GOETHE UNIVERSITÄT FRANKFURT AM MAIN

www.uni-frankfurt.de

Reasoning about Contextual Equivalence: From Untyped to Polymorphically Typed Calculi

David Sabel
joint work with
Manfred Schmidt-Schauß, Frederik Harwath

Institut für Informatik
Fachbereich Informatik und Mathematik
Goethe-Universität Frankfurt

ATPS'09,
Lübeck, 1. Oktober 2009

Einleitung

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Ziel

Nachweis von Programmgleichheiten unter polymorpher Typisierung
(Anwendung z.B. Korrektheit von Compileroptimierungen)

$$(\texttt{if}\ x\ \texttt{then}\ x\ \texttt{else}\ x) \overset{?}{\sim} x$$

Kontextuelle Äquivalenz

- basiert auf operationaler Semantik
- natürlicher Gleichheitsbegriff für Programme
- Gleichheitsnachweise i.a. aufwändig,
  es gibt mehrere Beweismethoden.

Parametrischer Polymorphismus

- Weit verbreitet in Typsystemen funktionaler Programmiersprachen
- Ausdrucksstark aber entscheidbar (Hindley-Milner)

Verwandte Arbeiten und Vorarbeiten

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Kontextuelle Äquivalenz in getypten Kalkülen

- Gordon, TCS, '99: simply typed PCF, Bisimulation
- Pitts, MSCS, 2000:
  Poly PCF, imprädikativer Polymorphismus, logical relations
- Voigtländer, Johann, TCS, 2007:
  PolySeq = Poly PCF + Seq, logical relations
- Johann, Voigtländer, I&C, 2009:
  PolySeq + Failure, logical relations

Eigene Vorarbeiten

- determ. Schmidt-Schauß, Sabel, Schütz, JFP,2008,
  nicht-determ. Sabel,Schmidt-Schauß,MSCS,2008
- Call-by-need, `letrec`, ungetypt
- Syntaktische Beweisverfahren zum Korrektheitsnachweis

Anforderungen / Ziele

1. (Korrekt-getypte) Programmgleichheiten des ungetypten Kalkül gelten auch im getypten
2. Anwendbarkeit von (getypten) Programmtransformation kann lokal entschieden werden.
3. Die (syntaktischen) Beweisverfahren für den ungetypten Kalkül können nach Übertragung/Anpassung im getypten Kalkül benutzt werden.

Haskell-ähnliche Kernsprache

Syntax, ungetypt L_{LC}

- Ausdrücke E

$$\begin{aligned} E \quad ::= & V \mid (E\ E) \mid \lambda V.E \mid (\texttt{seq}\ E\ E) \\ & \mid (\texttt{letrec}\ V_1 = E_1, \ldots, V_n = E_n\ \texttt{in}\ E) \\ & \mid (c_i\ E_1 \ldots E_{\mathrm{ar}(c_i)}) \mid (\texttt{case}_K\ E\ \texttt{of}\ Alt_1 \ldots Alt_{|D_K|}) \\ Alt_i ::= & ((c_i\ V_1 \ldots V_{\mathrm{ar}(c_i)})\ \texttt{->}\ E) \end{aligned}$$

- D_K = Menge von Datenkonstruktoren zum Typkonstruktor K
- Kontext $\mathbb{C}$ = Ausdruck mit Loch $[\cdot]$ an Ausdruckposition.
- $\mathbb{C}[s]$ = Einsetzung von s in das Loch von $\mathbb{C}$

Syntax von Typen

- nicht-quantifiziert: $T ::= X \mid (T \to T) \mid (K\ T_1 \ldots T_{\mathrm{ar}(K)})$
  wobei K Typkonstruktor
- quantifiziert: $\forall X_1, \ldots, X_n.T$ oder kurz $\forall \mathcal{X}.T$

Die polymorph getypte Sprache

L_{PLC}

- L_{PLC} = Menge von wohlgetypten L_{LC} Ausdrücken
- prädikativ parametrisch-polymorphe Typisierung:
  Polymorphe Typen nur für `letrec`-Variablen
  (andere Variablen monomorph)
- getypte Terme haben Typmarkierungen an allen Untertermen
- $\forall$-Quantoren nur an `letrec`-Bindungen
  `letrec` $x :: \forall \mathcal{X}.T = s :: T', \ldots$
- Berechnung der Typmarkierungen z.B. Typherleitungssystem

