# *Mechanized Proofs of Type Safety for a Family of $\lambda$-Calculi with References*

Michalis A. Papakyriakou
Nikolaos S. Papaspyrou

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

{mpapakyr, nickie}@softlab.ntua.gr

2nd Informal ACM SIGPLAN Workshop on
Mechanizing Metatheory
Freiburg, Germany, October 4, 2007

# Outline

- Introduction
  - Where did we start from?
  - References and linearity
  - What is this talk about?
- A tour of our proofs
  - Simply typed lambda calculus
  - Adding references
  - Adding polymorphism
  - Adding linearity
- Conclusions

# *Where did we start from?*

- Shao, Trifonov, Saha and Papaspyrou,
  "A Type System for Certified Binaries",
  *ACM TOPLAS*, vol. 27, no. 1, pp. 1-45, 2005.
- The big picture:
  - low-level code with verifiable specifications
  - one type language: variant of the CIC — Coq
  - several computation languages
  - CL depends on TL, but not vice-versa
  - TL defines the logic and the type system of the CL
- What is missing from the CL:
  - references and destructive update
  - recursive data types
  - …

# *References* (i)

ML-style references

- Reference allocation

  $\texttt{let}\ r = \texttt{new}\ 7\ \texttt{in} \ldots$ $\quad r$ : ref int

- Assignment

  $\ldots r := 42 \ldots$ $\quad$ destructive update!

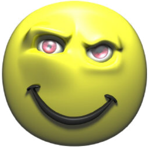
- Dereference

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

- No reference deallocation!

  $\ldots \texttt{free}\ r$ $\quad$ use garbage collection!

or do we want reference deallocation?



# References (ii)

But ML-style references are not enough in TSCB!

- We use singleton types for reasoning about computed values
- Suppose we start with an initial value
  $\texttt{let}\ r = \texttt{new}\ \overline{7}\ \texttt{in} \ldots$ $r : \mathsf{ref}\ (\mathsf{sint}\ \widehat{7})$
- and then we want to change it
  $\ldots r := \overline{42} \ldots$ destructive update!
- Type error!
  $r : \mathsf{ref}\ (\mathsf{sint}\ \widehat{7})$ $\overline{42} : \mathsf{sint}\ \widehat{42}$ $\mathsf{sint}\ \widehat{7} \neq \mathsf{sint}\ \widehat{42}$
- Strong update: the type changes!
- Can use weak update $r : \mathsf{ref}\ (\exists n : \mathbb{Z}.\, \mathsf{sint}\ n)$
  but then we cannot reason about $r$'s value

# References (iii)

How do we deal with this problem?

- Before changing the type of a reference, make sure that nobody else knows about it!
- Linear (substructural) type systems
- a linear reference $r : {}^{\mathsf{L}}\mathsf{ref}\ \tau$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathtt{new}\ e : {}^{\mathsf{L}}\mathsf{ref}\ \tau}\ \textit{(new)}$$

- an unrestricted reference $r : {}^{\mathsf{U}}\mathsf{ref}\ \tau$

$$\frac{\Gamma \vdash e : {}^{\mathsf{U}}\mathsf{ref}\ \tau}{\Gamma \vdash \mathtt{deref}\ e : \tau}\ \textit{(deref)}$$

# References (iv)

What do we get with a linear type system?

- Weak update — type is preserved

$$\frac{\Gamma \vdash e_1 : {}^{\mathsf{U}}\mathsf{ref}\,\tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 := e_2 : \mathsf{unit}}\ \textit{(weak)}$$

- Strong update — type changes

$$\frac{\Gamma \vdash e_1 : {}^{\mathsf{L}}\mathsf{ref}\,\tau \qquad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1 := e_2 : {}^{\mathsf{L}}\mathsf{ref}\,\tau'}\ \textit{(strong)}$$

- Deallocation

$$\frac{\Gamma \vdash e : {}^{\mathsf{L}}\mathsf{ref}\,\tau}{\Gamma \vdash \mathtt{free}\,e : \mathsf{unit}}\ \textit{(free)}$$

- But how do we convert a ${}^{\mathsf{L}}\mathsf{ref}\,\tau$ to a ${}^{\mathsf{U}}\mathsf{ref}\,\tau$ ?

