Mechanizing the Metatheory of a Language
With Linear Resources and Context Effects

Daniel K. Lee
Carnegie Mellon University

Derek Dreyer, Andreas Rossberg
Max Planck Institute for Software Systems

September 20, 2008
RTG Language Overview

- RTG for *recursive type generativity*.
- Invented for recursive module calculi [Dreyer, ICFP ’05].
- We formalize a variant used as an IL for mixin modules [Dreyer, Rossberg, ICFP ’08].
Key Features of RTG

- RTG is a module calculus with terms, type constructors, and modules.
- Key feature of the language is ability to forward declare types and define them later.
  - The variant we formalize allows for circular definitions.
- Useful for separately defining two modules whose type components might refer to each other, among other things.
Mechanizing the Metatheory of RTG

- Used Twelf in a formalization and type safety proof for RTG.
- Type definitions in RTG required advanced Twelf encoding techniques.
  - Utilized a variation on technique for encoding linearity.
  - Lead to an improvement to explicit contexts technique.
RTG Syntax

\[
\text{types} \quad A ::= \ldots \mid \alpha
\]

\[
\text{modules} \quad M ::= \ldots \mid \text{new } \alpha.M \mid \text{def } \alpha := A \text{ in } M
\]

- new \( \alpha.M \) introduces a binding.
- def \( \alpha := A \text{ in } M \) defines a variable already in scope.
RTG Type Contexts

typing contexts  \( \Gamma ::= \cdot \mid \Gamma, \alpha \) type

definition contexts  \( \Delta ::= \cdot \mid \Delta, \alpha ::= ? \mid \Delta, \alpha ::= A \)

- \( \alpha ::= ? \) is a consumable definability resource for \( \alpha \), denoting \( \alpha \) is waiting for a definition.

- \( \alpha ::= A \) is an unrestricted definition resource for \( \alpha \), denoting \( \alpha \) is equivalent to \( A \).
RTG Typing Rules

\[ \Gamma, \alpha \text{ type}; \Delta, \alpha :=? \vdash M : S \]
\[ \Gamma \vdash S \text{ sig} \quad \alpha \text{ defined once in } M \]
\[ \Gamma; \Delta \vdash \text{new } \alpha. M : S \]

- Syntactic restriction that type variables created by \texttt{new} are only defined once.
RTG Typing Rules

\[
\Gamma \vdash A \text{ type} \quad \Gamma; \Delta, \alpha := A \vdash M : S
\]

\[
\Gamma; \Delta, \alpha :=? \vdash \text{def } \alpha := A \text{ in } M : S
\]

- Inserting a definition into \( \Delta, \alpha :=? \) to yield \( \Delta, \alpha := A \) is a context effect.
Roadmap

- Linearity of definability is enforced with a judgment on syntax.
- Encoding definition context requires a new form of explicit contexts.
- Circular definitions complicate metatheory of type equivalence.
Definability as a Linear Resource

- Type safety requires at most one definition for a variable.
- Requires variables not be defined more than once.
- Definability is treated as a linear resource.
Typical Encoding of Linearity

\[
md : \text{type}.
\]

\[
\ldots
\]

\[
\text{linear} : (md \to md) \to \text{type}.
\]

\[
\lhd \text{linear ([a] M a) witnesses M is a single hole context.}
\]
Typical Encoding of Linearity

\[
\text{md/pair} : \text{md} \rightarrow \text{md} \rightarrow \text{md}. \quad \% <\text{M1}, \text{M2}>
\]

\[
\ldots
\]

\[
\text{linear/var} \quad : \quad \text{linear} \left([m] \ m\right).
\]

\[
\text{linear/pair1} \quad : \quad \text{linear} \left([m] \ \text{md/pair} \left(\text{M1} \ m\right) \ \text{M2}\right)
\]
\[
\quad \leftarrow \quad \text{linear} \left([m] \ \text{M1} \ m\right).
\]

\[
\text{linear/pair2} \quad : \quad \text{linear} \left([m] \ \text{md/pair} \ \text{M1} \ (\text{M2} \ m)\right)
\]
\[
\quad \leftarrow \quad \text{linear} \left([m] \ \text{M2} \ m\right).
\]

- Enforce linearity by restricting subterm in which variable can appear.
Linear Definability

\[
\text{defonce} : (\text{tp} \rightarrow \text{md}) \rightarrow \text{type}.
\]

\[
\text{defzero} : (\text{tp} \rightarrow \text{md}) \rightarrow \text{type}.
\]