Weitere Notationen

- Getypte Kontexte $\mathbb{C}[:: T]$:
  Kontext mit Typmarkierung am Loch
- Type-Erasure $\varepsilon(t) \in L_{LC}$:
  Ausdruck t nach Löschung aller Typmarkierungen

Operationale Semantik

Normalordnungsreduktion (call-by-need) $\xrightarrow{\text{no}}$ auf ungetypten Termen

Anwenden von Rewriting-Regeln an Reduktionspositionen (markiert durch $^{\text{sub}}$, $^{\text{vis}}$)

(lbeta) $\mathbb{C}[((\lambda x.s)^{\text{sub}}\ r)] \longrightarrow \mathbb{C}[(\texttt{letrec}\ x = r\ \texttt{in}\ s)]$

(cp) $\texttt{letrec}\ x = v^{\text{sub}}, \ldots\ \mathbb{C}[x^{\text{vis}}] \longrightarrow \texttt{letrec}\ x = v^{\text{sub}}, \ldots\ \mathbb{C}[v^{\text{vis}}]$
wobei $v \in \{x, \lambda x.s, (c\ x_1 \ldots x_n)\}$

(case) $\mathbb{C}[(\texttt{case}\ c^{\text{sub}}\ \texttt{of}\ \ldots(c\ \texttt{->}\ s)\ldots)] \longrightarrow \mathbb{C}[s]$

(llet-e) $(\texttt{letrec}\ Env_1, x = (\texttt{letrec}\ Env_2\ \texttt{in}\ s)^{\text{sub}}\ \texttt{in}\ t) \longrightarrow (\texttt{letrec}\ Env_1, Env_2, x = s\ \texttt{in}\ t)$

... ...

Operationale Semantik für getypte Terme

- Reduziere die Type-Erasure $\varepsilon(t)$
- WHNF: $(\texttt{letrec}\ Env\ \texttt{in}\ v)$ oder v, wobei $v = \lambda x.s$ oder $v = (c\ s_1 \ldots s_n)$
- Für ungetyptes t: $t \downarrow_{no}$ gdw. $t \xrightarrow{no,*} t'$ und t' ist WHNF.
- Für $t \in L_{PLC}$: $t \downarrow_{no}$ gdw. $\varepsilon(t) \downarrow_{no}$

Kontextuelle Äquivalenz

Kontextuelle Approximation $\leq_T$ und Gleichheit $\sim_T$

Für Ausdrücke $s, t :: T$:

$$s \approx_{wt} t \quad \text{gdw.} \quad \forall \mathbb{C}[\cdot :: T] : \mathbb{C}[s] \in L_{PLC} \iff \mathbb{C}[t] \in L_{PLC}$$

$$s \leq_T t \quad \text{gdw.} \quad s \approx_{wt} t \wedge \forall \mathbb{C}[\cdot :: T], (\mathbb{C}[s] \in L_{PLC}) : (\varepsilon(\mathbb{C}[s])\downarrow_{no} \Rightarrow \varepsilon(\mathbb{C}[t])\downarrow_{no})$$

$$s \sim_T t \quad \text{gdw.} \quad s \leq_T t \quad \wedge \quad t \leq_T s$$

Auf ungetypten Termen

$$s \leq t \text{ gdw. } \forall \mathbb{C} : \mathbb{C}[s]\downarrow_{no} \implies \mathbb{C}[t]\downarrow_{no} \text{ und } \sim = \leq \cap \geq$$

Programmtransformationen

Getypte Programmtransformation P

- binäre Relation auf L_{PLC}
- $(s,t) \in P \implies s,t$ vom gleichen Typ
- P_T: Restriktion von P auf Typ T

Anwendbarkeit

Anwendbarkeit ist lokal möglich aufgrund der Typ-Markierungen

$$\mathbb{C}[s :: T] \to \mathbb{C}[t :: T]$$

Korrektheit

P ist korrekt gdw. für alle $(s,t) \in P_T$ gilt: $s \sim_T t$.

