A Framework for Specifying, Prototyping, and Reasoning about Computational Systems

Andrew Gacek

Department of Computer Science and Engineering
University of Minnesota

PhD Defense
September 8, 2009
Motivation

We are interested in a framework for developing *formal systems*

Some example formal systems:

- Evaluation and typing in a programming language
- Provability in a logic
- Behavior in a concurrency system

A framework should support:

- Specification, prototyping, reasoning
- Working with objects with variable binding structure
Our Approach to Building a Framework

A logic-based approach:

- A *specification logic* which encodes formal systems through logical formulas
- Prototyping via a computational interpretation of the specification logic
- A *reasoning logic* which can internalize the specification logic and be used to prove properties of specifications

A higher-order approach:

- Both logics incorporate the \( \lambda \)-calculus in their term structure so we can represent binding
- They contain logical devices for analyzing such structure
Contributions

- The logic $G$ for reasoning about specifications
- Abella: an implementation of $G$ which incorporates the two-level logic approach to reasoning
- Rich examples constructed in Abella which verify the power of $G$ and the usefulness and practicality of the two-level logic approach to reasoning
Example: Mini-ML

Mini-ML Syntax

\[ a ::= \text{int} \mid a \rightarrow a \]
\[ t ::= x \mid t \ t \mid (\text{fn} \ x:a \Rightarrow t) \]

Mini-ML Evaluation

\[ t \downarrow v \text{ means } t \text{ evaluates to } v \]

\[ \frac{}{(\text{fn} \ x:a \Rightarrow r) \downarrow (\text{fn} \ x:a \Rightarrow r)} \]

\[ \frac{m \downarrow (\text{fn} \ x:a \Rightarrow r) \quad r[x := n] \downarrow v}{m \ n \downarrow v} \]
Reasoning about Mini-ML

Theorem (Determinacy of Evaluation)

If \( t \Downarrow v \) and \( t \Downarrow w \) then \( v = w \)

Proof.
Induction on the derivation of \( t \Downarrow v \)
Proceed by cases,

- \( t \) and \( v \) are both \((\text{fn } x : a \Rightarrow r)\)
  Must be that \( w \) is \((\text{fn } x : a \Rightarrow r)\)
- \( t \) is \( mn \)
  - Must have \( m \Downarrow (\text{fn } x : a \Rightarrow r) \) and \( r[x := n] \Downarrow v \)
  - Must have \( m \Downarrow (\text{fn } x : b \Rightarrow s) \) and \( s[x := n] \Downarrow w \)
  - By induction \( r = s \), and thus by induction \( v = w \)
A Higher-order Abstract Syntax Representation

Object level binding can be represented with meta-level abstraction

Constants for Mini-ML

\[ \text{int} :: \text{type} \]
\[ \text{arrow} :: \text{type} \rightarrow \text{type} \rightarrow \text{type} \]
\[ \text{app} :: \text{term} \rightarrow \text{term} \rightarrow \text{term} \]
\[ \text{fun} :: \text{type} \rightarrow (\text{term} \rightarrow \text{term}) \rightarrow \text{term} \]

Example

\[ \text{fn} \ x : \text{int} \Rightarrow \text{fn} \ y : \text{int} \Rightarrow x \]
\[ \text{fun} \ \text{int} \ (\lambda x. \text{fun} \ \text{int} \ (\lambda y. x)) \]

Binding issues are now treated in the meta-level
Basic Structure for Reasoning

- Formulas over expressions from the simply-typed $\lambda$-calculus
- Atomic formulas encode object system judgments
- Relationships between judgments can be expressed with logical formulas
- The formal system provides a means for deriving sequents of the form:
  $$H_1, \ldots, H_n \rightarrow C$$
Some Core Rules of the Logic

\[ \Gamma, B \rightarrow B \quad \text{id} \]

\[ \Gamma, B, \Gamma \rightarrow C \quad \text{cut} \]

\[ \Gamma, \bot \rightarrow C \quad \bot L \]

\[ B \rightarrow C \quad \bot R \]

\[ \Gamma, B_i \rightarrow C \quad \land L_i \]

\[ \Gamma, B \rightarrow B \land C \quad \land R \]

\[ \Gamma, B \rightarrow B \cup C \quad \cup R \]

\[ \Gamma, B[h/x] \rightarrow C \quad \exists L \]

\[ \Gamma, \exists x. B \rightarrow C \quad \exists R \]
Definitions

The syntax of definitions: \( \forall \vec{x}. H(\vec{x}) \triangleq B(\vec{x}) \)

