downarrow$, aber $C[(\texttt{choice}\ \Omega\ \texttt{True})]\Uparrow$,

Beweisen der Korrektheit ist wesentlich schwieriger ...

Die unendliche Menge aller Kontexte muss betrachtet werden!

Kontextlemma

"es reicht aus Reduktionskontexte zu betrachten"

$$s \leq^{\downarrow}_{c,\mathcal{R}} t \quad \text{gdw.} \quad \forall R \in \mathcal{R} : R[s]\downarrow \implies R[t]\downarrow$$

$$s \leq^{\Downarrow}_{c,\mathcal{R}} t \quad \text{gdw.} \quad \forall R \in \mathcal{R} : R[s]\Downarrow \implies R[t]\Downarrow$$

$$\leq_{c,\mathcal{R}} \quad := \quad \leq^{\downarrow}_{c,\mathcal{R}} \cap \leq^{\Downarrow}_{c,\mathcal{R}}$$

$$\boxed{\leq_{c,\mathcal{R}} \subseteq \leq_c}$$

für may-Konvergenz

$$\leq^{\downarrow}_{c,\mathcal{R}} = \leq^{\downarrow}_c$$

für must-Konvergenz

$$(\leq^{\downarrow}_{c,\mathcal{R}} \cap \leq^{\Downarrow}_{c,\mathcal{R}}) \subseteq \leq^{\Downarrow}_c$$

Korrektheitsbeweis von Programmtransformationen

Sei P eine Programmtransformation

Beweisskizze (zeige $P \subseteq \sim_c$)

Zeige für alle s, t und Reduktionskontexte R mit $R[s] \xrightarrow{P} R[t]$:

- P erhält die may-Konvergenz:
  - $R[s]{\downarrow} \implies R[t]{\downarrow}$
  - $R[t]{\downarrow} \implies R[s]{\downarrow}$
- P erhält die must-Konvergenz:
  - $R[s]{\Downarrow} \implies R[t]{\Downarrow}$
  - $R[t]{\Downarrow} \implies R[s]{\Downarrow}$

Das Kontextlemma impliziert dann die Korrektheit

Korrektheitsbeweis von Programmtransformationen

Sei P eine Programmtransformation

Beweisskizze (zeige $P \subseteq \sim_c$)

Zeige für alle s, t und Oberflächenkontexte S mit $S[s] \xrightarrow{P} S[t]$:

- P erhält die may-Konvergenz:
  - $S[s]\downarrow \implies S[t]\downarrow$
  - $S[t]\downarrow \implies S[s]\downarrow$
- P erhält die must-Konvergenz:
  - $S[s]\Downarrow \implies S[t]\Downarrow$
  - $S[t]\Downarrow \implies S[s]\Downarrow$

Das Kontextlemma impliziert dann die Korrektheit

Korrektheitsbeweis von Programmtransformationen

Sei P eine Programmtransformation

Beweisskizze (zeige $P \subseteq \sim_c$)

Zeige für alle s, t mit $s \xrightarrow{\mathcal{S},P} t$:

- P erhält die may-Konvergenz:
  - $s\downarrow \implies t\downarrow$
  - $t\downarrow \implies s\downarrow$
- P erhält die must-Konvergenz:
  - $s\Downarrow \implies t\Downarrow$
  - $t\Downarrow \implies s\Downarrow$

Das Kontextlemma impliziert dann die Korrektheit

Gabel- und Vertauschungsdiagramme für P

Diagramme sind meta-rewriting Regeln

$\widehat{P} \subseteq P$ $\quad \widehat{\mathbf{no}}, \widetilde{\mathbf{no}} \subseteq \mathbf{no}$ $\quad$ rel binäre Relation auf Ausdrücken $\quad k_1 + k_2 > 0$

$$\begin{tikzcd} \cdot \arrow[r, "{\mathcal{S},\widehat{P}}"] \arrow[d, "{\widehat{\mathbf{no}},\leq 1}"'] & \cdot \arrow[d, dashed, "{\widetilde{\mathbf{no}},k_2}"] \\ \cdot \arrow[r, dashed, "{rel,k_1}"'] & \cdot \end{tikzcd}$$

Gabeldiagramm

$$\begin{tikzcd} \cdot \arrow[r, "{i\mathcal{S},\widehat{P}}"] \arrow[d, dashed, "{\widetilde{\mathbf{no}},k_2}"'] & \cdot \arrow[d, "{\widehat{\mathbf{no}},\leq 1}"] \\ \cdot \arrow[r, dashed, "{rel,k_1}"'] & \cdot \end{tikzcd}$$

Vertauschungsdiagramm

Gabel- und Vertauschungsdiagramme für P

Diagramme sind meta-rewriting Regeln

$\widehat{P} \subseteq P$ $\widehat{\mathbf{no}}, \widetilde{\mathbf{no}} \subseteq \mathbf{no}$ rel binäre Relation auf Ausdrücken $k_1 + k_2 > 0$

$$\left\{ \begin{tikzcd} \cdot \arrow[r, "\mathcal{S}{,}\widehat{P}"] \arrow[d, "\widehat{\mathbf{no}}{,}\leq 1"'] & \cdot \arrow[d, dashed, "\widetilde{\mathbf{no}}{,}k_2"] \\ \cdot \arrow[r, dashed, "rel{,}k_1"'] & \cdot \end{tikzcd} \right\}$$

Menge von Gabeldiagrammen ist vollständig gdw. es für jede

$$\begin{tikzcd} s \arrow[r, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & t \\ r & \end{tikzcd}$$

ein anwendbares Diagramm gibt

$$\left\{ \begin{tikzcd} \cdot \arrow[r, "i\mathcal{S}{,}\widehat{P}"] \arrow[d, dashed, "\widetilde{\mathbf{no}}{,}k_2"'] & \cdot \arrow[d, "\widehat{\mathbf{no}}{,}\leq 1"] \\ \cdot \arrow[r, dashed, "rel{,}k_1"'] & \cdot \end{tikzcd} \right\}$$

Menge von Vert.-diagrammen ist vollständig gdw. es für jede

$$\begin{tikzcd} s \arrow[r, "i\mathcal{S}{,}P"] & t \arrow[d, "\mathbf{no}"] \\ & r \end{tikzcd}$$

ein anwendbares Diagramm gibt

Erhaltung der may-Konvergenz

zeige $s\downarrow \implies t\downarrow$

$$
\begin{tikzcd}
s \arrow[r, "\mathcal{S},P"] \arrow[d, "\mathbf{no}"'] & t \\
\cdot \arrow[d, "\mathbf{no}"'] & \\
\cdot \arrow[d, "\mathbf{no}"'] & \\
s' &
\end{tikzcd}
$$

WHNF

Erhaltung der may-Konvergenz

zeige $s\!\downarrow \implies t\!\downarrow$

$$
\begin{tikzcd}
s \arrow[r, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & t \arrow[d, dashed, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & \cdot \\
\cdot \arrow[d, "\mathbf{no}"'] & \\
s' &
\end{tikzcd}
$$

WHNF

Erhaltung der may-Konvergenz

zeige $s{\downarrow} \implies t{\downarrow}$

$$
\begin{tikzcd}
s \arrow[r, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & t \arrow[d, dashed, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & \cdot \arrow[d, dashed, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & \cdot \arrow[d, dashed, "\mathbf{no}"] \\
s' \arrow[r, dashed, "\mathcal{S}{,}P"] & t'
\end{tikzcd}
$$