Übertragung von Gleichheiten aus dem Ungetypten

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Offensichtlich

Falls $\varepsilon(s) \sim \varepsilon(t)$, $s, t :: T$ und $\mathbb{C}[s] \approx_{wt} \mathbb{C}[t]$, dann $\mathbb{C}[s] \sim_T \mathbb{C}[t]$.

Damit lassen sich bekannte Gleichungen aus dem ungetypten Kalkül "importieren"

[Schmidt-Schauß,Sabel,Schütz 2008, Schmidt-Schauß 2007]

- Alle Reduktionsregeln sind korrekt.
- Weitere korrekte Programmtransformationen, z.B.
  - Garbage Collection (gc),

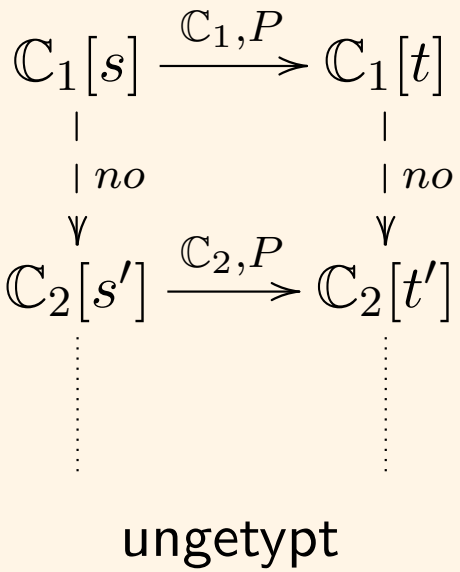
    $\texttt{letrec}\ x_1 = s_1, \ldots, x_n = s_n\ \texttt{in}\ t \ \rightarrow\ t$ falls $x_i \notin FV(t)$
  - Kopieren von Ausdrücken (gcp)

    $\texttt{letrec}\ x = s, Env\ \texttt{in}\ \mathbb{C}[x] \rightarrow \texttt{letrec}\ x = s, Env\ \texttt{in}\ \mathbb{C}[s]$

Wie getypte Gleichungen beweisen?

Die syntaktischen Verfahren (Diagrammtechnik, Kontextlemma) im ungetypten Kalkül führen Induktion auf Reduktionsfolgen durch,