- Linearity is too strict, want define once, not appears once.
- Use a \text{defonce} judgment to witness linearity of definitions.
- Requires a \text{defzero} judgment to witness absence of definitions.
Linear Definability

\[
\begin{align*}
defonce/pair1 & : \text{defonce } ([a] \text{ md/pair } (M_1 a) (M_2 a)) \\
& \quad \leftarrow \text{defonce } ([a] M_1 a) \\
& \quad \leftarrow \text{defzero } ([a] M_2 a).
defonce/pair2 & : \text{defonce } ([a] \text{ md/pair } (M_1 a) (M_2 a)) \\
& \quad \leftarrow \text{defzero } ([a] M_1 a) \\
& \quad \leftarrow \text{defonce } ([a] M_2 a).
\end{align*}
\]

- defonce defined similarly to linear, but using defzero instead of syntactic irrelevance.
Roadmap

- Linearity of definability is enforced with a judgment on syntax.
- Encoding definition context requires a new form of explicit contexts.
- Circular definitions complicate metatheory of type equivalence.
Definition Contexts

\[ \Gamma \vdash A \text{ type} \quad \Gamma; \Delta, \alpha := A \vdash M : S \]
\[ \Gamma; \Delta, \alpha :=? \vdash \text{def } \alpha := A \text{ in } M : S \]

- Definability is a *consumable resource*.
- A definition is an unrestricted resource.
- Safety requires adding a definition for a variable “consumes” its definability.
Contexts in LF

- Standard practice is to encode assumptions about variables implicitly using the LF context.
- Twelf world checking treats all such assumptions as unrestricted, so it cannot enforce
  - linearity of definability.
  - uniqueness of definitions.
- Encoding RTG requires a first order representation of definition context.
Explicit Contexts

- Advanced Twelf technique to prove theorems not directly provable with HOAS encodings. [Crary, WMM 06, LFMTP 08]
- Typically used a proof device, rather than formalization technique [Lee et al., POPL 07].
- We use explicit contexts as part of our formalism, to encode definition contexts.
Typical Explicit Contexts

cxt : type.

nil : cxt.
cons : cxt -> tm -> tp -> cxt.

% Used with worlds to distinguish variables
% Also assigns "ordering tokens" to them.
isvar : tm -> nat -> type.

% Uses ordering tokens to check cxt is ordered.
ordered : cxt -> type.

% Looks up from context.
lookup : cxt -> tm -> tp -> type.
Typical Explicit Contexts Encoding of Typing for STLC

\[
\text{ofe/lam : } \text{ofe } G \ (\text{lam } ([x] E x)) \ (\text{arr } T1 \ T2) \\
\quad \leftarrow (\{x\} \ \text{isvar} \ x \ I \\
\quad \quad \rightarrow \ \text{ofe } (\text{cons } G \ x \ T1) \ (E \ x) \ T2).
\]

\[
\text{ofe/var : } \text{ofe } G \ E \ T \\
\quad \leftarrow \ \text{ordered } G \quad \% \ \text{isvar } E \ I \ \text{is implicit} \\
\quad \leftarrow \ \text{lookup } G \ E \ T. \quad \% \ \text{via def of ordered}
\]
Proofs with typical explicit contexts may require “freshening” of ordering tokens and re-writing derivations.

- Requires induction metrics.
The Reordering Lemma

reorder
  : ({x} isvar x I -> ofe (G x) (M x) A)
      -> ({x} isvar x I' -> ordered (G x))

%%
  -> ({x} isvar x I' -> ofe (G x) (M x) A)
  -> type.
%mode reorder +D1 +D2 -D3.

▶ Called “bump” in Crary ’08.
▶ “α-vary” ordering token.
▶ Necessary to prove weakening.
▶ Tedious to prove.
An Improved Explicit Contexts Technique

- Ordering tokens and context well-formedness assumptions in derivations are a pain.
- Just remove them!
- When necessary, pass in and maintain context well-formedness assumptions to metatheorems.
  - Only need to maintain ordering tokens in metatheorems where context well-formedness matters.
  - A common paper technique.
% like isvar, but omits ordering token
var : tm -> type.

ofe/lam : ofe G (lam ([x] E x)) (arr T1 T2)
<- ({x} var x
    -> ofe (cons G x T1) (E x) T2).

ofe/var : ofe G E T
<- var E
<- lookup G E T.
Eliminating the Reordering Lemma

reorder2
: (\{x\} var x -> ofe (G x) (M x) A)
%%
  -> (\{x\} var x -> ofe (G x) (M x) A)
  -> type.
%mode reorder2 +D1 -D2.