Atomic formulas are interpreted as fixed-points of such definitions

\[
eval \ (\text{fun} \ A \ R) \ (\text{fun} \ A \ R) \triangleq \top
\]

\[
eval \ (\text{app} \ M \ N) \ V \triangleq \exists A. \exists R. \ eval \ M \ (\text{fun} \ A \ R) \land \ eval \ (R \ N) \ V
\]

We can encode this in a single definitional clause:

\[
eval \ T \ V \triangleq (\exists A, R. \ T = (\text{fun} \ A \ R) \land V = (\text{fun} \ A \ R)) \lor
\]

\[
(\exists M, N, A, R. \ T = (\text{app} \ M \ N) \land 
 eval \ M \ (\text{fun} \ A \ R) \land eval \ (R \ N) \ V)
\]
Let $p$ be defined by

$$
\forall \vec{x}. p \vec{x} \triangleq B \ p \vec{x}
$$

We also have rules for induction and co-induction for appropriate definitions
Formally Proving Determinacy of Evaluation

Theorem
\( \forall t, v, w. (\text{eval } t \ v \land \text{eval } t \ w) \supset v = w \)

Proof.
Apply rules for \( \forall \), \( \land \), and \( \supset \)

\[ \text{eval } t \ v, \text{eval } t \ w \rightarrow v = w \]

Case analysis on eval \( t \ v \)

- \( t = v = (\text{fun } a \ r) \)

\[ \text{eval } (\text{fun } a \ r) \ w \rightarrow (\text{fun } a \ r) = w \]

Case analysis on eval \( (\text{fun } a \ r) \ w \)

\[ \rightarrow (\text{fun } a \ r) = (\text{fun } a \ r) \]

- \( t = (\text{app } m \ n) \ldots \)
Dynamic Aspects of Binding

Consider a typing judgment for Mini-ML

\[
\frac{x : a \in \Gamma}{\Gamma \vdash x : a} \quad \frac{\Gamma \vdash m : a \rightarrow b \quad \Gamma \vdash n : a}{\Gamma \vdash m \ n : b}
\]

\[
\frac{\Gamma, x : a \vdash r : b}{\Gamma \vdash (\text{fn } x : a \rightarrow r) : a \rightarrow b} \quad x \notin \text{dom}(\Gamma)
\]

\[
of \ \Gamma \ X \ A \triangleq \text{member (} X : A \text{) } \Gamma
\]

\[
of \ \Gamma \ (\text{app } M \ N) \ B \triangleq \exists A. \ of \ \Gamma \ M \ (\text{arrow } A \ B) \land \ of \ \Gamma \ N \ A
\]

\[
of \ \Gamma \ (\text{fun } A \ R) \ (\text{arrow } A \ B) \triangleq \nabla x. \ of \ ((x : A) :: \Gamma) \ (R \ x) \ B
\]
Some Properties of the $\nabla$ Quantifier

$\nabla x. F$ introduces a fresh “variable name” for $x$

We have the following structural properties for $\nabla$:

$$\nabla x. \nabla y. F \equiv \nabla y. \nabla x. F$$

$$\nabla x. F \equiv F \quad \text{if } x \text{ does not appear in } F$$

If we allow $\nabla$ quantification at a type, then we assume there are infinitely many fresh names at that type
Logical Rules for the $\nabla$ Quantifier

$$\frac{B[a/x], \Gamma \rightarrow C}{\nabla x. B, \Gamma \rightarrow C} \quad \nabla L$$

$$\frac{\Gamma \rightarrow B[a/x]}{\Gamma \rightarrow \nabla x. B} \quad \nabla R$$

$a$ is a nominal constant not appearing in $B$

The treatment of nominal constants requires permutations of nominal constants to be considered in the equivalence of formulas.