Anforderung innerhalb dieser Verfahren

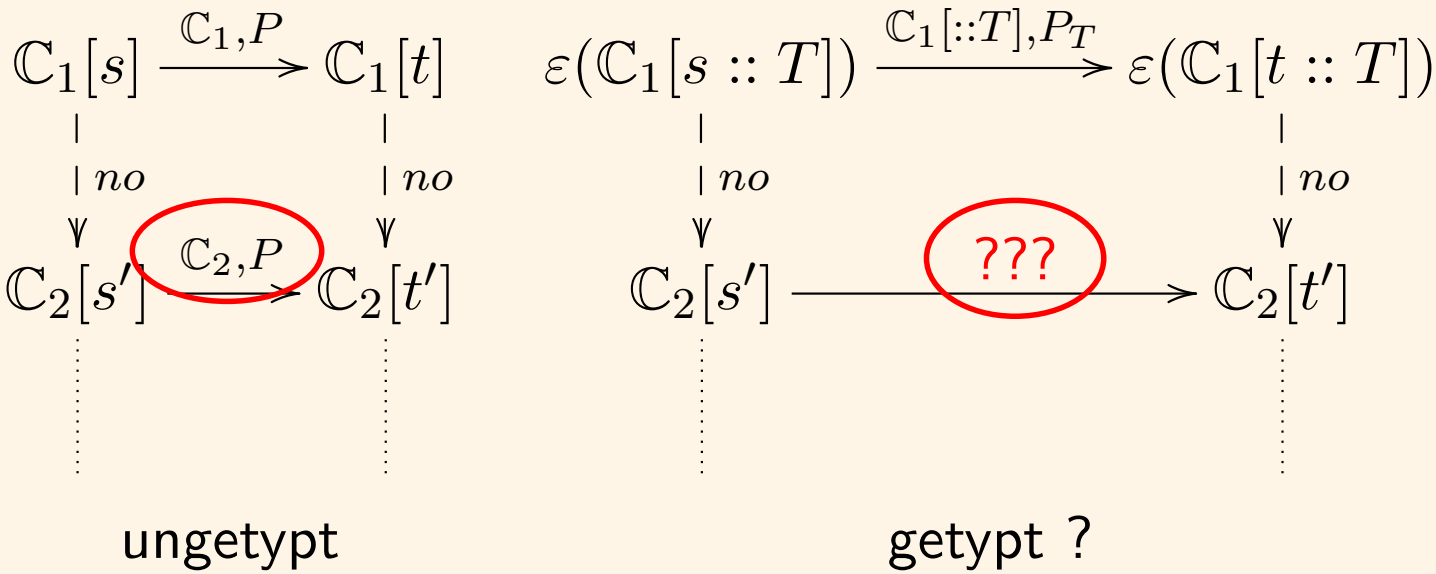


ungetypt

Wie getypte Gleichungen beweisen?

Die syntaktischen Verfahren (Diagrammtechnik, Kontextlemma) im ungetypten Kalkül führen Induktion auf Reduktionsfolgen durch,

Anforderung innerhalb dieser Verfahren



ungetypt

getypt ?

Korrektheitsbeweis getypter Gleichungen

Ansatz für getypte Gleichungen

$$\begin{array}{ccc} \mathbb{C}_1[s :: T] & \xrightarrow{\mathbb{C}_1[::T],P_T} & \mathbb{C}_1[t :: T] \\ \Big| & & \Big| \\ \Big| \; tno & & \Big| \; tno \\ \Downarrow & & \Downarrow \\ \mathbb{C}_2[s' :: T'] & \xrightarrow{\mathbb{C}_2[::T'],P_{T'}} & \mathbb{C}_2[t' :: T'] \\ \vdots & & \vdots \end{array}$$

Normalordnungsreduktion $\xrightarrow{tno}$ auf Typ-markierten Termen

- $\xrightarrow{tno}$ nimmt Typmarkierungen mit und passt diese an.
- Ziel: $\xrightarrow{tno}$ möglichst einfach
- Statt Typherleitung benutze Constraints an die Typmarkierungen
- Wohlgetypt = Constraints an die Typmarkierungen sind erfüllt

Typ-Constraints

$$((\lambda z.\left(\begin{array}{l}\texttt{letrec } id = (\lambda x.x),\\ \qquad\qquad y = (\lambda w.(id\ z))\\ \texttt{in } (y\ (id\ \texttt{True}^{\texttt{Bool}}))\end{array}\right))\ \texttt{Nil})$$

$bt(x) = b \quad bt(id) = \forall a.a \to a \quad bt(z) = [c] \quad bt(w) = d \quad bt(y) = \forall e.e \to [c]$

- Variablen haben einen eingebauten Typen $bt(x) = T$
- Variablen: Für jedes Vorkommen $x :: S$ und $S \preceq bt(x)$, Lambda- und Pattern-gebundene $x :: bt(x)$, wobei $bt(x)$ nicht quantifiziert.
- Für jeden Unterterm s: Typmarkierung ist $MonoTp(s)$
  (für Konstruktoren c und seq: Instanz von $typeOf(c)$ (bzw. $\forall a, b.a \to b \to b$))
- Für jede letrec-Bindung $x_i :: bt(x_i) = t_i :: T_i$ gilt $bt(x_i) \preceq \forall \mathcal{X}.T_i$
  wobei $MonoTp(t_i) = T_i$ und $\mathcal{X} = FTV(T_i) \setminus \left(\bigcup_{x \in FV(t_i)} FTV(bt(x))\right)$

Typ-Constraints

$$((\lambda z^{[c]}.\left(\begin{array}{l}\texttt{letrec } id^{\forall a.a\to a} = (\lambda x^{b}.x^{b}),\\ \qquad\qquad y^{\forall e.e\to[c]} = (\lambda w^{d}.(id^{[c]\to[c]}\ z^{[c]}))\\ \texttt{in } (y^{\texttt{Bool}\to[c]}\ (id^{\texttt{Bool}\to\texttt{Bool}}\ \texttt{True}^{\texttt{Bool}}))\end{array}\right)\ \texttt{Nil})$$

$bt(x) = b \quad bt(id) = \forall a.a \to a \quad \boldsymbol{bt(z) = [c]} \quad bt(w) = d \quad bt(y) = \forall e.e \to [c]$

