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

\[
\begin{align*}
a & ::= \text{int} \mid a \to a \\
t & ::= x \mid t t \mid (\text{fn } x : a \Rightarrow t)
\end{align*}
\]

Mini-ML Evaluation

\( t \Downarrow v \) means \( t \) 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 $m \mathbin{n}$
  - 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

\[
\begin{align*}
\text{int} &:: \text{type} \\
\text{arrow} &:: \text{type} \to \text{type} \to \text{type} \\
\text{app} &:: \text{term} \to \text{term} \to \text{term} \\
\text{fun} &:: \text{type} \to (\text{term} \to \text{term}) \to \text{term}
\end{align*}
\]

Example

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

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:

$$H_1, \ldots, H_n \longrightarrow C$$
Some Core Rules of the Logic

\[ \begin{align*}
\Gamma, B & \rightarrow B \quad \text{id} \\
\Gamma, \bot & \rightarrow C \quad \bot \mathcal{L} \\
\Gamma, B_i & \rightarrow C \quad \wedge \mathcal{L}_i \\
\Gamma, B_1 \wedge B_2 & \rightarrow C \quad \wedge \mathcal{L} \\
\Gamma & \rightarrow B \quad \Gamma, \Gamma & \rightarrow C \quad \text{cut} \\
\Gamma, B & \rightarrow D \quad \Gamma, D & \rightarrow C \quad \supset \mathcal{L} \\
\Gamma, B \supset D & \rightarrow C \quad \supset \mathcal{L} \\
\Gamma, B[h/x] & \rightarrow C \quad \exists \mathcal{L} \\
\Gamma, \exists x. B & \rightarrow C \quad \exists \mathcal{L} \\
\Gamma & \rightarrow B[t/x] \quad \exists \mathcal{R} \\
\Gamma & \rightarrow \exists x. B \quad \exists \mathcal{R}
\end{align*} \]
Definitions

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

Atomic formulas are interpreted as fixed-points of such definitions

$eval \ (fun \ A \ R) \ (fun \ A \ R) \triangleq \top$

$eval \ (app \ M \ N) \ V \triangleq \exists A. \exists R. \ eval \ M \ (fun \ A \ R) \land eval \ (R \ N) \ V$

We can encode this in a single definitional clause:

$eval \ T \ V \triangleq (\exists A, R. \ T = (fun \ A \ R) \land V = (fun \ A \ R)) \lor$

$(\exists M, N, A, R. \ T = (app \ M \ N) \land$

$eval \ M \ (fun \ A \ R) \land eval \ (R \ N) \ V)$
Logical Rules for Definitions

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 \( \text{eval } t \ v \)

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

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

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

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

- \( t = (\text{app } m \ n) \ldots \)
Consider a typing judgment for Mini-ML

\[
\begin{align*}
\frac{x : a \in \Gamma}{\Gamma \vdash x : a} & \quad \frac{\Gamma \vdash m : a \rightarrow b \quad \Gamma \vdash n : a}{\Gamma \vdash m \ n : b} \\
\frac{\Gamma, x : a \vdash r : b}{\Gamma \vdash (\text{fn } x : a \Rightarrow r) : a \rightarrow b} & \quad x \notin \text{dom}(\Gamma)
\end{align*}
\]

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

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

\(\text{of } \Gamma (\text{fun } A R) (\text{arrow } A B) \triangleq \exists x. \text{ of } ((x : A :: \Gamma) (R x) B)\)
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

\[
\begin{align*}
B[a/x], \Gamma & \rightarrow C & \nabla\mathcal{L} \\
\nabla x. B, \Gamma & \rightarrow C & \nabla\mathcal{L} \\
\Gamma & \rightarrow B[a/x] & \nabla\mathcal{R} \\
\Gamma & \rightarrow \nabla x. B & \nabla\mathcal{R}
\end{align*}
\]

$a$ 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

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

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

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

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

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

cntx \ \Gamma \ 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*:

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

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

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

\[ \Gamma \rightarrow t = t = \mathcal{R} \]

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

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

\[ = \mathcal{L} \]

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

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

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

\[ \triangleright \mathcal{L} \]

\[ \cdot [\cdot] \] is a generalized notion of substitution which respects the scope of nominal constants
Summary of the Logic $\mathcal{G}$

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

\[ \Gamma \rightarrow B, B, \Gamma \rightarrow C \]

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

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 \textit{cut} can be eliminated, we provide a syntactic procedure that eliminates instances \textit{cut}

\[
\frac{\Pi_1}{\Gamma \rightarrow B_1} \quad \frac{\Pi_2}{\Gamma \rightarrow B_2} \quad \frac{\Pi}{B_1, \Gamma \rightarrow C}
\]
\[
\frac{\land \mathcal{R}}{\Gamma \rightarrow B_1 \land B_2}
\]
\[
\frac{\Gamma \rightarrow B_1 \land B_2, \Gamma \rightarrow C}{\Gamma \rightarrow C}
\]

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

**Theorem**

If \( t \Downarrow v \) and \( \vdash t : a \) then \( \vdash v : a \)

**Proof.**

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

\[
\Delta, A \vdash G \\
\Delta \vdash A \supset G
\]

\[
\Delta \vdash G[c/x] \\
\Delta \vdash \forall x. G
\]

\[
\Delta \vdash G_1[t/x] \quad \cdots \quad \Delta \vdash G_m[t/x]
\]

\[
\Delta \vdash A
\]

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

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 $\Delta, L \vdash G$ is encoded as the atomic formula $\text{seq } \neg L \neg \neg G \neg$

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

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

\[
\text{prog } (\text{of } (\text{fun } A \ R) \ (\text{arrow } A \ B)) \\
(\text{all } \lambda x.\text{(imp } (\text{of } x \ A) \ (\text{of } (R \ x) \ B))) \ \triangleq \ \top
\]
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**
\[ \forall L, G. \ \nabla x. \ \text{seq } (L \ x) \ (G \ x) \supset \forall t. \ \text{seq } (L \ t) \ (G \ t) \]

**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_rm 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' 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_rr Ty$ (y \ R x y) (L x) (R' x)} -> {Xs, var y |- subst_rm Tx$ (x \ R x y) N (R' y)} ->

exists M', {Xs |- subst_m Tx$ M' N M''} \ {Xs |- subst_rr Ty$ R' L' M''} \ /

(forall Xs N L L' R R' R'', 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 Ty$ (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 Ty$ L N L') ->

{Xs, var x |- subst_rm Tx$ (y \ R x y) (L x) (M' x) T'} -> {Xs, var y |- subst_rm Ty$ (x \ R x y) N (R' y)} ->

exists M', {Xs |- subst_m Tx$ M' N M''} \ {Xs |- subst_rm Tx$ R' L' M'' T'}) \ /

(forall Xs N L L' R R' R'', nabla x y, %%% rr vs. rm (x y) %%%

vctx Xs -> tm m Xs N -> (Xs |- subst_m Ty$ L N L') ->

{Xs, var x |- subst_rm Tx$ (y \ R x y) (L x) (R' x)} -> {Xs, var y |- subst_rm Tx$ (x \ R x y) N (R' y)} ->

exists M', {Xs |- subst_rm Tx$ R' N M''} \ {Xs |- subst_m Tx$ R' L' M''}) \ /

(forall Xs N L L' R R' R'', nabla x y, %%% rr vs. rr (x y) %%%

vctx Xs -> tm m Xs N -> (Xs |- subst_m Ty$ L N L') ->

{Xs, var x |- subst_rr Tx$ (y \ R x y) (L x) (R' x)} -> {Xs, var y |- subst_rm Ty$ (x \ R x y) N (R' y)} ->

exists R~, {Xs |- subst_rr Tx$ R' N R''} \ {Xs |- subst_rm Ty$ 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