In particular, we change the initial rule to

$$\frac{}{\Gamma, B \rightarrow B', \text{id, if } B = \pi. B'}$$
Typing Example with $\nabla$

\[
of \Gamma \ X \ A \triangleq \text{member} \ (X : A) \ \Gamma \\
of \Gamma \ (\text{app} \ M \ N) \ B \triangleq \exists A. \ of \ \Gamma \ M \ (\text{arrow} \ A \ B) \ \land \ of \ \Gamma \ N \ A \\
of \Gamma \ (\text{fun} \ A \ R) \ (\text{arrow} \ A \ B) \triangleq \nabla x. \ of \ ((x : A) :: \Gamma) \ (R \ x) \ B
\]

\[
\begin{align*}
\vdots \\
\rightarrow & \ \text{member} \ (c : \text{int}) \ ((d : \text{int}) :: (c : \text{int}) :: \text{nil}) \\
\rightarrow & \ of \ ((d : \text{int}) :: (c : \text{int}) :: \text{nil}) \ c \ \text{int} \\
\rightarrow & \ \nabla x. \ of \ ((x : \text{int}) :: (c : \text{int}) :: \text{nil}) \ c \ \text{int} \\
\rightarrow & \ of \ ((c : \text{int}) :: \text{nil}) \ (\text{fun} \ \text{int} \ (\lambda y. \ c)) \ (\text{arrow} \ \text{int} \ \text{int}) \\
\rightarrow & \ \nabla x. \ of \ ((x : \text{int}) :: \text{nil}) \ (\text{fun} \ \text{int} \ (\lambda y. \ x)) \ (\text{arrow} \ \text{int} \ \text{int}) \\
\rightarrow & \ of \ \text{nil} \ (\text{fun} \ \text{int} \ (\lambda x. \ \text{fun} \ \text{int} \ (\lambda y. \ x))) \ (\text{arrow} \ \text{int} \ (\text{arrow} \ \text{int} \ \text{int}))
\end{align*}
\]
Reasoning about Type Uniqueness

\[ \forall t, a, b. (\text{of } nil \ t \ a \land \text{of } nil \ t \ b) \supset a = b \]

\[ \forall \Gamma, t, a, b. (\text{of } \Gamma \ t \ a \land \text{of } \Gamma \ t \ b) \supset a = b \]

\[ \forall \Gamma, t, a, b. (\text{cntx } \Gamma \land \text{of } \Gamma \ t \ a \land \text{of } \Gamma \ t \ b) \supset a = b \]

\text{cntx } \Gamma \text{ should enforce}

- \( \Gamma = (x_1 : a_1) :: (x_2 : a_2) :: \ldots :: (x_n : a_n) :: \text{nil} \)
- Each \( x_i \) is atomic
- Each \( x_i \) is unique

Definitions can serve to capture such meta-level properties

\text{cntx } \text{nil} \triangleq \top

\text{cntx } ((X : A) :: L) \triangleq \text{“} X \text{ atomic and not occurring in } L \text{“} \land \text{cntx } L
Analyzing Occurrences of Nominal Constants

We introduce the device of *nominal abstraction*:

\[(\lambda x_1 \ldots \lambda x_n.s) \triangleright t\]

This holds exactly when there exist nominal constants \(c_1, \ldots, c_n\) such that \((\lambda x_1 \ldots \lambda x_n.s)\) is equal to \((\lambda c_1 \ldots \lambda c_n.t)\)

**Examples**

- “\(X\) is atomic”
  \((\lambda z.z) \triangleright X\)