# *Linear references* *(i)*

The let! construct $\texttt{let!}\ (x)\ y = e_1\ \texttt{in}\ e_2$

- Temporarily converts a $^{\mathsf{L}}\mathsf{ref}\ \tau$ to a $^{\mathsf{U}}\mathsf{ref}\ \tau$
- Example

$\texttt{let}\ r = \texttt{new}\ 6\ \texttt{in}$ $\qquad r : {}^{\mathsf{L}}\mathsf{ref\ int}$

$\texttt{let!}\ (r)$ $\qquad r : {}^{\mathsf{U}}\mathsf{ref\ int}$

$\quad y = (\texttt{let}\ a = \texttt{deref}\ r\ \texttt{in}$

$\qquad\qquad r := \texttt{deref}\ r + 1;$

$\qquad\qquad a * \texttt{deref}\ r)$

$\texttt{in}$ $\qquad r : {}^{\mathsf{L}}\mathsf{ref\ int}$

$\quad \texttt{free}\ r;$

$\quad \texttt{print}\ y$

# *Linear references* *(ii)*

How do we know let! is only temporary?

- The ${}^{\mathsf{U}}\mathsf{ref}\ \tau$ must not escape the scope of `let!`

$$
\begin{array}{ll}
\texttt{let } r = \texttt{new } 6 \texttt{ in} & r : {}^{\mathsf{L}}\mathsf{ref\ int} \\
\texttt{let! } (r) & r : {}^{\mathsf{U}}\mathsf{ref\ int} \\
\quad f = \lambda u\!:\!\mathsf{unit}.\, \texttt{deref } r & f : \mathsf{unit} \to \mathsf{int} \\
\texttt{in} & r : {}^{\mathsf{L}}\mathsf{ref\ int} \\
\quad \texttt{free } r;\ \ f\ () &
\end{array}
$$

- The unrestricted $r$ may escape by being
  - returned as (part of) the value computed by `let!`
  - used in function closures
  - assigned to other references
  - ...

# *Linear references (iii)*

How do we know let! is only temporary?

- Several solutions proposed
  - hyperstrict evaluation (Wadler, 1990)
  - observer types (Odersky, 1992)
  - adoption and focus (Fähndrich & DeLine, 2002)
  - ...
  - yet another (wannabe) solution (MP & NP, 2007+)
- Our work plan:
  - define a $\lambda$-calculus with references, polymorphism, linear types, let!
  - mechanically prove its type safety
  - extend it to fit in the TSCB framework

# *What is this talk about?*

- A family of languages

$$
\begin{tikzcd}
\lambda^{\to} \arrow[r, "\text{references}"] & \lambda^{\text{ref}} \arrow[r, "\text{polymorphism}"] \arrow[d, "\text{linearity}"] & \lambda^{\forall,\,\text{ref}} \arrow[dd, dotted] \\
 & \lambda^{\text{ref, free}} \arrow[d, "\text{let!}"] & \\
 & \lambda^{\text{ref, free, let!}} \arrow[r, dotted] & \lambda^{\forall,\,\text{ref, free, let!}}
\end{tikzcd}
$$

- The goal
  Proof of type safety (progress + preservation)
- The tools
  Isabelle/HOL, ISAR style, locally nameless

# *The basics: safety proof for $\lambda^{\rightarrow}$* *(i)*

- Abstract syntax

  $$\tau \;::=\; b \mid \tau \rightarrow \tau$$

  $$e \;::=\; x \mid \lambda x{:}\tau.\, e \mid e_1\, e_2$$

- Environments are sets of pairs $(x \triangleright \tau)$:
  finite and consistent

# *The basics: safety proof for $\lambda^{\rightarrow}$ (ii)*

- Typing in locally nameless $\Gamma \vdash \texttt{e} : \tau$

```
inductive Typing intros
  t_var: ⟦ Γ ⊢ OK; (x▷τ)∈Γ ⟧ ⟹ Γ ⊢ TmFreeVar x : τ
  t_app: ⟦ Γ ⊢ e1 : τ1→τ2; Γ ⊢ e2 : τ1 ⟧ ⟹
             Γ ⊢ e1·e2 : τ2
  t_abs: ⟦ Γ ⊢ OK; finite L;
           ∀x. ¬ x free in e ∧ ¬ Γ defines x ∧ x∉L ⟶
                Γ,(x:τ1) ⊢ freshen_tm x e : τ2 ⟧ ⟹
             Γ ⊢ λ[τ1]. e : τ1→τ2
```

# *Adding references: $\lambda^{\text{ref}}$* *(i)*

- Abstract syntax

$$\tau \;::=\; b \mid \tau \to \tau \mid \mathsf{ref}\ \tau$$

$$\begin{aligned} e \;::=\; & c \mid x \mid \lambda x{:}\tau.\, e \mid e_1\, e_2 \\ \mid\; & \mathtt{new}\ e \mid e_1 := e_2 \mid \mathtt{deref}\ e \mid \mathtt{loc}\ \ell \end{aligned}$$