- Variablen haben einen eingebauten Typen $bt(x) = T$
- Variablen: Für jedes Vorkommen $x :: S$ und $S \preceq bt(x)$, Lambda- und Pattern-gebundene $x :: bt(x)$, wobei $bt(x)$ nicht quantifiziert.
- Für jeden Unterterm s: Typmarkierung ist $MonoTp(s)$
  (für Konstruktoren c und seq: Instanz von $typeOf(c)$ (bzw. $\forall a, b.a \to b \to b$))
- Für jede letrec-Bindung $x_i :: bt(x_i) = t_i :: T_i$ gilt $bt(x_i) \preceq \forall \mathcal{X}.T_i$
  wobei $MonoTp(t_i) = T_i$ und $\mathcal{X} = FTV(T_i) \setminus \left(\bigcup_{x \in FV(t_i)} FTV(bt(x))\right)$

Typ-Constraints

$$((\lambda z^{[c]}.\left(\begin{array}{l}\texttt{letrec } \boldsymbol{id}^{\forall a.a\to a} = (\lambda x^b.x^b),\\ \qquad\qquad y^{\forall e.e\to[c]} = (\lambda w^d.(\boldsymbol{id}^{[c]\to[c]}\ z))\\ \texttt{in } (y^{\texttt{Bool}\to[c]}\ (\boldsymbol{id}^{\texttt{Bool}\to\texttt{Bool}}\ \texttt{True}^{\texttt{Bool}}))\end{array}\right)\ \texttt{Nil})$$

$bt(x) = b \quad \boldsymbol{bt(id) = \forall a.a \to a} \quad bt(z) = [c] \quad bt(w) = d \quad bt(y) = \forall e.e \to [c]$

- Variablen haben einen eingebauten Typen $bt(x) = T$
- Variablen: Für jedes Vorkommen $x :: S$ und $S \preceq bt(x)$, Lambda- und Pattern-gebundene $x :: bt(x)$, wobei $bt(x)$ nicht quantifiziert.
- Für jeden Unterterm s: Typmarkierung ist $MonoTp(s)$
  (für Konstruktoren c und seq: Instanz von $typeOf(c)$ (bzw. $\forall a, b.a \to b \to b$))
- Für jede letrec-Bindung $x_i :: bt(x_i) = t_i :: T_i$ gilt $bt(x_i) \preceq \forall \mathcal{X}.T_i$
  wobei $MonoTp(t_i) = T_i$ und $\mathcal{X} = FTV(T_i) \setminus \left(\bigcup_{x \in FV(t_i)} FTV(bt(x))\right)$

Typ-Constraints

$$((\lambda z^{[c]}.\left(\begin{array}{l}\texttt{letrec } id^{\forall a.a\to a} = (\lambda x^b.x^b),\\ \qquad\qquad y^{\forall e.e\to[c]} = (\lambda w^d.(id^{[c]\to[c]}\ z))\\ \texttt{in } (y^{\texttt{Bool}\to[c]}\ (id^{\texttt{Bool}\to\texttt{Bool}}\ \texttt{True}^{\texttt{Bool}}))\end{array}\right)\ \texttt{Nil}^{[c]})$$

$bt(x) = b \quad bt(id) = \forall a.a \to a \quad bt(z) = [c] \quad bt(w) = d \quad bt(y) = \forall e.e \to [c]$

- Variablen haben einen eingebauten Typen $bt(x) = T$
- Variablen: Für jedes Vorkommen $x :: S$ und $S \preceq bt(x)$, Lambda- und Pattern-gebundene $x :: bt(x)$, wobei $bt(x)$ nicht quantifiziert.
- Für jeden Unterterm s: Typmarkierung ist $MonoTp(s)$
  (für Konstruktoren c und seq: Instanz von $typeOf(c)$ (bzw. $\forall a, b.a \to b \to b$))
- Für jede `letrec`-Bindung $x_i :: bt(x_i) = t_i :: T_i$ gilt $bt(x_i) \preceq \forall \mathcal{X}.T_i$
  wobei $MonoTp(t_i) = T_i$ und $\mathcal{X} = FTV(T_i) \setminus \left(\bigcup_{x \in FV(t_i)} FTV(bt(x))\right)$