- “\(X\) is atomic and does not occur in \(L\)”
  \((\lambda z.fresh\ z\ L) \triangleright fresh\ X\ L\)
Nominal Abstraction as a Modular Extension of Equality

\[ \Gamma \rightarrow t = t = R \]

\[ \{ \Gamma[\theta] \rightarrow C[\theta] \mid \text{all } \theta \text{ such that } (s = t)[\theta] \} = L \]

\[ s = t, \Gamma \rightarrow C \]

\[ \Gamma \rightarrow s \triangleright t \triangleright R, \text{ if } s \triangleright t \text{ holds} \]

\[ \{ \Gamma[\theta] \rightarrow C[\theta] \mid \text{all } \theta \text{ such that } (s \triangleright t)[\theta] \} \triangleright L \]

\[ s \triangleright t, \Gamma \rightarrow C \]

\( \cdot[\cdot] \) is a generalized notion of substitution which respects the scope of nominal constants.
Summary of the Logic $G$

We have a logic with . . .

- simply-typed $\lambda$-terms for representation
- atomic formulas for encoding judgments
- fixed-point definitions for encoding rules
- induction (and co-induction) over appropriate fixed-point definitions
- $\nabla$ quantifier for introducing fresh names
- nominal abstraction for analyzing occurrences of names
Cut and Cut-elimination

\[ \frac{\Gamma \rightarrow B \quad B, \Gamma \rightarrow C}{\Gamma \rightarrow C} \quad \text{cut} \]

Cut is useful for...

- using lemmas during reasoning
- enabling shorter proofs
- allowing flexible proof construction

Cut is problematic for...

- proving the consistency of our logic
- designing automatic proof search

The best solution is to show cut-elimination
How to Prove Cut-elimination in General

To show that cut can be eliminated, we provide a syntactic procedure that eliminates instances cut

\[ \frac{\Pi_1}{\Gamma \rightarrow B_1} \qquad \frac{\Pi_2}{\Gamma \rightarrow B_2} \qquad \frac{\prod}{\Gamma \rightarrow B_1 \land B_2} \quad \land \mathcal{R} \quad \frac{B_1, \Gamma \rightarrow C}{B_1 \land B_2, \Gamma \rightarrow C} \quad \land \mathcal{L}_1 \]

The difficulty is then showing that this procedure always terminates
Proving Cut-elimination for $\mathcal{G}$

Tiu and Momigliano prove cut-elimination for $\text{Linc}^-$ (a subset of $\mathcal{G}$) using a notion of parametric reducibility for derivations that is based on the Girard’s proof of strong normalizability for System F.

A key lemma in this proof is:

- If $\Gamma \rightarrow C$ has a proof then $\Gamma[\theta] \rightarrow C[\theta]$ has a simpler proof

$\mathcal{G}$ expands on $\text{Linc}^-$ with $\nabla$-quantification, nominal constants, and nominal abstraction.

The following two lemmas are key:

- If $\Gamma \rightarrow C$ has a proof then $\langle \vec{\pi} \rangle.\Gamma \rightarrow \pi. C$ has the same proof
- If $\Gamma \rightarrow C$ has a proof then $\Gamma[\theta] \rightarrow C[\theta]$ has a simpler proof

Then Tiu and Momigliano’s proof extends to cut-elimination for $\mathcal{G}$.
Adequacy

How do we connect results in $G$ to results about the object system?

- We show a bijection between the expressions of the object system and their representation as terms in $G$.
- We then show an “if and only if” relationship between judgments of the object system and their encoding as atomic formulas in $G$.

Adequacy means that this kind of connection exists between an object system and its encoding in a logic.

Cut-elimination plays an essential role here since it restricts the sort of proofs we have to consider.
Using Adequacy (Example)

Suppose we have proven
\[ \forall T, V, A. \ (\text{eval } T \ V \land \text{of nil } T \ A) \supset \text{of nil } V \ A \quad (1) \]

Theorem
If \( t \downarrow v \) and \( \vdash t : a \) then \( \vdash v : a \)

Proof.