- Values

$$v \;::=\; c \mid \lambda x{:}\tau.\, e \mid \mathtt{loc}\ \ell$$

- Stores are sets of pairs $(\ell \mapsto v)$
- It simplifies things to take $\ell \equiv x$
- Typing still uses one environment $\quad \Gamma \vdash \mathtt{e} : \tau$

# *Adding references:* $\lambda^{\text{ref}}$ *(ii)*

- It further simplifies things to use variables as the real values
- Semantics with "temporaries" `S;e↪S';e'`

```
inductive Eval intros
  e_val: ⟦ ¬ S defines z; S ⊨ Store; value v ⟧ ⟹
           S;v ↪ S,(z↦v);TmFreeVar z
  e_beta: ⟦ S;z⇓λ[τ].e1 ⟧ ⟹
      S;(TmFreeVar z)·(TmFreeVar y) ↪ S;freshen_tm y e1
```

- Store typing $\models$ S : $\Gamma$
- In preservation, S and $\Gamma$ expand
  - temporaries are added — computed values
  - locations are added — allocated objects

# *Adding polymorphism: $\lambda^{\forall,\mathrm{ref}}$* *(i)*

- Abstract syntax

$$
\begin{array}{lcl}
\tau & ::= & b \mid \tau \to \tau \mid \mathsf{ref}\,\tau \mid \alpha \mid \forall\alpha.\,\tau \\
e & ::= & c \mid x \mid \lambda x{:}\tau.\,e \mid e_1\,e_2 \mid \Lambda\alpha.\,e \mid e\,[\tau] \\
 & \mid & \texttt{new}\,e \mid e_1 := e_2 \mid \texttt{deref}\,e \mid \texttt{loc}\,\ell \\
v & ::= & c \mid \lambda x{:}\tau.\,e \mid \Lambda\alpha.\,v \mid \texttt{loc}\,\ell
\end{array}
$$

- Substitution of types and terms
- Two substitution lemmata

# *Adding polymorphism:* $\lambda^{\forall,\mathrm{ref}}$ *(ii)*

- In the type substitution lemma, at some point in the case `e = Λ[*]. e_b`, we must show

$$\tau_1\{i+1 \mapsto x\}\{0 \mapsto \tau_2\{i \mapsto x\}\} = \tau\{i \mapsto x\}$$

$$\Longrightarrow \quad \tau_1\{i+1 \mapsto \tau'\}\{0 \mapsto \tau_2\{i \mapsto \tau'\}\} = \tau\{i \mapsto \tau'\}$$

  (provided all mentioned types are closed and $x$ is not free in $\tau, \tau_1, \tau_2$)
- Easier to generalize: substitution functions
  - meta-level functions representing contexts
  - mapping closed terms to terms

# *Adding polymorphism:* $\lambda^{\forall,\mathrm{ref}}$ *(ii)*

- In the type substitution lemma, at some point in the case `e = Λ[*]. e_b`, we must show

$$f(x) = g(x)$$
$$\Longrightarrow \quad f(\tau') = g(\tau')$$

  (provided all mentioned types are closed and $x$ is not free in $\tau, \tau_1, \tau_2$)
- Easier to generalize: substitution functions
  - meta-level functions representing contexts
  - mapping closed terms to terms

# *Adding linearity:* $\lambda^{\text{ref, free}}$ *(i)*

- Abstract syntax

$$
\begin{array}{lcll}
q & ::= & \mathsf{L} \mid \mathsf{U} & \text{qualifiers} \\
\varphi & ::= & b \mid \tau \to \tau \mid \mathsf{ref}\ \tau & \text{pretypes} \\
\tau & ::= & {}^{q}\varphi & \text{types} \\
e & ::= & {}^{q}c \mid x \mid {}^{q}\lambda x{:}\tau.\, e \mid e_1\, e_2 & \\
 & \mid & \mathtt{new}\ e \mid e_1 := e_2 \mid \mathtt{deref}\ e \mid {}^{q}\mathtt{loc}\ \ell & \\
 & \mid & \mathtt{free}\ e \mid e_1 :=: e_2 &
\end{array}
$$

- Two additional constructs
  - Explicit deallocation
  - Swapping: assignment without losing the previous contents

# *Adding linearity:* $\lambda^{\text{ref, free}}$ *(ii)*

- Easier to separate temporaries from locations
- Stores are sets of pairs $(x \mapsto v)$ — temporaries
- Memories are sets of pairs $(\ell \mapsto x)$ — locations
- Looking up the store: linear values are removed
- Compatible type environments $\Gamma 1 \sim \Gamma 2$
- Typing uses two environments $\Gamma;\mathtt{M} \vdash \mathtt{e} : \tau$

```
inductive Typing intros
  t_app: ⟦ Γ1 ∼ Γ2; Γ1∪Γ2 ⊨ OK;
           Γ1;M ⊢ e1 : Qual q (τ1→τ2);
           Γ2;M ⊢ e2 : τ1 ⟧ ⟹
             Γ1∪Γ2;M ⊢ e1·e2 : τ2
```

