A Mechanized Proof of Type Safety for the Polymorphic λ-Calculus with References

Michalis A. Papakyriakou   Prodromos E. Gerakios
Nikolaos S. Papaspyrou

National Technical University of Athens
School of Electrical and Computer Engineering
Software Engineering Laboratory

{mpapakyr, pgerakios, nickie}@softlab.ntua.gr

6th Panhellenic Logic Symposium
Volos, Greece, 5-8 July 2007

Outline

Introduction
- Type systems and type safety
- Polymorphic λ-calculus
- References
- Mechanized proof

The language $\lambda^{\forall,\,\mathrm{ref}}$

Encoding $\lambda^{\forall,\,\mathrm{ref}}$ in Isabelle/HOL

A tour of the proof

Conclusions and future work

What is this paper about?

- The language
  Polymorphic λ-calculus with references
- The goal
  A proof of type safety
- The method
  Mechanized proof
  Using Isabelle/HOL

Type systems

- A type system defines:
  - how a programming language classifies values and expressions into types
  - how elements of these types can be manipulated
  - how these types can interact
- A type indicates a set of values that have the same generic meaning or intended purpose
- The purpose of type systems: to prevent certain forms of erroneous or undesirable program behaviour

Type safety

- If a program is free of static type errors, then its execution is free of dynamic type errors
- Kinds of dynamic errors that can be avoided:
  - programs can only access appropriate memory locations (memory safety)
  - programs can only transfer control to appropriate program points (control safety)

Type safety

- If a program is free of static type errors, then its execution is free of dynamic type errors
- Kinds of dynamic errors that can be avoided:
  - programs can only access appropriate memory locations (memory safety)
  - programs can only transfer control to appropriate program points (control safety)
- The standard procedure
  - Syntax
  - Operational semantics
  - Typing rules
  - Safety = preservation + progress

Polymorphic λ-calculus

- System F, F_2
- Girard, 1971; Reynolds, 1974
- First-class polymorphism
  - Useful for code reuse and modular type checking

Polymorphic λ-calculus

- System F, F_2
- Girard, 1971; Reynolds, 1974
- First-class polymorphism
  - Useful for code reuse and modular type checking
  - Polymorphic types and functions

$$\begin{aligned} \mathrm{id} &: \forall\alpha.\,\alpha \to \alpha \\ &= \Lambda\alpha.\,\lambda x{:}\alpha.\,x \end{aligned}$$

Polymorphic λ-calculus

- System F, F_2
- Girard, 1971; Reynolds, 1974
- First-class polymorphism
  - Useful for code reuse and modular type checking
  - Polymorphic types and functions

$$\begin{aligned} \mathsf{id} &: \forall\alpha.\,\alpha \to \alpha \\ &= \Lambda\alpha.\,\lambda x\!:\!\alpha.\,x \end{aligned}$$

  - Explicit type application

$$\mathsf{append} : \forall\alpha.\,\mathsf{list}\,\alpha \to \mathsf{list}\,\alpha \to \mathsf{list}\,\alpha$$

$$\ldots \mathsf{append}\,[\mathsf{int}]\,[1,2,3]\,[4,5] \ldots$$

ML-style references

Imperative programming in functional style

- Reference allocation
  let $r = \texttt{new}\ 7 \ldots$ $\qquad r : \mathsf{Ref\ int}$

ML-style references

Imperative programming in functional style

- Reference allocation
  let $r = \texttt{new}\ 7 \ldots$ $\qquad r : \text{Ref int}$
- Assignment
  $\ldots \text{in}\ r := 42; \ldots$ $\qquad$ destructive update!

ML-style references

Imperative programming in functional style

- Reference allocation

  let $r = \texttt{new}\ 7 \ldots$ $\qquad$ r : Ref int
- Assignment

  $\ldots$ in $r := 42; \ldots$ $\qquad$ destructive update!
- Dereference

  $\ldots$ print $(\texttt{deref}\ r)$; $\qquad$ prints 42

ML-style references

Imperative programming in functional style

- Reference allocation

  let $r = \texttt{new}\ 7 \ldots$ $\quad$ r : Ref int
- Assignment

  $\ldots$ in $r := 42; \ldots$ $\quad$ destructive update!
- Dereference

  $\ldots$ print $(\texttt{deref}\ r);$ $\quad$ prints 42
- No reference deallocation!