Typ-Constraints

$$((\lambda z^{[c]}.\left(\begin{array}{l}\texttt{letrec } id^{\forall a.a\to a} = (\lambda x^b.x^b)^{b\to b},\\ \qquad\quad y^{\forall e.e\to[c]} = (\lambda w^d.(id^{[c]\to[c]}\ z^{[c]})^{[c]})^{d\to[c]}\\ \texttt{in } (y^{\texttt{Bool}\to[c]}\ (id^{\texttt{Bool}\to\texttt{Bool}}\ \texttt{True}^{\texttt{Bool}})^{\texttt{Bool}})^{[c]}\end{array}\right)^{[c]})^{[c]\to[c]}\ \texttt{Nil}^{[c]})^{[c]}$$

$bt(x) = b \quad bt(id) = \forall a.a \to a \quad bt(z) = [c] \quad bt(w) = d \quad bt(y) = \forall e.e \to [c]$

- Variablen haben einen eingebauten Typen $bt(x) = T$
- Variablen: Für jedes Vorkommen $x :: S$ und $S \preceq bt(x)$, Lambda- und Pattern-gebundene $x :: bt(x)$, wobei $bt(x)$ nicht quantifiziert.
- Für jeden Unterterm s: Typmarkierung ist $MonoTp(s)$
  (für Konstruktoren c und seq: Instanz von $typeOf(c)$ (bzw. $\forall a, b.a \to b \to b$))
- Für jede letrec-Bindung $x_i :: bt(x_i) = t_i :: T_i$ gilt $bt(x_i) \preceq \forall \mathcal{X}.T_i$
  wobei $MonoTp(t_i) = T_i$ und $\mathcal{X} = FTV(T_i) \setminus \left(\bigcup_{x \in FV(t_i)} FTV(bt(x))\right)$

Typ-Constraints

$$((\lambda z^{[c]}.\left(\begin{array}{l}\texttt{letrec}\ id^{\forall a.a\to a} = (\lambda x^{b}.x^{b})^{b\to b},\\ \qquad\quad y^{\forall e.e\to[c]} = (\lambda w^{d}.(id^{[c]\to[c]}\ z^{[c]})^{[c]})^{d\to[c]}\\ \texttt{in}\ (y^{\texttt{Bool}\to[c]}\ (id^{\texttt{Bool}\to\texttt{Bool}}\ \texttt{True}^{\texttt{Bool}})^{\texttt{Bool}})^{[c]}\end{array}\right)^{[c]})^{[c]\to[c]}\ \texttt{Nil}^{[c]})^{[c}$$

$bt(x) = b \quad bt(id) = \forall a.a \to a \quad bt(z) = [c] \quad bt(w) = d \quad bt(y) = \forall e.e \to [c]$

- Variablen haben einen eingebauten Typen $bt(x) = T$
- Variablen: Für jedes Vorkommen $x :: S$ und $S \preceq bt(x)$, Lambda- und Pattern-gebundene $x :: bt(x)$, wobei $bt(x)$ nicht quantifiziert.
- Für jeden Unterterm s: Typmarkierung ist $MonoTp(s)$
  (für Konstruktoren c und seq: Instanz von $typeOf(c)$ (bzw. $\forall a, b.a \to b \to b$))
- Für jede `letrec`-Bindung $x_i :: bt(x_i) = t_i :: T_i$ gilt $bt(x_i) \preceq \forall \mathcal{X}.T_i$
  wobei $MonoTp(t_i) = T_i$ und $\mathcal{X} = FTV(T_i) \setminus \left(\bigcup_{x \in FV(t_i)} FTV(bt(x))\right)$

Getypte Normalordnungsreduktion

- Reduktionsregeln wie vorher
- Typanpassung in fast allen Fällen offensichtlich.
- Ausnahme (cp):

$$\texttt{letrec}\ x = v :: T, \ldots \mathbb{C}[x :: S] \ldots \\ \rightarrow \texttt{letrec}\ x = v :: T, \ldots \mathbb{C}[\rho(v) :: S]$$

wobei ρ den Typ von v richtig instanziiert

Beispiel

$$\texttt{letrec}\ id^{\forall a.a \rightarrow a} = (\lambda x^b.x^b)^{b \rightarrow b} \\ \texttt{in}\ (id^{\texttt{Bool} \rightarrow \texttt{Bool}}\ \texttt{True})$$

$$\rightarrow \quad \texttt{letrec}\ id^{\forall a.a \rightarrow a} = \lambda x^b.x^b \\ \texttt{in}\ (\lambda x_1^{\texttt{Bool}}.x_1^{\texttt{Bool}})^{\texttt{Bool} \rightarrow \texttt{Bool}}\ \texttt{True})$$