# *Adding linearity:* $\lambda^{\text{ref, free}}$ *(iii)*

- Substitution lemma: can only substitute free variables for DeBruijn indices
- Store typing and memory typing are inductively defined
- Two invariants on stores
  - locations are only top-level, i.e. $(x \mapsto {}^{q}\texttt{loc}\ \ell)$
  - linear locations appear only once
- Hack: the semantics of `new` places the new location in the store and returns a temporary

# *Adding linearity:* $\lambda^{\text{ref, free}}$ *(iv)*

- Preservation

$$\left.\begin{array}{l} S;\mu;e \hookrightarrow S';\mu';e' \\ \Gamma_e;\emptyset \vdash e : \tau \\ M \models S : \Gamma_s \cup \Gamma_m \\ \Gamma_m \models \mu : M \\ \Gamma_s \sim \Gamma_m \\ \text{invariants}(S) \\ \Gamma_e \cup \Gamma_r \subseteq \Gamma_s \\ \Gamma_e \sim \Gamma_r \end{array}\right\} \implies \begin{array}{l} \exists \Gamma'_e, \Gamma'_s, \Gamma'_m, M'. \\ \quad \Gamma'_e;\emptyset \vdash e' : \tau \\ \quad M' \models S' : \Gamma'_s \cup \Gamma'_m \\ \quad \Gamma'_m \models \mu' : M' \\ \quad \Gamma'_s \sim \Gamma'_m \\ \quad \text{invariants}(S') \\ \quad \Gamma'_e \cup \Gamma_r \subseteq \Gamma'_s \\ \quad \Gamma'_e \sim \Gamma_r \end{array}$$

- $\Gamma_r$ contains temporaries that have been used elsewhere in the evaluation

# *Adding linearity:* $\lambda^{\text{ref, free}}$ (v)

- Progress

$$
\left.\begin{array}{l}
\Gamma_e; \emptyset \vdash e : \tau \\
\Gamma_e \subseteq \Gamma_s \\
M \models S : \Gamma_s \cup \Gamma_m \\
\Gamma_m \models \mu : M \\
\text{invariants}(S)
\end{array}\right. \implies \text{not\_stuck}(e, S, \mu)
$$

# *Adding linearity:* $\lambda^{\text{ref, free}}$ *(vi)*

| File | $\lambda^{\rightarrow}$ | $\lambda^{\text{ref}}$ | $\lambda^{\forall,\text{ref}}$ | $\lambda^{\text{ref, free}}$ |
|---|---|---|---|---|
| `Environ.thy` | 46 | 46 | 46 | 46 |
| `Syntax.thy` | 94 | 116 | 699 | 139 |
| `Typing.thy` | 74 | 83 | 738 | 366 |
| `Semantics.thy` | 47 | 143 | 138 | 231 |
| `Metatheory.thy` | 153 | 553 | 1151 | 2865 |
| Total | 414 | 941 | 2772 | 3647 |

# *Adding linearity:* $\lambda^{\text{ref, free}}$ *(vi)*

| File | $\lambda^{\text{ref, free}}$ |
|---|---|
| `Environ.thy` | 46 |
| `Syntax.thy` | 139 |
| `Typing.thy` | 366 |
| `Semantics.thy` | 231 |
| `Metatheory.thy` | 2865 |
| Total | 3647 |

$\longrightarrow$

| | |
|---|---|
| weakening | 5 |
| substitution | 489 |
| store typing | 565 |
| invariants | 75 |
| preservation | 1360 |
| progress | 326 |
| safety | 32 |

# *Conclusions*

- Related work: fully fledged languages with polymorphism and references
  - ML (Dubois, 2000; Lee, Crary & Harper, 2007)
  - Java (von Oheimb, 2001; Leavens, Naumann & Rosenberg, 2006)
  - references and impredicative polymorphism?
- Related work: references and linear type systems
  - Walker & Watkins, 2001
  - Fluet, Morrisett & Ahmed, 2005, 2006
  - ...
- Contribution
  - mechanized proofs of type safety for $\lambda^{\forall,\,\mathrm{ref}}$ and $\lambda^{\mathrm{ref},\,\mathrm{free}}$ in Isabelle/HOL

# Thank you...
# Questions?

**by auto**

```
*** Terminal proof method failed
*** At command "by".
```

**sorry**