  $\ldots$ free r $\quad$ use garbage collection!

Polymorphic references

- The problem

$\text{let } r = \Lambda\alpha.\, \texttt{new}\ (\lambda x{:}\alpha.\, x) \text{ in}$ $\qquad r : \forall\alpha.\, \text{Ref}\ (\alpha \to \alpha)$

$\quad r\ [\text{int}] := \text{succ};$

$\quad \texttt{deref}\ (r\ [\text{bool}])\ \text{true}$ $\qquad$ dynamic type error

Polymorphic references

- The problem

  let $r = \Lambda\alpha.\, \texttt{new}\ (\lambda x{:}\alpha.\, x)$ in $\qquad r : \forall\alpha.\, \text{Ref}\ (\alpha \to \alpha)$

  $r\ [\text{int}] := \text{succ};$

  $\texttt{deref}\ (r\ [\text{bool}])\ \text{true}$ $\qquad$ dynamic type error

- A solution: value restriction

  In $\Lambda\alpha.\, v$, the term v must be a value

Mechanized proof *(i)*

- Why not with pencil and paper?
  - easy to make a mistake
  - easy to "fix" a mistake
  - if one is willing to spend time and effort to write a thorough proof with pencil and paper, why not use a proof assistant?

Mechanized proof *(i)*

- Why not with pencil and paper?
  - easy to make a mistake
  - easy to "fix" a mistake
  - if one is willing to spend time and effort to write a thorough proof with pencil and paper, why not use a proof assistant?
- Proof assistants
  - tools to develop formal proofs by man-machine collaboration
  - interactive proof editor, with which a human can guide the search for proofs
  - some steps of the proofs can be provided by the computer
  - not (necessarily) automatic theorem proving!

Mechanized proof *(ii)*

- Some available proof assistants:
  Isabelle/HOL, Coq, Twelf, NuPRL, PVS, PhoX, MINLOG, . . .
- Isabelle/HOL
  - Larry Paulson, Cambridge University
  - Tobias Nipkow, TU München
  - `http://isabelle.in.tum.de/`

Syntax of $\lambda^{\forall,\,ref}$

$$
\begin{array}{rcl}
\tau & ::= & \mathsf{Unit} \mid \alpha \mid \tau \to \tau \mid \forall\alpha.\,\tau \mid \mathsf{Ref}\ \tau \\
e & ::= & \mathtt{unit} \mid x \mid \lambda x{:}\tau.\,e \mid \Lambda\alpha.\,e \mid e_1\,e_2 \mid e\,[\tau] \\
 & \mid & \mathtt{new}\ e \mid \mathtt{deref}\ e \mid e_1 := e_2 \mid \mathtt{loc}\ l \\
v & ::= & \mathtt{unit} \mid \lambda x{:}\tau.\,e \mid \Lambda\alpha.\,v \mid \mathtt{loc}\ l
\end{array}
$$

Typing rules of $\lambda^{\forall,\,ref}$

$$\Gamma; \Delta; M \vdash \texttt{unit} : \mathsf{Unit} \qquad \Gamma, x:\tau; \Delta; M \vdash x : \tau$$

$$\frac{\Delta \models \tau \qquad \Gamma, x:\tau; \Delta; M \vdash e : \tau'}{\Gamma; \Delta; M \vdash \lambda x{:}\tau.\, e : \tau \to \tau'} \qquad \frac{\Gamma; \Delta, \alpha; M \vdash v : \tau}{\Gamma; \Delta; M \vdash \Lambda\alpha.\, v : \forall\alpha.\, \tau}$$

$$\frac{\Gamma; \Delta; M \vdash e_1 : \tau \to \tau' \qquad \Gamma; \Delta; M \vdash e_2 : \tau}{\Gamma; \Delta; M \vdash e_1\, e_2 : \tau'}$$

$$\frac{\Gamma; \Delta; M \vdash e : \forall\alpha.\, \tau \qquad \Delta \models \tau'}{\Gamma; \Delta; M \vdash e\,[\tau'] : \tau\{\alpha \mapsto \tau'\}} \qquad \frac{\Gamma; \Delta; M \vdash e : \tau}{\Gamma; \Delta; M \vdash \texttt{new}\, e : \mathsf{Ref}\ \tau}$$

