Proof Methods for Polymorphically Typed Contextual Equivalence

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

Institut für Informatik
Fachbereich Informatik und Mathematik
Goethe-Universität Frankfurt

Logik in der Informatik '09,
Frankfurt, 6.-7. November 2009

Introduction

Goal

Proof of program equalities with polymorphic typing
(Application e.g. correctness of optimizations in compilers)

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

Contextual Equivalence

- based on operational semantics
- natural notion of equality of programs
- proofs of correctness are difficult
  different proof methods exist

Parametric Polymorphism

- Widely used type system in functional programming languages
- Expressive but decidable (Hindley-Milner)

Related Work and Own Work

Contextual Equivalence in Typed Calculi

- Gordon, TCS, '99: simply typed PCF, Bisimulation
- Pitts, MSCS, 2000:
  Poly PCF, System F polymorphism, logical relations
- Voigtländer, Johann, TCS, 2007:
  PolySeq = Poly PCF + seq, logical relations

Own Work

- determ. Schmidt-Schauß, Sabel, Schütz, JFP,2008,
  non-determ. Sabel,Schmidt-Schauß,MSCS,2008
- Call-by-need, `letrec`, untyped
- Syntactic proof methods for correctness of program transformations

Requirements and Goals

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

1. applicability of (typed) program transformations can be decided locally

   $s :: \tau \sim_\tau t :: \tau$

   $\overset{?}{\Longrightarrow}$ for well-typed $C[s]$: $C[s] \to C[t]$ is correct

2. (correctly typed) program equalities of the untyped calculus are valid in the typed calculus

   $$s \sim t \implies s :: \tau \sim_\tau t :: \tau$$

3. the (syntactical) proof methods of the untyped calculus can be adjusted to the typed calculus

Haskell-like Core Language

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Syntax, untyped L_{LC}

- Expressions 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 = set of data constructors of type constructor K
- Context $\mathbb{C}$ = expression with a hole $[\cdot]$ at expression position.
- $\mathbb{C}[s]$ = plugging-in s into the hole of $\mathbb{C}$

Syntax of Types

- non-quantified: $T ::= X \mid (T \to T) \mid (K\ T_1 \ldots T_{\mathrm{ar}(K)})$
  where K is a type constructor
- quantified: $\forall X_1, \ldots, X_n.T$ or for short $\forall \mathcal{X}.T$

The Polymorphically Typed Language

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

L_{PLC}

- L_{PLC} = set of well-typed expressions
- parametric polymorphic typing:
  polymorphic types only for `letrec`-variables
  (other variables are typed monomorphically)
- typed expressions have type labels on all subexpressions
- $\forall$-quantifiers on `letrec`-bindings, only
  `letrec` $x :: \forall\mathcal{X}.T = s :: T', \ldots$
- Type labels may be computed by a derivation system

Further Notions

- Typed contexts $\mathbb{C}[:: T]$:
  Context with type label at the hole
- Type-Erasure $\varepsilon(t) \in L_{LC}$:
  Expression t after erasing all type labels

Operational Semantics

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Normal order reduction (call-by-need) $\xrightarrow{no}$ on untyped terms

Applying rewriting rules on reduction positions(labeled by $^{\text{sub}}$, $^{\text{vis}}$)

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

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

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

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

... ...

Operational Semantics of Typed Expressions

- Reduce the type erasure $\varepsilon(t)$
- WHNF: $(\texttt{letrec}\ Env\ \texttt{in}\ v)$ or v, where $v = \lambda x.s$ or $v = (c\ s_1 \ldots s_n)$
- For untyped t: $t\downarrow_{no}$ iff $t \xrightarrow{no,*} t'$ and t' is a WHNF.
- For $t \in L_{PLC}$: $t\downarrow_{no}$ iff $\varepsilon(t)\downarrow_{no}$

Contextual Equivalence

Contextual Approximation $\leq_T$ und Equivalence $\sim_T$

For expressions $s, t :: T$:

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

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

$$s \sim_T t \quad \text{iff} \quad s \leq_T t \quad \wedge \quad t \leq_T s$$

On untyped terms

$s \leq t$ iff $\forall \mathbb{C} : \mathbb{C}[s]\downarrow_{no} \implies \mathbb{C}[t]\downarrow_{no}$ and $\sim = \leq \cap \geq$

Program Transformations

Typed Program Transformation P

- binary relation on L_{PLC}
- $(s,t) \in P \implies s,t$ of the same type
- P_T: Restriction of P to type T

Correctness

P is correct iff for all $(s,t) \in P_T$ holds: $s \sim_T t$.

Applicability

Applicability is locally decidable by the type labels

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

Lifting Equivalences from the Untyped Calculus

Obviously

If $\varepsilon(s) \sim \varepsilon(t)$, $s, t :: T$ and $\mathbb{C}[s] \approx_{wt} \mathbb{C}[t]$, then $\mathbb{C}[s] \sim_T \mathbb{C}[t]$.

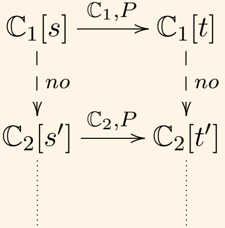
Thus known equivalences of the untyped calculus can be lifted [Schmidt-Schauß,Sabel,Schütz 2008, Schmidt-Schauß 2007]

- all reduction rules are correct
- further correct program transformations, e.g.
  - Garbage Collection (gc),
    $\texttt{letrec}\ x_1 = s_1, \ldots, x_n = s_n\ \texttt{in} t \rightarrow t$ if $x_i \notin FV(t)$
  - Copying of expressions (gcp)
    $\texttt{letrec}\ x = s, Env\ \texttt{in}\ \mathbb{C}[x] \rightarrow \texttt{letrec}\ x = s, Env\ \texttt{in}\ \mathbb{C}[s]$

How to Prove Correctness of Typed Equations?