WHNF WHNF

Erhaltung der may-Konvergenz

zeige $s\downarrow \implies t\downarrow$

$$\begin{tikzcd}
s \arrow[r, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & t \arrow[d, dashed, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & \cdot \arrow[d, dashed, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & \cdot \arrow[d, dashed, "\mathbf{no}"] \\
s' \arrow[r, dashed, "\mathcal{S}{,}P"] & t'
\end{tikzcd}$$

WHNF WHNF

zeige $t\downarrow \implies s\downarrow$

$$\begin{tikzcd}
s \arrow[r, "i\mathcal{S}{,}P"] & t \arrow[d, "\mathbf{no}"] \\
& \cdot \arrow[d, "\mathbf{no}"] \\
& \cdot \arrow[d, "\mathbf{no}"] \\
& t'
\end{tikzcd}$$

WHNF

Erhaltung der may-Konvergenz

zeige $s\downarrow \implies t\downarrow$

$$
\begin{tikzcd}
s \arrow[r, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & t \arrow[d, dashed, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & \cdot \arrow[d, dashed, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "\mathcal{S}{,}P"] \arrow[d, "\mathbf{no}"'] & \cdot \arrow[d, dashed, "\mathbf{no}"] \\
s' \arrow[r, dashed, "\mathcal{S}{,}P"] & t'
\end{tikzcd}
$$

WHNF WHNF

zeige $t\downarrow \implies s\downarrow$

$$
\begin{tikzcd}
s \arrow[r, "i\mathcal{S}{,}P"] \arrow[d, dashed, "\mathbf{no}"'] & t \arrow[d, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "i\mathcal{S}{,}P"] \arrow[d, dashed, "\mathbf{no}"'] & \cdot \arrow[d, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "i\mathcal{S}{,}P"] \arrow[d, dashed, "\mathbf{no}"'] & \cdot \arrow[d, "\mathbf{no}"] \\
s' \arrow[r, dashed, "i\mathcal{S}{,}P"] & t'
\end{tikzcd}
$$

WHNF WHNF

Korrektheitbeweis einer Transformation

Beweisskizze

Zeige für alle s, t mit $s \xrightarrow{\mathcal{S},P} t$:

- P erhält die may-Konvergenz: ✓
  - $s{\downarrow} \implies t{\downarrow}$
  - $t{\downarrow} \implies s{\downarrow}$
- P erhält die must-Konvergenz: ?
  - $s{\Downarrow} \implies t{\Downarrow}$
  - $t{\Downarrow} \implies s{\Downarrow}$

Korrektheitbeweis einer Transformation

Beweisskizze

Zeige für alle s, t mit $s \xrightarrow{\mathcal{S},P} t$:

- P erhält die may-Konvergenz:
  - $s\downarrow \implies t\downarrow$
  - $t\downarrow \implies s\downarrow$
- P erhält die must-Konvergenz (= erhält die may-Divergenz):
  - $s\Downarrow \implies t\Downarrow$ **äquivalent zu** $t\uparrow \implies s\uparrow$
  - $t\Downarrow \implies s\Downarrow$ **äquivalent zu** $s\uparrow \implies t\uparrow$

$s\uparrow$ äquivalenz zu $\exists t : s \xrightarrow{\textbf{no},*} t \wedge t\Uparrow$

Erhaltung der must-Konvergenz

zeige $s{\uparrow} \implies t{\uparrow}$

$$\begin{tikzcd}[row sep=large, column sep=huge]
s \arrow[r, "\mathcal{S},P"] \arrow[d, "\mathbf{no}"'] & t \arrow[d, dashed, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "\mathcal{S},P"] \arrow[d, "\mathbf{no}"'] & \cdot \arrow[d, dashed, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "\mathcal{S},P"] \arrow[d, "\mathbf{no}"'] & \cdot \arrow[d, dashed, "\mathbf{no}"] \\
s' \arrow[r, dashed, "\mathcal{S},P"] & t'
\end{tikzcd}$$

must-divergent | must-divergent

zeige $t{\uparrow} \implies s{\uparrow}$

$$\begin{tikzcd}[row sep=large, column sep=huge]
s \arrow[r, "i\mathcal{S},P"] \arrow[d, dashed, "\mathbf{no}"'] & t \arrow[d, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "i\mathcal{S},P"] \arrow[d, dashed, "\mathbf{no}"'] & \cdot \arrow[d, "\mathbf{no}"] \\
\cdot \arrow[r, dashed, "i\mathcal{S},P"] \arrow[d, dashed, "\mathbf{no}"'] & \cdot \arrow[d, "\mathbf{no}"] \\
s' \arrow[r, dashed, "i\mathcal{S},P"] & t'
\end{tikzcd}$$

must-divergent | must-divergent

Diagramme sind im Allgemeinen komplexer

$$
\begin{tikzcd}
\cdot \arrow[r, "{\mathcal{S},\mathsf{lll}}"] \arrow[d, "{\mathbf{no},a}"'] & \cdot \arrow[d, dashed, "{\mathbf{no},a}"] \\
\cdot \arrow[r, dashed, "{\mathcal{S},\mathsf{lll}}"'] & \cdot
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[r, "{\mathcal{S},\mathsf{lll}}"] \arrow[d, "{\mathbf{no},a}"'] & \cdot \arrow[dl, dashed, "{\mathbf{no},a}"] \\
\cdot &
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[r, "{\mathcal{S},\mathsf{lll}}"] \arrow[d, "{\mathbf{no},\mathsf{lll}}"'] & \cdot \arrow[d, dashed, "{\mathbf{no},\mathsf{lll},+}"] \\
\cdot \arrow[r, dashed, "{\mathbf{no},\mathsf{lll},+}"'] & \cdot
\end{tikzcd}
$$

$$
\begin{tikzcd}
\cdot \arrow[r, "{\mathbf{no},\mathsf{lamb}}"] \arrow[dd, "{\mathbf{no},a}"'] & \cdot \arrow[d, dashed, "{\mathbf{no},\mathsf{lll},+}"] \\
 & \cdot \arrow[d, dashed, "{\mathbf{no},a}"] \\
\cdot \arrow[r, dashed, "{\mathbf{no},\mathsf{lll},+}"'] & \cdot
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[r, "{\mathcal{S},\mathsf{lamb}}"] \arrow[dd, "{\mathbf{no},\mathsf{amb}}"'] & \cdot \arrow[d, dashed, "{\mathbf{no},\mathsf{lll},*}"] \\
 & \cdot \arrow[d, dashed, "{\mathbf{no},\mathsf{amb}}"] \\
\cdot \arrow[r, dashed, "{\mathcal{S},\mathsf{gc}^{-1}}"'] & \cdot
\end{tikzcd}
$$

$\Longrightarrow$ speziellere Induktionsmaße und stärkere Behauptungen

Ergebnisse: Korrekte Programmtransformationen

Alle deterministischen Kalkülregeln (d.h. partielle Auswertung ist korrekt):

(lbeta) $(\lambda x.s)\ r \longrightarrow (\texttt{letrec}\ x = r\ \texttt{in}\ s)$

(cp) $\texttt{letrec}\ x_1 = \lambda x.s, x_2 = x_1 \ldots, x_m = x_{m-1} \ldots C[x_m] \ldots$
$\longrightarrow \texttt{letrec}\ x_1 = \lambda x.s, x_2 = x_1, \ldots x_m = x_{m-1} \ldots C[\lambda x.s] \ldots$