$$\frac{\Gamma; \Delta; M \vdash e : \mathsf{Ref}\ \tau}{\Gamma; \Delta; M \vdash \texttt{deref}\, e : \tau}$$

$$\frac{\Gamma; \Delta; M \vdash e_1 : \mathsf{Ref}\ \tau \qquad \Gamma; \Delta; M \vdash e_2 : \tau}{\Gamma; \Delta; M \vdash e_1 := e_2 : \mathsf{Unit}}$$

$$\Gamma; \Delta; M, l:\tau \vdash \texttt{loc}\, l : \mathsf{Ref}\, \tau$$

Operational semantics of $\lambda^{\forall,\mathit{ref}}$

$$\frac{S; e_1 \longrightarrow S'; e_1'}{S; e_1\, e_2 \longrightarrow S'; e_1'\, e_2} \qquad \frac{S; e_2 \longrightarrow S'; e_2'}{S; v_1\, e_2 \longrightarrow S'; v_1\, e_2'}$$

$$\frac{S; e \longrightarrow S'; e'}{S; e\,[\tau] \longrightarrow S'; e'\,[\tau]}$$

$$\frac{S; e \longrightarrow S'; e'}{S; \texttt{new}\; e \longrightarrow S'; \texttt{new}\; e'} \qquad \frac{S; e \longrightarrow S'; e'}{S; \texttt{deref}\; e \longrightarrow S'; \texttt{deref}\; e'}$$

$$\frac{S; e_1 \longrightarrow S'; e_1'}{S; e_1 := e_2 \longrightarrow S'; e_1' := e_2} \qquad \frac{S; e_2 \longrightarrow S'; e_2'}{S; v_1 := e_2 \longrightarrow S'; v_1 := e_2'}$$

$$S; (\lambda x\!:\!\tau.\, e)\, v \longrightarrow S; e\{x \mapsto v\} \qquad S; (\Lambda\alpha.\, v)\,[\tau] \longrightarrow S; v\{\alpha \mapsto \tau\}$$

$$S; \texttt{new}\; v \longrightarrow S, l \mapsto v; \texttt{loc}\; l$$

$$S, l \mapsto v; \texttt{deref}\; (\texttt{loc}\; l) \longrightarrow S, l \mapsto v; v$$

$$S, l \mapsto v'; \texttt{loc}\; l := v \longrightarrow S, l \mapsto v; v$$

Encoding $\lambda^{\forall,\,ref}$ *in Isabelle/HOL*

- Main problems
  - The representation of bound variables
  - The representation of type environments
- The details are usually ignored in pencil and paper proofs

Encoding $\lambda^{\forall,\, ref}$ *in Isabelle/HOL*

- Main problems
  - The representation of bound variables
  - The representation of type environments
- The details are usually ignored in pencil and paper proofs

- We represent type environments as finite sets

Bound variables *(i)*

Proposed solutions

- Named variables
- DeBruijin indices
- Locally nameless
- Nominal
- Higher-order abstract syntax

Bound variables (ii)

Named variables

- Typical in the pencil and paper study of λ-calculus
- ... assuming fresh variable names!
- Main problem: α-equivalence
- Capture avoiding substitution may have to rename variables

Bound variables (iii)

DeBruijn indices

- Variables are index numbers, counting enclosing λ-abstractions
- Idea: $\lambda f : \tau \rightarrow \tau .\, \lambda x : \tau .\, f\,(f\,x)$
  becomes: $\lambda [\tau \rightarrow \tau] .\, \lambda [\tau] .\, 1\,(1\,0)$
- Main problems:
  - global variables
  - substitution requires shifting indices

Bound variables (iv)

Locally nameless is a combination of named variables and DeBruijn indices

- "Global" variables are named
- Bound variables are indices
- Substitution only
  - of bound variables (indices), especially index 0
  - with closed terms (without indices)
- Advantages
  - no need for renaming
  - no need for shifting