- Eliminating ordering tokens from derivations makes reordering lemma trivial.
- Simplifies a big annoyance when using explicit contexts.
We encoded RTG with a hybrid of HOAS and explicit contexts.

- Use LF context for typing assumptions.
- Use simplified explicit contexts for definition context.
  - Assigning a definition to a variable encoded as an operation on explicit representation of definition contexts.
Simplified Explicit Contexts Necessary for Hybrid Approach

▶ Proving reorder lemma would require explicit contexts for both definition and typing contexts. (painful!)
▶ Simplified explicit contexts make hybrid encoding possible.
▶ Demands of formalizing RTG led us to discover simplified explicit contexts.
Roadmap

- Linearity of definability is enforced with a judgment on syntax.
- Encoding definition context requires a new form of explicit contexts.
- Circular definitions complicate metatheory of type equivalence.
Proving Type Safety

- Preservation and progress must account for definitions, but are mostly standard.
- Hole in development related to injectivity of type equality.
Type Constructor Equality

$$\Gamma; \Delta, \alpha := A \vdash \alpha \equiv A$$

- RTG’s has higher order types with definitions [Stone’s ATTAPL Chapter]
- $\beta$-, $\eta$- equality with higher-kindred type constructors.
- $\delta$- equality, due to definitions.
- Circular definitions mean no normal forms.
Injectivity and Inequality

- Canonical forms lemma require inequality properties:
  - If $A_1 \rightarrow A_2 \equiv \text{unit}$, then false.

- Preservation requires injectivity:
  - If $A_1 \rightarrow A_2 \equiv B_1 \rightarrow B_2$, then $A_1 \equiv B_1$ and $A_2 \equiv B_2$. 
Proving injectivity and inequality

- Injectivity usually proven with normalization-based techniques.
  - Twelf proofs of injectivity rely on techniques based on notions of *hereditary substitution* that preserve normal forms.
- Circular definitions rule out normal forms.
- Injectivity still provable with a logical relations argument.
- Encoding injectivity proof using Twelf an open question.
  - Currently assumed using `%trustme`. 
Future Work

- Extend RTG formalization with more features.
- Formalize an elaboration into RTG.
- Use other tools to formalize RTG for comparison
  - A Coq development is in progress.
Conclusion

- Formalized and proved type safety for a language with a number of non-standard features.
  - Linear Resources
  - Context Effects
- Other interesting issues not discussed in talk:
  - Actual formalism accounts for more features.
  - Dealing with adding type variables to environment during evaluation.
Questions?
A Hybrid Approach

cxt : type.

nil : cxt.

% D, a := ?
nodef : cxt -> tp -> cxt.

% D, a := A
def : cxt -> tp -> tp -> cxt.

- Use HOAS for typing assumptions.
- Use explicit contexts for definition context.
Linear Definability

... md/deftp : tp -> tp -> md -> md. % a := A in M ...

defzero/deftp : defzero ([a] md/deftp A1 (A2 a) (M a))
<- defzero ([a] M a)).
...
defonce/deftp1 : defonce ([a] md/deftp a (A a) (M a))
defonce/deftp2 : defonce ([a] md/deftp A1 (A2 a) (M a))

▶ Enforces a variable is used linearly in definition positions.
▶ Allows unrestricted use in other places type constructors can appear.
A Hybrid Approach

\[ \Gamma, \alpha \text{ type; } \Delta, \alpha \coloneqq A \vdash M : S \quad \Gamma \vdash S \text{ sig} \]
\[ \Gamma; \Delta \vdash \text{new } \alpha.M : S \]

\text{of/newtp} : \text{of } D (\text{md/newtp } ([a] M a)) \quad S
\text{<- } \{a\} \text{ tp-wf } a
\text{<- } \text{isdefvar } a
\text{<- } \text{of } (\text{nodef } D a) (M a) \quad S
\text{<- } \text{defonce } ([a] M a).
A Hybrid Approach

\[ \Gamma \vdash A_2 \text{ type} \quad \Delta, \alpha_1 := A_2; \Gamma \vdash M : S \]
\[ \Delta, \alpha_1 := ?; \Gamma \vdash \text{def } \alpha_1 := A_2 \text{ in } M : S \]

of/deftp : of D (md/deftp A1 A2 M) S
- <- isdefvar A1
- <- tp-wf A2
- <- define D A1 A2 D'
- <- of D' M S.