(seq)
- $\texttt{seq}\ v\ t \longrightarrow t$ (v Wert)
- $\texttt{letrec}\ x_1 = v, x_2 = x_1 \ldots, x_m = x_{m-1} \ldots C[\texttt{seq}\ x_m\ t] \ldots$ (v Wert)
  $\longrightarrow \texttt{letrec}\ x_1 = (\lambda x.s), x_2 = x_1, \ldots x_m = x_{m-1} \ldots C[t] \ldots$

(llet-in) $(\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ r)) \longrightarrow (\texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ r)$

(llet-e) $(\texttt{letrec}\ x_1 = s_1, \ldots, x_i = (\texttt{letrec}\ Env_2\ \texttt{in}\ s_i), \ldots, x_n = s_n\ \texttt{in}\ r)$
$\longrightarrow (\texttt{letrec}\ x_1 = s_1, \ldots, x_i = s_i, \ldots, x_n = s_n, Env_2\ \texttt{in}\ r)$

(lapp) $((\texttt{letrec}\ Env\ \texttt{in}\ t)\ s) \longrightarrow (\texttt{letrec}\ Env\ \texttt{in}\ (t\ s))$

(lcase) $(\texttt{case}_T\ (\texttt{letrec}\ Env\ \texttt{in}\ t)\ alts) \longrightarrow (\texttt{letrec}\ Env\ \texttt{in}\ (\texttt{case}_T\ t\ alts))$

(lseq) $(\texttt{seq}\ (\texttt{letrec}\ Env\ \texttt{in}\ s)\ t) \longrightarrow (\texttt{letrec}\ Env\ \texttt{in}\ (\texttt{seq}\ s\ t))$

(lamb-l) $(\texttt{amb}\ (\texttt{letrec}\ Env\ \texttt{in}\ s)\ t) \longrightarrow (\texttt{letrec}\ Env\ \texttt{in}\ (\texttt{amb}\ s\ t))$

(lamb-r) $(\texttt{amb}\ s\ (\texttt{letrec}\ Env\ \texttt{in}\ t)) \longrightarrow (\texttt{letrec}\ Env\ \texttt{in}\ (\texttt{amb}\ s\ t))$

(case)
- $\texttt{case}_T\ c_{T,i}\ \ldots (c_{T,i} \rightarrow t) \ldots \longrightarrow t$
- $\texttt{letrec}\ x_1 = c_{T,i}, x_2 = x_1, \ldots, x_m = x_{m-1} \ldots\ C[\texttt{case}_T\ x_m\ \ldots\ (c_{T,i} \rightarrow t)\ \ldots] \ldots$
  $\longrightarrow \texttt{letrec}\ x_1 = c_{T,i}, x_2 = x_1, \ldots, x_m = x_{m-1} \ldots\ C[t] \ldots$
- $\texttt{case}_T\ (c_{T,i}\ t_1 \ldots t_n)\ \ldots ((c_{T,i}\ y_1 \ldots y_n) \rightarrow t) \ldots \longrightarrow \texttt{letrec}\ y_1 = t_1, \ldots, y_n = t_n\ \texttt{in}\ t$
- $\texttt{letrec}\ x_1 = (c_{T,i}\ t_1 \ldots t_n), x_2 = x_1, \ldots, x_m = x_{m-1} \ldots C[\texttt{case}_T\ x_m \ldots ((c_{T,i}\ z_1 \ldots z_n) \rightarrow t)\ \ldots]$
  $\longrightarrow \texttt{letrec}\ x_1 = (c_{T,i}\ y_1 \ldots y_n), y_1 = t_1, \ldots, y_n = t_n, x_2 = x_1, \ldots, x_m = x_{m-1}$
  $\ldots\ C[\texttt{letrec}\ z_1 = y_1, \ldots, z_n = y_n\ \texttt{in}\ t] \ldots$

Ergebnisse: Korrekte Programmtransformationen

- Garbage collection
  - $\texttt{letrec}\ x_1 = s_1, \ldots, x_n = s_n\ \texttt{in}\ t \to t$ wenn $x_i \notin FV(t)$
  - $\texttt{letrec}\ x_1 = s_1, \ldots, x_n = s_n, Env\ \texttt{in}\ t \to \texttt{letrec}\ Env\ \texttt{in}\ t$ wenn $x_i \notin FV(t)$
- Kopieren von Variablen und Konstruktoren
  - $\texttt{letrec}\ x = y, \ldots C[x] \ldots \to \texttt{letrec}\ x = y, \ldots C[y] \ldots$
  - $\texttt{letrec}\ x = (c\ \overrightarrow{s_i}), \ldots C[x] \ldots \to \texttt{letrec}\ x = (c\ \overrightarrow{y_i}), \{y_i = s_i\}_{i=1}^{\mathrm{ar}(c)}, \ldots C[(c\ \overrightarrow{y_i})] \ldots$
- Kopieren von Ausdrücken mit nur einem Vorkommen (Inlining)
  - $\texttt{letrec}\ x = s, \ldots S[x] \ldots \to \texttt{letrec}\ \ldots S[s] \ldots$, wenn x nur einmal vorkommt
  - $\texttt{letrec}\ x = s\ \texttt{in}\ S[x] \to S[s]$, wenn x nur einmal vorkommt
- und einige mehr ((abs), (xch), (lappr), (lseqr), (lcons))

Standardisierungstheorem

- Wenn $t \xrightarrow{\mathcal{C},(\text{ptc}\vee\text{amb}),*} t'$ wobei t' eine WHNF ist, dann gilt $t\downarrow$.
- Wenn $t \xrightarrow{(\mathcal{C},\text{ptc})\vee(\mathcal{S},\text{ambs}),*} t'$ wobei $t'\Uparrow$, dann gilt $t\uparrow$.

wobei

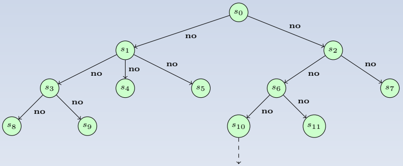
- ptc korrekte Programmtransformationen
- (ambs) eingeschränkte (amb)-Reduktion
  $\texttt{letrec}\ x_1 = v, x_2 = x_1 \ldots, x_m = x_{m-1} \ldots S[\texttt{amb}\ x_m\ t] \ldots$
  $\longrightarrow \texttt{letrec}\ x_1 = (\lambda x.s), x_2 = x_1, \ldots x_m = x_{m-1} \ldots S[x_m] \ldots$

Beweis erfordert eine Untersuchung der (amb)-Reduktion:

	$\leq_c^{\downarrow}$	$\leq_c^{\Downarrow}$	$\geq_c^{\downarrow}$	$\geq_c^{\Downarrow}$	$\leq_{c,\mathcal{S}}^{\Downarrow}$
(amb)	$\not\subseteq$	$\not\subseteq$	$\subseteq$	$\not\subseteq$	$\not\subseteq$
(ambs)	$\not\subseteq$	$\not\subseteq$	$\subseteq$	$\not\subseteq$	$\subseteq$

Endliche Simulation mit vollständigen Nachfolgermengen

Vollständige Nachfolgermenge $M \in \mathcal{CSS}(s)$ eines Ausdrucks $s :=$
alle Blätter eines gültigen Schnitts des Auswertungsbaums



Kontextuelle Äquivalenz auf endl. Mengen

M, N endliche Mengen von Ausdrücken:

$M \ \langle \leq^{\downarrow} \rangle \ N$ gdw. $(\forall s \in M : \exists t \in N : s \leq_c^{\downarrow} t)$

$M \ \langle \leq^{\Downarrow} \rangle \ N$ gdw. $(\forall t \in N : \exists s \in M : s \leq_c^{\Downarrow} t)$

$\langle \leq \rangle := \langle \leq^{\downarrow} \rangle \cap \langle \leq^{\Downarrow} \rangle, \quad \langle \sim \rangle := \langle \leq \rangle \cap \langle \geq \rangle$

Theorem Seien s, t geschlossene Ausdrücke, $M \in \mathcal{CSS}(s)$, $N \in \mathcal{CSS}(t)$.