- By adequacy we know \( \rightarrow \text{eval } \Gamma t \neg \neg v \neg \neg \) and
  \( \rightarrow \text{of nil } \Gamma t \neg \neg a \neg \) have proofs in \( \mathcal{G} \)
- Using these with (1) and various rules of \( \mathcal{G} \) (particularly cut)
  we can construct a proof of \( \rightarrow \text{of nil } \Gamma v \neg \neg a \neg \)
- By adequacy we know \( \vdash v : a \)
A Specification Logic

\[ \Delta, A \models G \quad \frac{\Delta \models G[c/x]}{\Delta \models \forall x. G} \]

\[ \Delta \models G_1[\vec{t}/\vec{x}] \quad \cdots \quad \Delta \models G_m[\vec{t}/\vec{x}] \quad \frac{\Delta \models A}{\Delta \models A} \]

where \( \forall \vec{x}.(G_1 \supset \cdots \supset G_m \supset A') \in \Delta \) and \( A'[\vec{t}/\vec{x}] = A \)

Proofs in this logic reflect computations in many formal systems

\[ \forall m, n, a, b. (\text{of } m (\text{arrow } a b) \supset \text{of } n a \supset \text{of } (\text{app } m n) b) \]

\[ \forall r, a, b. ((\forall x. \text{of } x a \supset \text{of } (r x) b) \supset \text{of } (\text{fun } a r) (\text{arrow } a b)) \]
The Two-level Logic Approach to Reasoning

The specification logic sequent $\Delta, L \models G$ is encoded as the atomic formula $\text{seq } \neg L \neg G$

$$\text{seq } L \text{ (imp } A \text{ G) } \triangleq \text{seq } (A :: L) \text{ G}$$
$$\text{seq } L \text{ (all } B) \triangleq \forall x.\text{seq } L \text{ (B x)}$$
$$\text{seq } L \text{ A } \triangleq \text{member } A \text{ L}$$
$$\text{seq } L \text{ A } \triangleq \exists b.\text{prog } A \text{ b } \land \text{seq } L \text{ b}$$

Where $\text{prog}$ encodes the formulas of $\Delta$:

$$\text{prog } (\text{of } (\text{fun } A \text{ R}) \text{ (arrow } A \text{ B}))$$
$$(\text{all } \lambda x.\text{(imp } (\text{of } x \text{ A}) \text{ (of } (R x) \text{ B}))) \triangleq \top$$
Benefits of the Two-level Logic Approach to Reasoning

We can formally prove properties of \( seq \) once, and use them as lemmas about particular specifications

**Monotonicity**
\[
\forall L, K, G. (\forall X. \text{member } X L \supset \text{member } X K) \supset \text{seq } L \ G \supset \text{seq } K \ G
\]

**Instantiation**
\[
\forall L, G. \ \forall x. \ \text{seq } (L \ x) \ (G \ x) \supset \forall t. \ \text{seq } (L \ t) \ (G \ t)
\]

**Cut admissibility**
\[
\forall L, A, G. \ \text{seq } (A :: L) \ G \supset \text{seq } L \ A \supset \text{seq } L \ G
\]
Abella is an interactive, tactics-based implementation of the reasoning logic which focuses on the two-level logic approach to reasoning and hides most of the supporting machinery.

- [http://abella.cs.umn.edu](http://abella.cs.umn.edu)
- Open source and freely available
- Includes documentation, walkthroughs, and live examples
- Released in February 2008
- Hundreds of downloads so far
Successful Applications

- Determinacy, type preservation, and equivalence of various evaluation strategies
- POPLmark Challenge 1a, 2a
- Cut admissibility for a sequent calculus with quantifiers
- Properties of bisimulation in the $\pi$-calculus
- Church-Rosser property for $\lambda$-calculus
  - Contributed by Randy Pollack
- Substitution for Canonical LF
  - Contributed by Todd Wilson
  - The “triple-8” and “double-3” proofs
Statement of the Triple-8 Lemma