Eigenschaften von $\xrightarrow{tno}$

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Korrespondenz von $\xrightarrow{no}$ und $\xrightarrow{tno}$

Für $s :: S$ gilt:

- $s \xrightarrow{tno} t$ impliziert $t :: S$ und $\varepsilon(s) \xrightarrow{no} \varepsilon(t)$.
- $\varepsilon(s) \xrightarrow{no} t$ impliziert $\exists t' :: S\colon\ s \xrightarrow{tno} t'$ und $\varepsilon(t') = t$

Theorem

Für $s, t :: T\colon\ s \leq_T t$ gdw.
$s \approx_{wt} t$ und $\forall \mathbb{C}[\cdot :: T] :, \mathbb{C}[s] \in L_{PLC} : ((\mathbb{C}[s])\downarrow_{tno} \Rightarrow (\mathbb{C}[t])\downarrow_{tno})$

$\Longrightarrow$ Korrektheit von Transformationen kann mittels $\xrightarrow{tno}$ nachgewiesen werden

Beweistechniken

Definitionen

Programmtransformation P ist

- FV-geschlossen gdw. für alle $(s,t) \in P : FV(s) = FV(t)$
- ρ-geschlossen gdw. P FV-geschlossen und
  für alle $(s,t) \in P : (\rho(s), \rho(t)) \in P$

Theorem

Transformation P ist FV-geschlossen $\implies P \subseteq \approx_{wt}$.

Kontextlemma für L_{PLC}

Für ρ-geschlossenes P:
Wenn $\forall (s,t) \in P$ und alle Oberflächenkontexte $\mathbb{S}$:

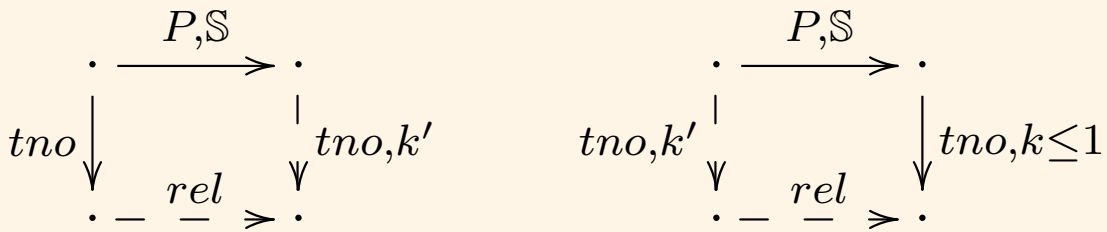
$$\mathbb{S}[s] \in L_{PLC} \implies (\mathbb{S}[s] \downarrow_{tno} \implies \mathbb{S}[t] \downarrow_{tno}).$$

Dann gilt für alle T: $P_T \subseteq \leq_T$

Beweistechnik: Diagramme

Gabel- & Vertauschungsdiagramme

Vollst. Darstellung der Reduktions- und Transformations-Überlappungen und Zusammenführbarkeit