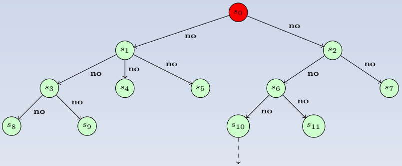
$$M \ \langle \leq \rangle \ N \quad \implies \quad s \leq_c t.$$

Beispiel $s :=$ (`amb True False`) und $t :=$ (`amb False True`)

$\{\texttt{True}, \texttt{False}\} \in \mathcal{CSS}(s) \cap \mathcal{CSS}(t)$, d.h. $s \sim_c t$

Endliche Simulation mit vollständigen Nachfolgermengen

Vollständige Nachfolgermenge $M \in \mathcal{CSS}(s)$ eines Ausdrucks $s :=$
alle Blätter eines gültigen Schnitts des Auswertungsbaums



Kontextuelle Äquivalenz auf endl. Mengen

M, N endliche Mengen von Ausdrücken:

$M\ \langle\leq^{\downarrow}\rangle\ N$ gdw. $(\forall s \in M : \exists t \in N : s \leq_c^{\downarrow} t)$

$M\ \langle\leq^{\Downarrow}\rangle\ N$ gdw. $(\forall t \in N : \exists s \in M : s \leq_c^{\Downarrow} t)$

$\langle\leq\rangle := \langle\leq^{\downarrow}\rangle \cap \langle\leq^{\Downarrow}\rangle, \quad \langle\sim\rangle := \langle\leq\rangle \cap \langle\geq\rangle$

Theorem Seien s, t geschlossene Ausdrücke, $M \in \mathcal{CSS}(s)$, $N \in \mathcal{CSS}(t)$.

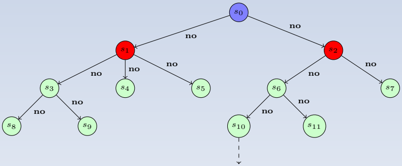
$$M\ \langle\leq\rangle\ N \implies s \leq_c t.$$

Beispiel $s :=$ (amb True False) und $t :=$ (amb False True)

$\{\texttt{True}, \texttt{False}\} \in \mathcal{CSS}(s) \cap \mathcal{CSS}(t)$, d.h. $s \sim_c t$

Endliche Simulation mit vollständigen Nachfolgermengen

Vollständige Nachfolgermenge $M \in \mathcal{CSS}(s)$ eines Ausdrucks $s :=$
alle Blätter eines gültigen Schnitts des Auswertungsbaums



Kontextuelle Äquivalenz auf endl. Mengen

M, N endliche Mengen von Ausdrücken:

$M \ \langle \leq^{\downarrow} \rangle \ N$ gdw. $(\forall s \in M : \exists t \in N : s \leq_c^{\downarrow} t)$

$M \ \langle \leq^{\Downarrow} \rangle \ N$ gdw. $(\forall t \in N : \exists s \in M : s \leq_c^{\Downarrow} t)$

$\langle \leq \rangle := \langle \leq^{\downarrow} \rangle \cap \langle \leq^{\Downarrow} \rangle, \quad \langle \sim \rangle := \langle \leq \rangle \cap \langle \geq \rangle$

Theorem Seien s, t geschlossene Ausdrücke, $M \in \mathcal{CSS}(s)$, $N \in \mathcal{CSS}(t)$.

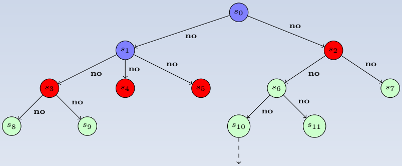
$$M \ \langle \leq \rangle \ N \quad \Longrightarrow \quad s \leq_c t.$$

Beispiel $s :=$ (amb True False) und $t :=$ (amb False True)

$\{\mathtt{True}, \mathtt{False}\} \in \mathcal{CSS}(s) \cap \mathcal{CSS}(t)$, d.h. $s \sim_c t$

Endliche Simulation mit vollständigen Nachfolgermengen

Vollständige Nachfolgermenge $M \in \mathcal{CSS}(s)$ eines Ausdrucks $s :=$
alle Blätter eines gültigen Schnitts des Auswertungsbaums



Kontextuelle Äquivalenz auf endl. Mengen

M, N endliche Mengen von Ausdrücken:

$M \ \langle \leq^{\downarrow} \rangle \ N$ gdw. $(\forall s \in M : \exists t \in N : s \leq_c^{\downarrow} t)$

$M \ \langle \leq^{\Downarrow} \rangle \ N$ gdw. $(\forall t \in N : \exists s \in M : s \leq_c^{\Downarrow} t)$

$\langle \leq \rangle := \langle \leq^{\downarrow} \rangle \cap \langle \leq^{\Downarrow} \rangle, \quad \langle \sim \rangle := \langle \leq \rangle \cap \langle \geq \rangle$

Theorem Seien s, t geschlossene Ausdrücke, $M \in \mathcal{CSS}(s)$, $N \in \mathcal{CSS}(t)$.

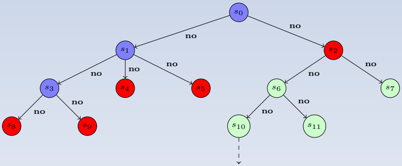
$$M \ \langle \leq \rangle \ N \quad \Longrightarrow \quad s \leq_c t.$$

Beispiel $s :=$ (`amb True False`) und $t :=$ (`amb False True`)

$\{\texttt{True}, \texttt{False}\} \in \mathcal{CSS}(s) \cap \mathcal{CSS}(t)$, d.h. $s \sim_c t$

Endliche Simulation mit vollständigen Nachfolgermengen

Vollständige Nachfolgermenge $M \in \mathcal{CSS}(s)$ eines Ausdrucks $s :=$
alle Blätter eines gültigen Schnitts des Auswertungsbaums



Kontextuelle Äquivalenz auf endl. Mengen

M, N endliche Mengen von Ausdrücken:

$M \ \langle\leq^{\downarrow}\rangle \ N$ gdw. $(\forall s \in M : \exists t \in N : s \leq_c^{\downarrow} t)$

$M \ \langle\leq^{\Downarrow}\rangle \ N$ gdw. $(\forall t \in N : \exists s \in M : s \leq_c^{\Downarrow} t)$

$\langle\leq\rangle := \langle\leq^{\downarrow}\rangle \cap \langle\leq^{\Downarrow}\rangle, \quad \langle\sim\rangle := \langle\leq\rangle \cap \langle\geq\rangle$

Theorem Seien s, t geschlossene Ausdrücke, $M \in \mathcal{CSS}(s)$, $N \in \mathcal{CSS}(t)$.