Theorem subst_m&r : forall Tx Ty, stype Tx -> stype Ty ->
forall Tx$ Ty$, {subt Tx$ Tx} -> {subt Ty$ Ty} ->
(forall Xs N L L' M M' M'', nabla x y, %%%% m vs. m (y x) %%%%)
   vctx Xs -> tm m Xs N -> (Xs |- subst_m Tx$ L N L') ->
   {Xs, var x |- subst_m Ty$ (y \ M x y) (L x) (M' x)} -> {Xs, var y |- subst_m Tx$ (x \ M x y) N (M' y)} ->
   exists M'', {Xs |- subst_m Tx$ M' N M''} \ {Xs |- subst_m Ty$ M' L'' M''}) \ /
(forall Xs N L L' R M' T' R'', nabla x y, %%%% rm vs. rr (y x) %%%%)
   vctx Xs -> tm m Xs N -> (Xs |- subst_m Tx$ L N L') ->
   {Xs, var x |- subst_rm Ty$ (y \ R x y) (L x) (M' x) T'} -> {Xs, var y |- subst_rr Tx$ (x \ R x y) N (R' y)} ->
   exists M'', {Xs |- subst_m Tx$ M' N M''} \ {Xs |- subst_rm Ty$ R' L' M'' T'} \ /
(forall Xs N L L' R R' M'' R'', nabla x y, %%%% rr vs. rm (y x) %%%%)
   vctx Xs -> tm m Xs N -> (Xs |- subst_m Tx$ L N L') ->
   {Xs, var x |- subst_rm Ty$ (y \ R x y) (L x) (R' x)} -> {Xs, var y |- subst_rr Tx$ (x \ R x y) N (R' y)} ->
   exists R'', {Xs |- subst_rr Tx$ R' N R''} \ {Xs |- subst_rr Ty$ R' L' R''}) \ /
(forall Xs N L L' M M' M'', nabla x y, %%%% m vs. m (x y) %%%%)
   vctx Xs -> tm m Xs N -> (Xs |- subst_m Tx$ L N L') ->
   {Xs, var x |- subst_m Tx$ (y \ M x y) (L x) (M' x)} -> {Xs, var y |- subst_m Tx$ (x \ M x y) N (M' y)} ->
   exists M'', {Xs |- subst_m Tx$ M' N M''} \ {Xs |- subst_m Tx$ M' L'' M'')} \ /
(forall Xs N L L' R M' T' R'', nabla x y, %%%% rm vs. rr (x y) %%%%)
   vctx Xs -> tm m Xs N -> (Xs |- subst_m Tx$ L N L') ->
   {Xs, var x |- subst_m Tx$ (y \ R x y) (L x) (M' x) T'} -> {Xs, var y |- subst_rr Ty$ (x \ R x y) N (R' y)} ->
   exists M'', {Xs |- subst_m Tx$ M' N M''} \ {Xs |- subst_m Tx$ R' L' M'' T'}) \ /
(forall Xs N L L' R R' M'' R'', nabla x y, %%%% rr vs. rm (x y) %%%%)
   vctx Xs -> tm m Xs N -> (Xs |- subst_m Tx$ L N L') ->
   {Xs, var x |- subst_rm Ty$ (y \ R x y) (L x) (R' x)} -> {Xs, var y |- subst_rr Tx$ (x \ R x y) N (M' y) T'} ->
   exists M'', {Xs |- subst_rm Ty$ R' N M'' T'} \ {Xs |- subst_m Tx$ M' L'' M'')} \ /
(forall Xs N L L' R R' M'', nabla x y, %%%% rr vs. rr (x y) %%%%)
   vctx Xs -> tm m Xs N -> (Xs |- subst_m Tx$ L N L') ->
   {Xs, var x |- subst_rr Tx$ (y \ R x y) (L x) (R' x)} -> {Xs, var y |- subst_rr Tx$ (x \ R x y) N (R' y)} ->
   exists R'', {Xs |- subst_rr Tx$ R' N R''} \ {Xs |- subst_rr Tx$ R' L' R''}).
Conclusions & Future Work

Summary of contributions:

- The logic $G$ and nominal abstraction
- The Abella system and its incorporation of the two-level logic approach to reasoning
- Rich examples which validate $G$, Abella, and the two-level logic approach to reasoning

Future directions:

- Alternative specification logics
- Stronger forms of definitions and (co-)inductive principles
- Improving the usability of Abella
- An integrated toolset