A tour of the proof

We proved the type safety of $\lambda^{\forall,\mathrm{ref}}$ incrementally

- simply typed λ-calculus

414 lines in Isabelle/HOL

A tour of the proof

We proved the type safety of $\lambda^{\forall,\mathrm{ref}}$ incrementally

- simply typed λ-calculus

  414 lines in Isabelle/HOL

- simply typed λ-calculus with references

  941 lines in Isabelle/HOL

A tour of the proof

We proved the type safety of $\lambda^{\forall,\mathrm{ref}}$ incrementally

- simply typed λ-calculus

  414 lines in Isabelle/HOL

- simply typed λ-calculus with references

  941 lines in Isabelle/HOL

- $\lambda^{\forall,\mathrm{ref}}$

  2,815 lines in Isabelle/HOL

Well formed environments

Environments are finite set of pairs, containing unique bindings for variables

- Type environment

  $\Delta \models \texttt{TY} \equiv \texttt{finite } \Delta$

- Term environment

  $\Gamma;\Delta \models \texttt{OK} \equiv$
  $\texttt{finite } \Gamma \wedge \Delta \models \texttt{TY} \wedge$
  $(\forall \texttt{x } \tau 1\ \tau 2.\ (\texttt{x} \triangleright \tau 1) \in \Gamma \wedge (\texttt{x} \triangleright \tau 2) \in \Gamma \longrightarrow$
  $\tau 1 = \tau 2 \wedge \Delta \models \tau 1)$

- Store

  $\texttt{S} \models \texttt{Store} \equiv$
  $\texttt{finite S} \wedge$
  $(\forall \texttt{x v1 v2.}\ (\texttt{x} \mapsto \texttt{v1}) \in \texttt{S} \wedge (\texttt{x} \mapsto \texttt{v2}) \in \texttt{S} \longrightarrow$
  $\texttt{v1} = \texttt{v2} \wedge \texttt{value v1})$

Store typing

The store in which a term is evaluated must correspond to the typing environments

$$\models \texttt{S} : \Gamma;\Delta \quad \equiv$$
$$\quad \texttt{S} \models \texttt{Store} \wedge$$
$$\quad (\forall \texttt{x}\ \tau.\ (\texttt{x} \triangleright \tau) \in \Gamma \longrightarrow (\exists \texttt{v}.\ (\texttt{x} \mapsto \texttt{v}) \in \texttt{S} \wedge \Gamma;\Delta \vdash \texttt{v} : \tau))$$

Standard lemmata/theorems (i)

- weakening: the environment can be extended with fresh bindings that are not used

Standard lemmata/theorems (i)

- weakening: the environment can be extended with fresh bindings that are not used
- substitution: typing is preserved when a term of the same type is substituted for a free variable

lemma substitution:
  assumes $\Gamma;\Delta \vdash$ e' : τ'
  and Γ,(x:τ');$\Delta \vdash$ vsubst_tm (TmFreeVar x) i e : τ
  and $\neg$ x free in e
  shows $\Gamma;\Delta \vdash$ vsubst_tm e' i e : τ

Standard lemmata/theorems *(ii)*

- preservation: operational semantics preserves typing

theorem `preservation`:
  assumes $\Gamma;\Delta \vdash e : \tau$
  and $S;e \hookrightarrow S';e'$
  and $\models S : \Gamma;\Delta$
  shows $\exists \Gamma'\ \Delta'.\ \Gamma \subseteq \Gamma' \wedge \Delta \subseteq \Delta' \wedge \models S' : \Gamma';\Delta' \wedge$
  $\Gamma';\Delta' \vdash e' : \tau$

Standard lemmata/theorems (iii)

- progress: a well-typed term is either a computed value or the operational semantics can make one more evaluation step

theorem `progress`:
  assumes $\Gamma;\Delta \vdash$ `e` : τ
  and $\models$ `S` : $\Gamma;\Delta$
  shows `not_stuck e S`

Lines of code in Isabelle/HOL

file	lines
Environ.thy	46
Syntax.thy	729
Typing.thy	757
Semantics.thy	138
Metatheory.thy	1,145
total	2,815

theorem/lemma	lines
preservation	274
progress	149
weakening of term environment	23
weakening of type environment	45
substitution of terms	192
substitution of types	360

Conclusions

- Mechanized proof of type-safety for $\lambda^{\forall,\mathrm{ref}}$ using Isabelle/HOL
- The first fully mechanized type safety proof for a language with mutable references and impredicative polymorphism

Future work

- Extend the language with explicit reference deallocation
  - Employ a substructural (linear) type system
  - This language should have a way of converting linear to unrestricted references
- Extend the linear language with a way to convert a linear reference to an unrestricted one