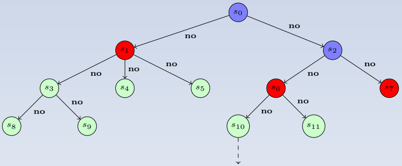
$$M \ \langle\leq\rangle \ N \quad \Longrightarrow \quad s \leq_c t.$$

Beispiel $s :=$ (`amb True False`) und $t :=$ (`amb False True`)

$\{\texttt{True}, \texttt{False}\} \in \mathcal{CSS}(s) \cap \mathcal{CSS}(t)$, d.h. $s \sim_c t$

Endliche Simulation mit vollständigen Nachfolgermengen

Vollständige Nachfolgermenge $M \in \mathcal{CSS}(s)$ eines Ausdrucks $s :=$
alle Blätter eines gültigen Schnitts des Auswertungsbaums



Kontextuelle Äquivalenz auf endl. Mengen

M, N endliche Mengen von Ausdrücken:

$M\ \langle\leq^{\downarrow}\rangle\ N$ gdw. $(\forall s \in M : \exists t \in N : s \leq_c^{\downarrow} t)$

$M\ \langle\leq^{\Downarrow}\rangle\ N$ gdw. $(\forall t \in N : \exists s \in M : s \leq_c^{\Downarrow} t)$

$\langle\leq\rangle := \langle\leq^{\downarrow}\rangle \cap \langle\leq^{\Downarrow}\rangle, \quad \langle\sim\rangle := \langle\leq\rangle \cap \langle\geq\rangle$

Theorem Seien s, t geschlossene Ausdrücke, $M \in \mathcal{CSS}(s)$, $N \in \mathcal{CSS}(t)$.

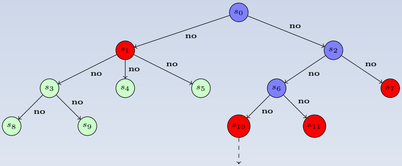
$$M\ \langle\leq\rangle\ N \quad \Longrightarrow \quad s \leq_c t.$$

Beispiel $s :=$ (`amb True False`) und $t :=$ (`amb False True`)

$\{\texttt{True}, \texttt{False}\} \in \mathcal{CSS}(s) \cap \mathcal{CSS}(t)$, d.h. $s \sim_c t$

Endliche Simulation mit vollständigen Nachfolgermengen

Vollständige Nachfolgermenge $M \in \mathcal{CSS}(s)$ eines Ausdrucks $s :=$
alle Blätter eines gültigen Schnitts des Auswertungsbaums



Kontextuelle Äquivalenz auf endl. Mengen

M, N endliche Mengen von Ausdrücken:

$M \ \langle \leq^{\downarrow} \rangle \ N$ gdw. $(\forall s \in M : \exists t \in N : s \leq_c^{\downarrow} t)$

$M \ \langle \leq^{\Downarrow} \rangle \ N$ gdw. $(\forall t \in N : \exists s \in M : s \leq_c^{\Downarrow} t)$

$\langle \leq \rangle := \langle \leq^{\downarrow} \rangle \cap \langle \leq^{\Downarrow} \rangle, \quad \langle \sim \rangle := \langle \leq \rangle \cap \langle \geq \rangle$

Theorem Seien s, t geschlossene Ausdrücke, $M \in \mathcal{CSS}(s)$, $N \in \mathcal{CSS}(t)$.

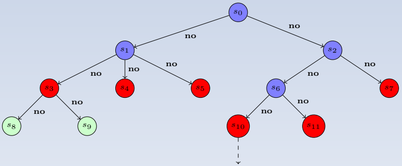
$$M \ \langle \leq \rangle \ N \quad \Longrightarrow \quad s \leq_c t.$$

Beispiel $s :=$ (`amb True False`) und $t :=$ (`amb False True`)

$\{$`True`, `False`$\} \in \mathcal{CSS}(s) \cap \mathcal{CSS}(t)$, d.h. $s \sim_c t$

Endliche Simulation mit vollständigen Nachfolgermengen

Vollständige Nachfolgermenge $M \in \mathcal{CSS}(s)$ eines Ausdrucks $s :=$
alle Blätter eines gültigen Schnitts des Auswertungsbaums



Kontextuelle Äquivalenz auf endl. Mengen

M, N endliche Mengen von Ausdrücken:

$M \ \langle \leq^{\downarrow} \rangle \ N$ gdw. $(\forall s \in M : \exists t \in N : s \leq^{\downarrow}_c t)$

$M \ \langle \leq^{\Downarrow} \rangle \ N$ gdw. $(\forall t \in N : \exists s \in M : s \leq^{\Downarrow}_c t)$

$\langle \leq \rangle := \langle \leq^{\downarrow} \rangle \cap \langle \leq^{\Downarrow} \rangle, \quad \langle \sim \rangle := \langle \leq \rangle \cap \langle \geq \rangle$

Theorem Seien s, t geschlossene Ausdrücke, $M \in \mathcal{CSS}(s)$, $N \in \mathcal{CSS}(t)$.

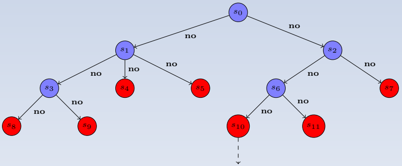
$$M \ \langle \leq \rangle \ N \quad \Longrightarrow \quad s \leq_c t.$$

Beispiel $s :=$ (`amb True False`) und $t :=$ (`amb False True`)

$\{\texttt{True}, \texttt{False}\} \in \mathcal{CSS}(s) \cap \mathcal{CSS}(t)$, d.h. $s \sim_c t$

Endliche Simulation mit vollständigen Nachfolgermengen

Vollständige Nachfolgermenge $M \in \mathcal{CSS}(s)$ eines Ausdrucks $s :=$
alle Blätter eines gültigen Schnitts des Auswertungsbaums



Kontextuelle Äquivalenz auf endl. Mengen

M, N endliche Mengen von Ausdrücken:

$M \ \langle\leq^{\downarrow}\rangle\ N$ gdw. $(\forall s \in M : \exists t \in N : s \leq_c^{\downarrow} t)$

$M \ \langle\leq^{\Downarrow}\rangle\ N$ gdw. $(\forall t \in N : \exists s \in M : s \leq_c^{\Downarrow} t)$

$\langle\leq\rangle := \langle\leq^{\downarrow}\rangle \cap \langle\leq^{\Downarrow}\rangle, \quad \langle\sim\rangle := \langle\leq\rangle \cap \langle\geq\rangle$

Theorem Seien s, t geschlossene Ausdrücke, $M \in \mathcal{CSS}(s)$, $N \in \mathcal{CSS}(t)$.

$$M\ \langle\leq\rangle\ N \quad \Longrightarrow \quad s \leq_c t.$$

Beispiel $s :=$ (`amb True False`) und $t :=$ (`amb False True`)

$\{\texttt{True}, \texttt{False}\} \in \mathcal{CSS}(s) \cap \mathcal{CSS}(t)$, d.h. $s \sim_c t$

Untersuchung der Kontextuellen Ordnung und Äquivalenz

Der "bottom-avoiding"-Kontext

$$BA \equiv (\mathtt{amb}\ (\lambda x.x)\ (\mathtt{seq}\ [\cdot]\ (\lambda x.\Omega)))\ (\lambda x.x)$$

für alle s: $BA[s]\Downarrow$ gdw. $s\Uparrow$

impliziert: "must-Konvergenz kann auf must-Divergenz testen"

Untersuchung der Kontextuellen Ordnung und Äquivalenz

Der "bottom-avoiding"-Kontext

$$BA \equiv (\texttt{amb}\ (\lambda x.x)\ (\texttt{seq}\ [\cdot]\ (\lambda x.\Omega)))\ (\lambda x.x)$$

für alle s: $BA[s]{\Downarrow}$ gdw. $s{\Uparrow}$

impliziert: "must-Konvergenz kann auf must-Divergenz testen"