Our syntactic methods for untyped calculi use
induction on reduction sequences

these methods require:

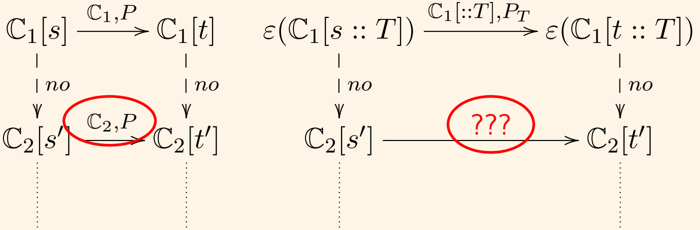


untyped

How to Prove Correctness of Typed Equations?

Our syntactic methods for untyped calculi use
induction on reduction sequences

these methods require:

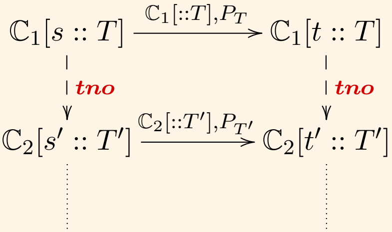


untyped typed ?

Correctness Proof of Typed Equivalences

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Approach



using constraints on the type labels

- instead of type derivation use constraints on the type labels
- well-typed = constraints on the type labels hold
- formalism uses built-in types for variables
- is sound wrt. standard iterative type derivation

Typed Normal Order Reduction

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

- $\xrightarrow{tno}$ preserves and adjusts type labels
- reduction rules as before
- type adjustment in most cases obvious
- exception (cp): $\texttt{letrec}\ x = v :: T, \ldots \mathbb{C}[x :: S] \ldots$
  $\rightarrow \texttt{letrec}\ x = v :: T, \ldots \mathbb{C}[\rho(v) :: S]$
  where ρ instantiates the type of v

For $s :: S$:

- $s \xrightarrow{tno} t$ implies $t :: S$ and $\varepsilon(s) \xrightarrow{no} \varepsilon(t)$.
- $\varepsilon(s) \xrightarrow{no} t$ implies $\exists t' :: S$: $s \xrightarrow{tno} t'$ and $\varepsilon(t') = t$

Theorem

For $s, t :: T$: $s \leq_T t$ iff
$s \approx_{wt} t$ and $\forall \mathbb{C}[\cdot :: T] :, \mathbb{C}[s] \in L_{PLC} : ((\mathbb{C}[s])\downarrow_{tno} \Rightarrow (\mathbb{C}[t])\downarrow_{tno})$

Proof Methods

Definitions

A program transformation P is

- FV-closed iff for all $(s,t) \in P : FV(s) = FV(t)$
- ρ-closed iff P is FV-closed and
  for all $(s,t) \in P : (\rho(s), \rho(t)) \in P$

Theorem

If a transformation P is FV-closed, then $P \subseteq \approx_{wt}$.

Context Lemma

For ρ-closed P:
If $\forall (s,t) \in P$ und all surface contexts $\mathbb{S}$:

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

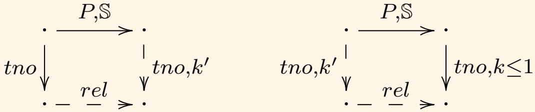
Then for all T holds: $P_T \subseteq \leq_T$

Diagrams

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Forking & Commuting Diagrams

Complete representation of the overlappings and joinability between reduction and transformation steps

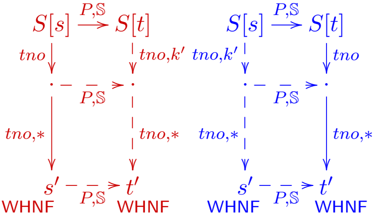


where $k + k' > 0$, rel relation on L_{PLC}-expressions

allow: inductive construction of reduction sequences

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

$\xrightarrow[\text{Lemma}]{\text{Context}} P_T \subseteq \sim_T$



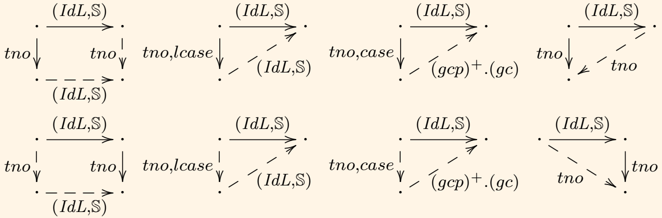
Example: A Type Dependent Program Transformation

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Transformation (IdL)

$$(\texttt{case}_{\texttt{List}}\ s\ \texttt{of}\ (\texttt{Nil} \texttt{->} \texttt{Nil})\ ((\texttt{Cons}\ x\ xs) \texttt{->} (\texttt{Cons}\ x\ xs))) \longrightarrow s :: [T]$$

Diagrams



Proposition

If $t :: [T] \xrightarrow{\mathsf{IdL}} t' :: [T]$, then $t \sim_{[T]} t'$.

Conclusion and Further Work

Conclusion

- Contextual equivalence for parametric polymorphism
- Syntactic proof methods applicable
- Correctness of typed program transformations
- Main technique: type labeling and type inheritance

Further Work

- Further equivalences / program transformations
- Extension to non-determinstic calculi with may- and must convergence
- Relation to Hindley/Milner typing