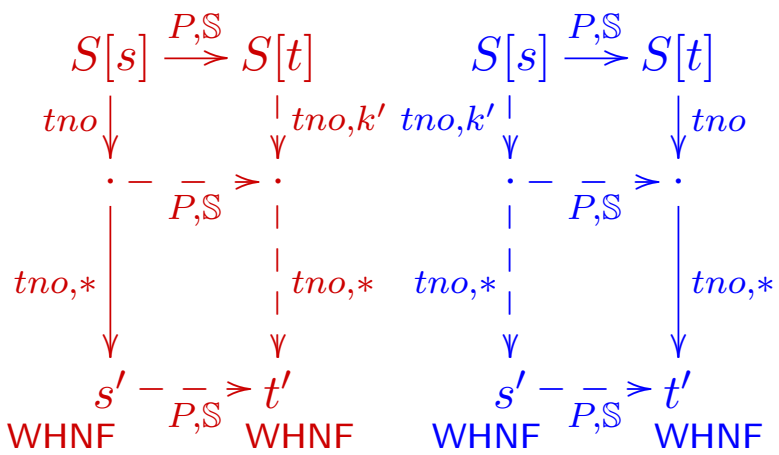


mit $k + k' > 0$, rel Relation auf L_{PLC}-Ausdrücken

ermöglichen: Induktive Konstruktion von Reduktionsfolgen

- $S[s] \downarrow_{tno} \Longrightarrow S[t] \downarrow_{tno}$
- $S[t] \downarrow_{tno} \Longrightarrow S[s] \downarrow_{tno}$

$\xrightarrow[\text{für } L_{PLC}]{\text{Kontextlemma}} P_T \subseteq \sim_T$



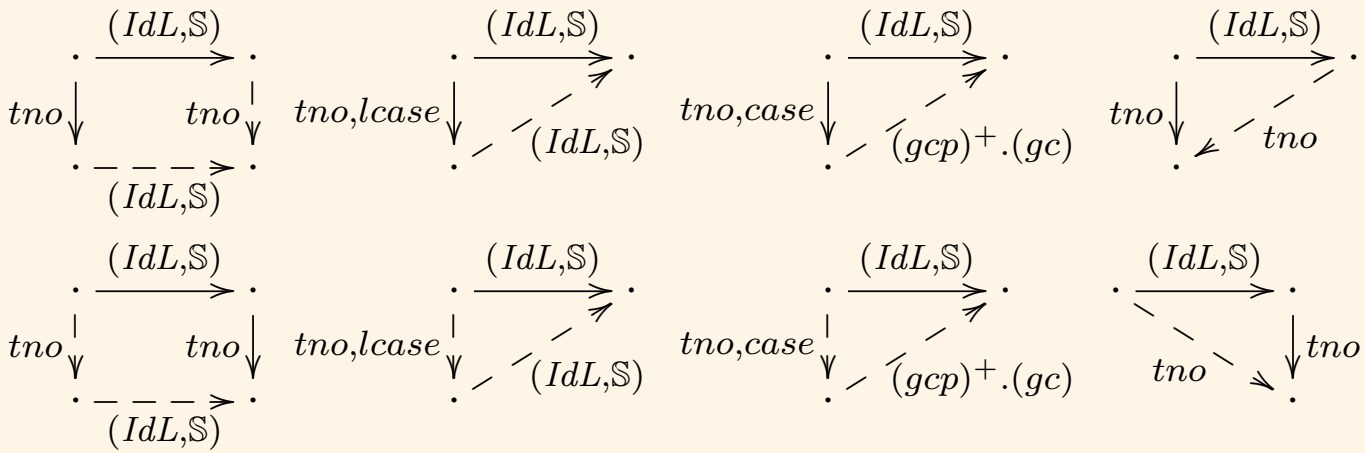
Beispiel: Typabhängige Programmtransformation

(IdL)-Transformation

$(\texttt{case}_{\texttt{List}}\ s\ \texttt{of}\ (\texttt{Nil}\ \texttt{->}\ \texttt{Nil})\ ((\texttt{Cons}\ x\ xs)\ \texttt{->}\ (\texttt{Cons}\ x\ xs)))$

$\longrightarrow s :: (\texttt{List}\ T)$

Diagramme



Proposition

Wenn $t :: (\texttt{List}\ T) \xrightarrow{\text{IdL}} t' :: (\texttt{List}\ T)$, dann $t \sim_{(\texttt{List}\ T)} t'$.

Fazit und Ausblick

Fazit

- Kontextuelle Äquivalenz für parametrischen Polymorphismus
- Syntaktische Beweismethoden möglich
- Korrektheit von getypten Programmtransformationen
- Wesentliche Technik: Typmarkierungen und Typvererbung

Ausblick

- Weitere Gleichheiten / Programmtransformationen
- Erweiterung auf nicht-deterministische Kalküle mit may- und must-Konvergenz
- Beziehung zur Hindley/Milner Typisierung