Theorem

- $\leq_c^{\Downarrow} \subseteq \geq_c^{\downarrow}$
- $s \leq_c t \implies s \sim_c^{\downarrow} t$ (aber $\leq_c \not\subseteq \geq_c$)
- $s \sim_c t$ gdw. $\forall C : C[s]{\Downarrow} \iff C[t]{\Downarrow}$

Bewiesene charakteristische Gesetze

- $(\mathtt{amb}\ s\ t) \sim_c t \sim_c (\mathtt{amb}\ t\ s)$, wenn s ein Ω-Term
- $(\mathtt{dchoice}\ v_1\ v_2) \sim_c (\mathtt{amb}\ v_1\ v_2) \sim_c (\mathtt{choice}\ v_1\ v_2)$, geschlossene Werte v_1, v_2
- $(\mathtt{amb}\ v\ v) \sim_c v$, v Wert
- $(\mathtt{amb}\ s\ t) \sim_c (\mathtt{amb}\ t\ s)$
- $(\mathtt{amb}\ v_1\ (\mathtt{amb}\ v_2\ v_3)) \sim_c \mathtt{amb}((\mathtt{amb}\ v_1\ v_2)\ v_3)$, v_i geschlossene Werte
- $(\mathtt{choice}\ s\ t) \sim_c (\mathtt{choice}\ t\ s)$, s, t geschlossen
- $(\mathtt{choice}\ s\ s) \sim_c s$, s geschlossen
- $(\mathtt{choice}\ s_1\ (\mathtt{choice}\ s_2\ s_3)) \sim_c (\mathtt{choice}(\mathtt{choice}\ s_1\ s_2)\ s_3)$, s_i geschlossen
- $(\mathtt{pconv}\ s\ t\ r) \sim_c r$, s, t geschlossen, $s{\Downarrow} \vee t{\Downarrow}$
- $(\mathtt{por}\ s\ \mathtt{True}) \sim_c \mathtt{True} \sim_c (\mathtt{por}\ \mathtt{True}\ s)$
- $(\mathtt{por}\ \mathtt{False}\ \mathtt{False}) \sim_c \mathtt{False}$

Abstrakte Maschinensemantik

Vorteile einer abstrakten Maschinensemantik

- Maschinentransitionen kleinschrittiger als Normalordnungsreduktionen
- Unwinding explizit durch Maschinentransitionen modelliert
- Ergebnisse des Unwinding auf Stacks "gespeichert"
- einfach implementierbares Modell

Die "Concurrent Abstract Machine" CAM

Basis:
- Sestoft's Maschine **mark 1** [Sestoft, 1997] für verzögerte Auswertung
- Moran's Erweiterung der mark 1 um **Nebenläufigkeit** [Moran, 1998]

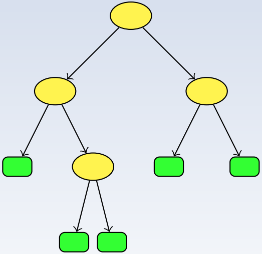
aber:
- Anpassungen gegenüber Moran, da seine Maschine nicht korrekt
- hierachische Organisation der Threads mittels Prozessbäumen

Syntax:
- eingeschränkt: nur Variablen als Argument von (Konstruktor-) Applikationen
- Übersetzung: $(s\ t) \to \texttt{letrec}\ x = t\ \texttt{in}\ (s\ x)$, bzw.
  $(c\ s_1 \ldots s_n) \to \texttt{letrec}\ x_1 = s_1, \ldots x_n = s_n\ \texttt{in}\ (c\ x_1 \ldots x_n)$

Zustand der Maschine

Paar (Γ, PT) wobei

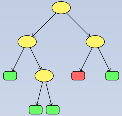
- Γ ist ein Heap: Menge von Bindungen von Variablen an Ausdrücken $x \mapsto s$ und gesperrten Bindungen $x \not\mapsto$.
- Ein Prozessbaum PT: binärer Baum wobei



- innere Knoten mit Stacks markiert sind
- Blätter mit Threads markiert sind
- ein Thread ist ein Paar (t, S) wobei t ein Ausdruck und S ein Stack ist
- Stackelemente sind $\{\#_{\text{APP}}(x), \#_{\text{SEQ}}(s), \#_{\text{CASE}}(alts), \#_{\text{UPD}}(x)\}$

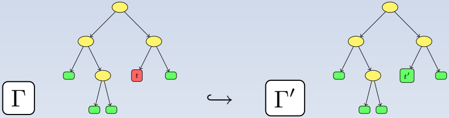
Zustandstransition

- Blatt (Thread) nichtdeterministisch auswählen
- Gewählter Thread steuert Transition

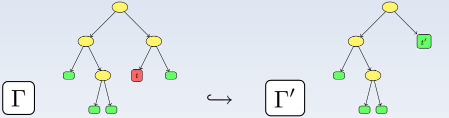


3 Arten an Transitionen:

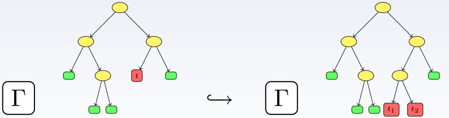
- Änderung des gewählten Threads und evtl. Heaps



- Änderung des Teilbaums direkt über dem gewählten Blatt



- Aufspaltung in 2 neue Threads



Zustandstransition

Deterministische Transitionen, nur auf Thread-Ebene

$$\boxed{((s\ x), S)} \xrightarrow{\text{pushApp}} \boxed{(s, \#_{\text{APP}}(x) : S)}$$

$$\boxed{((\lambda y.s), \#_{\text{APP}}(x) : S)} \xrightarrow{\text{takeApp}} \boxed{(s[x/y], S)}$$

$$\boxed{((\mathbf{seq}\ s\ t), S)} \xrightarrow{\text{pushSeq}} \boxed{(s, \#_{\text{SEQ}}(t) : S)}$$

$$\boxed{(v, \#_{\text{SEQ}}(t) : S)} \xrightarrow{\text{takeSeq}} \boxed{(t, S)} \quad \text{falls } v \text{ Wert}$$

$$\boxed{((\mathbf{case}_T\ s\ alts), S)} \xrightarrow{\text{pushAlts}} \boxed{(s, \#_{\text{CASE}}(T, alts) : S)}$$

$$\boxed{(c_{T,i}\ x_1 \ldots\ x_n, \#_{\text{CASE}}(T, alts) : S)} \xrightarrow{\text{takeAlts}} \boxed{(t_i[x_i/y_i], S)} \quad \text{falls } (c_{T,i}\ y_1 \ldots y_n) \to t_i \in alts$$

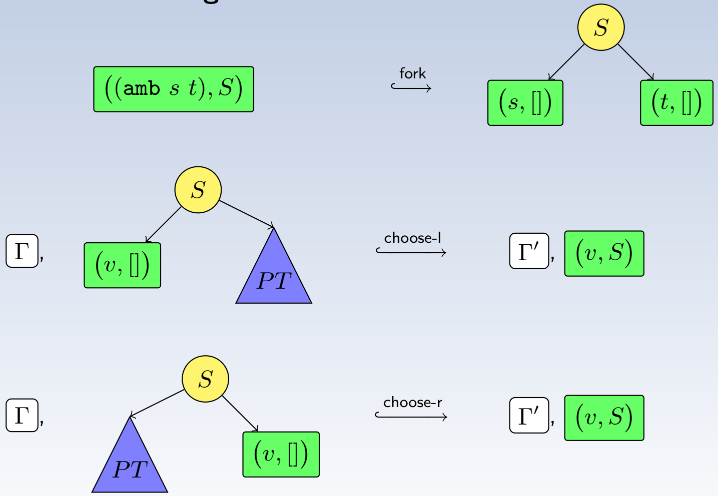
Transitionen die den Heap benutzen

$$\boxed{\{x \mapsto s\} \dot\cup \Gamma}, \boxed{(x, S)} \xrightarrow{\text{enter}} \boxed{\{x \not\mapsto\} \dot\cup \Gamma}, \boxed{(s, \#_{\text{UPD}}(x) : S)}$$

$$\boxed{\{x \not\mapsto\} \dot\cup \Gamma}, \boxed{(v, \#_{\text{UPD}}(x) : S)} \xrightarrow{\text{update}} \boxed{\{x \mapsto v\} \dot\cup \Gamma}, \boxed{(v, S)} \quad \text{falls } v \text{ Wert}$$

Zustandstransition

amb-Auswertung



Beispiel

Heap: $\emptyset$

$$\big(\mathtt{letrec}\ x = (\mathtt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1))\ \mathtt{in}\ (\mathtt{amb}\ (\lambda z.z)\ x)\ x, []\big)$$

$$\xhookrightarrow{\text{mkBinds}}$$

Beispiel

Heap: $\{x \mapsto (\mathtt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1))\}$

$$\boxed{((\mathtt{amb}\ (\lambda z.z)\ x)\ x, [])}$$

$$\xhookrightarrow{\text{pushApp}}$$

Beispiel

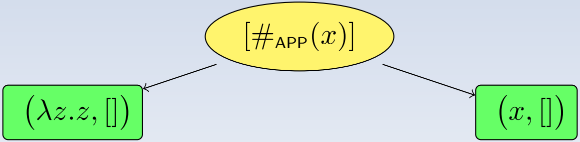
$$\boxed{\text{Heap: } \{x \mapsto (\texttt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1))\}}$$

$$\boxed{((\texttt{amb}\ (\lambda z.z)\ x), [\#_{\text{APP}}(x)])}$$

$$\xhookrightarrow{\text{fork}}$$

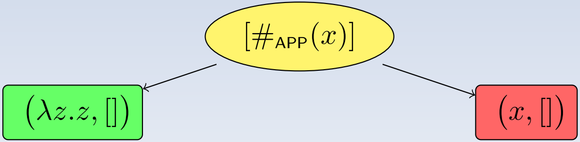
Beispiel

Heap: $\{x \mapsto (\texttt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1))\}$



Beispiel

Heap: $\{x \mapsto (\texttt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1))\}$



$\xhookrightarrow{\text{enter}}$

Beispiel

Heap: $\{x \not\mapsto\}$

$[\#_{\mathsf{APP}}(x)]$

$(\lambda z.z, [])$

$((\texttt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1)), [\#_{\mathsf{UPD}}(x)])$

Beispiel

Heap: $\{x \not\mapsto\}$

$[\#_{\text{APP}}(x)]$

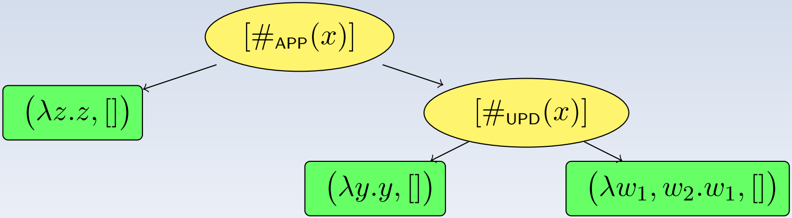
$(\lambda z.z, [])$

$((\texttt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1)), [\#_{\text{UPD}}(x)])$

$\xhookrightarrow{\text{fork}}$

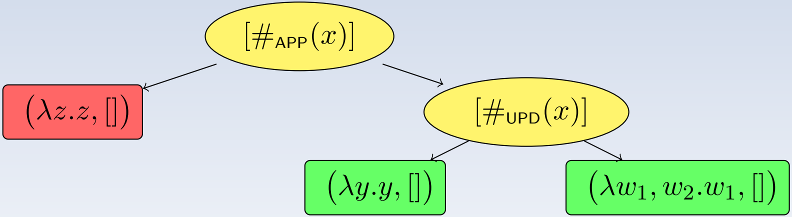
Beispiel

Heap: $\{x \not\mapsto\}$



Beispiel

Heap: $\{x \not\mapsto\}$



$\xhookrightarrow{\text{choose-l}}$

Beispiel

Heap: $\{x \not\mapsto\}$

$(\lambda z.z), [\#_{\text{APP}}(x)])$

$[\#_{\text{UPD}}(x)]$

$(\lambda y.y, [])$

$(\lambda w_1, w_2.w_1, [])$

Heap anpassen

Beispiel

Heap: $\{x \not\mapsto\}$

$(\lambda z.z), [\#_{\text{APP}}(x)])$

$((\texttt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1)), [\#_{\text{UPD}}(x)])$

Heap anpassen

Beispiel

Heap: $\{x \mapsto (\texttt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1))\}$

$(\lambda z.z), [\#_{\mathsf{APP}}(x)])$

$((\texttt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1)), [])$

Heap anpassen

Beispiel

$\boxed{\text{Heap: } \{x \mapsto (\texttt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1))\}}$

$\boxed{((\lambda z.z), [\#_{\text{APP}}(x)])}$

$\xhookrightarrow{\text{takeApp}}$

Beispiel

Heap: $\{x \mapsto (\mathtt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1))\}$

$(x, [])$

$\xhookrightarrow{\text{enter}}$

Beispiel

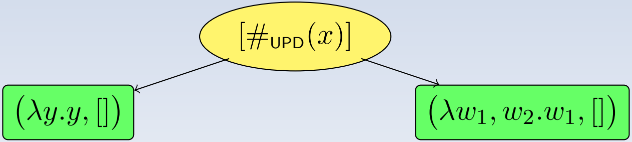
$$\boxed{\text{Heap: } \{x \mapsto (\mathtt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1))\}}$$

$$\boxed{((\mathtt{amb}\ (\lambda y.y)\ (\lambda w_1, w_2.w_1)), [\#_{\mathsf{UPD}}(x)])}$$

$$\xhookrightarrow{\text{fork}}$$

Beispiel

Heap: $\{x \not\mapsto\}$



Beispiel

Heap: $\{x \not\mapsto\}$

$$\begin{tikzcd} & {[\#_{\text{UPD}}(x)]} \arrow[dl] \arrow[dr] & \\ (\lambda y.y, []) & & (\lambda w_1, w_2.w_1, []) \end{tikzcd}$$

$$\xhookrightarrow{\text{choose-l}}$$

Beispiel

Heap: $\{x \not\mapsto\}$

$(\lambda y.y, [\#_{\text{UPD}}(x)])$

$(\lambda w_1, w_2.w_1, [])$

Beispiel

Heap: $\{x \not\mapsto\}$

$(\lambda y.y, [\#_{\text{UPD}}(x)])$

$(\lambda w_1, w_2.w_1, [])$

Heap anpassen

Beispiel

Heap: $\{x \not\mapsto\}$

$(\lambda y.y, [\#_{\mathsf{UPD}}(x)])$

$\xhookrightarrow{\text{update}}$

Beispiel

Heap: $\{x \mapsto (\lambda y.y)\}$

$(\lambda y.y, [])$

akzeptiere!

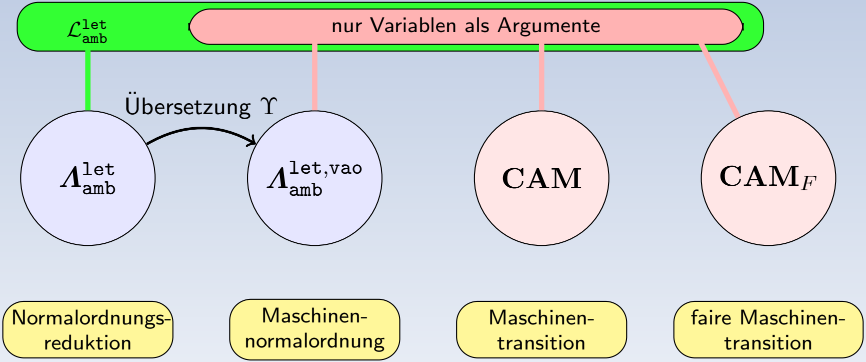
Korrektheit

Korrektheit im Rahmen von Übersetzungen

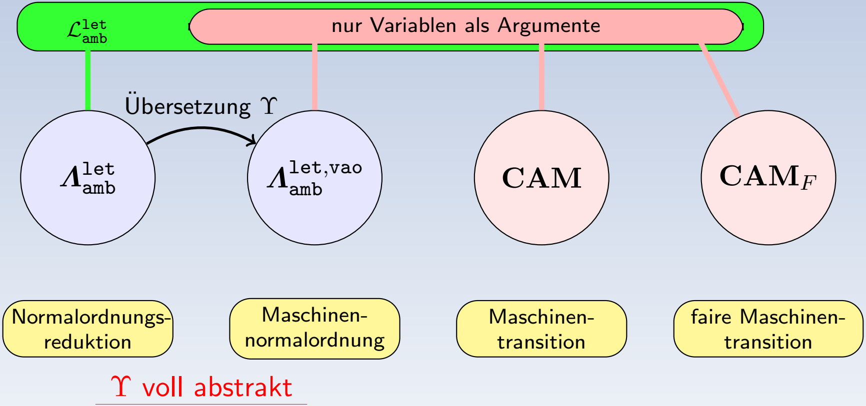
$\tau :: \mathsf{calc}_A \to \mathsf{calc}_B$ Übersetzung

- zumindest: Konvergenzäquivalenz:
  $$\tau(s){\downarrow_B} \iff s{\downarrow_A} \text{ und } \tau(s){\Downarrow_B} \iff s{\Downarrow_A}$$
- besser (bzgl. Gleichheiten): [Schmidt-Schauß et al., 2008]:
  - Adequatheit der Übersetzung
    $$\tau(s) \leq_B \tau(t) \implies s \leq_A t$$
  - Volle Abstraktheit impliziert äquivalente Gleichheitstheorien:
    $$\tau(s) \leq_B \tau(t) \Leftrightarrow s \leq_A t$$

Korrektheitsbeweis $\Lambda^{\mathtt{let}}_{\mathtt{amb}} \to \mathbf{CAM}_F$

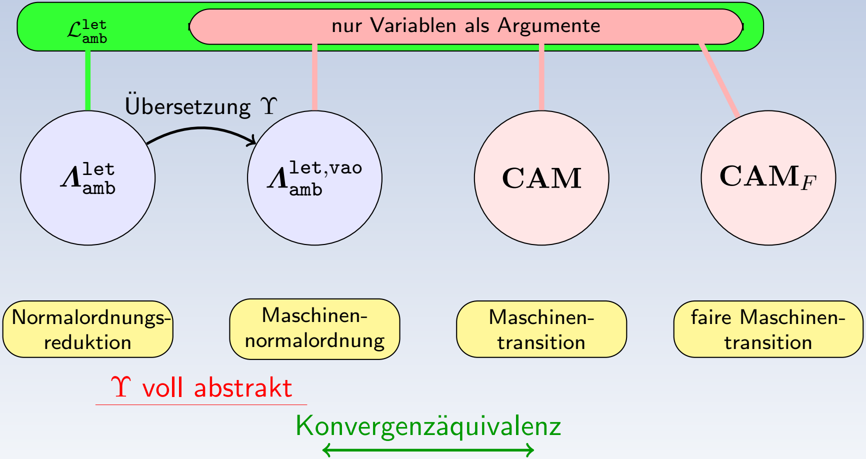


Korrektheitsbeweis $\Lambda^{\mathtt{let}}_{\mathtt{amb}} \to \mathbf{CAM}_F$



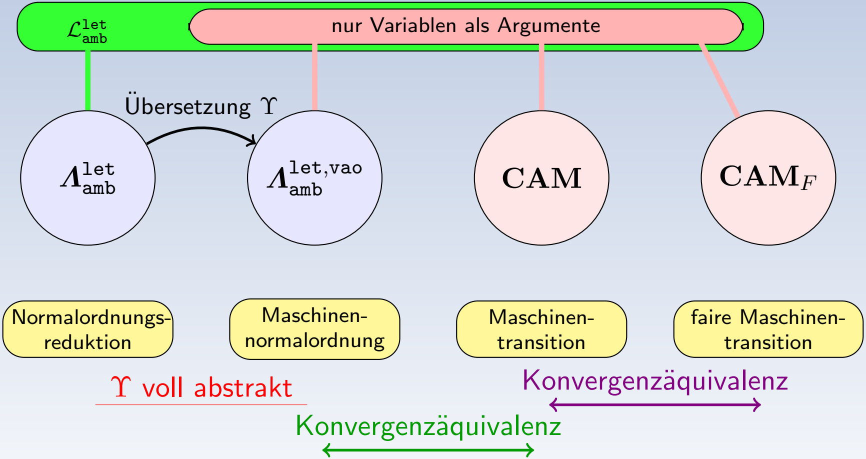
- unter Benutzung der Frameworks aus [Schmidt-Schauß et al., 2008]

Korrektheitsbeweis $\Lambda^{\mathtt{let}}_{\mathtt{amb}} \to \mathbf{CAM}_F$



- unter Benutzung der Frameworks aus [Schmidt-Schauß et al., 2008]
- schwieriger Teil: Wenn $s{\downarrow}_{\mathbf{mo}}$, dann $s{\downarrow}_{\mathbf{CAM}}$

Korrektheitsbeweis $\Lambda_{\mathtt{amb}}^{\mathtt{let}} \to \mathbf{CAM}_F$



- unter Benutzung der Frameworks aus [Schmidt-Schauß et al., 2008]
- schwieriger Teil: Wenn $s{\downarrow_{\mathbf{mo}}}$, dann $s{\downarrow_{\mathbf{CAM}}}$

Fazit

- Programmtransformationen einer nebenläufigen Programmiersprache
- Operationaler Ansatz erfolgreich
- kontextuelle Äquivalenz mit may- & must-Konvergenz ergibt erwartete Gleichungen
- Mit den entwickelten Methoden viele Programmtransformationen als korrekt nachweisbar
- Sowohl Transformationen als auch Übersetzungen
- Resultate von Moran verbessert und nach Änderungen zum erfolgreichen Abschluss geführt

Ausblick

- Operationaler Ansatz auf viele Kalküle anwendbar
  - nur small-step Reduktion, Wertbegriff notwendig
  - generisches Kontextlemma bereits entwickelt [Schmidt-Schauß & Sabel, 2007]
- Auf nebenläufigen Prozesskalkül mit Speicher und Futures bereits angewendet [Niehren, Sabel, Schmidt-Schauß, Schwinghammer, 2007]
- $\Lambda^{\texttt{let}}_{\texttt{amb}}$: Weitere Beweismethoden denkbar, z.B. `take`-Lemma
- Verbesserung der endlichen Simulation
- getypte Kalküle betrachten
- Diagramm-Methode automatisieren