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 λ-calculus in their term structure so we can represent binding
- They contain logical devices for analyzing such structure

Contributions

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

Example: Mini-ML

Mini-ML Syntax

$$a ::= \texttt{int} \mid a \to a$$

$$t ::= \texttt{x} \mid t\ t \mid (\texttt{fn x:}a\ \texttt{=>}\ t)$$

Mini-ML Evaluation

$t \Downarrow v$ means t evaluates to v

$$\frac{}{(\texttt{fn x:}a\ \texttt{=>}\ r) \Downarrow (\texttt{fn x:}a\ \texttt{=>}\ r)}$$

$$\frac{m \Downarrow (\texttt{fn x:}a\ \texttt{=>}\ r) \qquad r[\texttt{x} := n] \Downarrow v}{m\ n \Downarrow v}$$

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 (`fn x:`a `=>` r)
  Must be that w is (`fn x:`a `=>` r)
- t is $m\ n$
  - Must have $m \Downarrow$ (`fn x:`a `=>` r) and $r[\mathtt{x} := n] \Downarrow v$
  - Must have $m \Downarrow$ (`fn x:`b `=>` s) and $s[\mathtt{x} := n] \Downarrow w$
  - By induction $r = s$, and thus by induction $v = w$ $\square$

A Higher-order Abstract Syntax Representation

Object level binding can be represented with meta-level abstraction

Constants for Mini-ML

$$\mathit{int} :: \mathit{type}$$

$$\mathit{arrow} :: \mathit{type} \to \mathit{type} \to \mathit{type}$$

$$\mathit{app} :: \mathit{term} \to \mathit{term} \to \mathit{term}$$

$$\mathit{fun} :: \mathit{type} \to (\mathit{term} \to \mathit{term}) \to \mathit{term}$$

Example

```
fn x : int => fn y : int => x
```

$$\mathit{fun}\ \mathit{int}\ (\lambda x.\ \mathit{fun}\ \mathit{int}\ (\lambda y.\ x))$$

Binding issues are now treated in the meta-level

Basic Structure for Reasoning

- Formulas over expressions from the simply-typed λ-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

$$\frac{}{\Gamma, B \longrightarrow B}\ id \qquad \frac{\Gamma \longrightarrow B \quad B, \Gamma \longrightarrow C}{\Gamma \longrightarrow C}\ cut$$

$$\frac{}{\Gamma, \bot \longrightarrow C}\ \bot\mathcal{L} \qquad \frac{}{\Gamma \longrightarrow \top}\ \top\mathcal{R}$$

$$\frac{\Gamma, B_i \longrightarrow C}{\Gamma, B_1 \wedge B_2 \longrightarrow C}\ \wedge\mathcal{L}_i \qquad \frac{\Gamma \longrightarrow B \quad \Gamma \longrightarrow C}{\Gamma \longrightarrow B \wedge C}\ \wedge\mathcal{R}$$

$$\frac{\Gamma \longrightarrow B \quad \Gamma, D \longrightarrow C}{\Gamma, B \supset D \longrightarrow C}\ \supset\mathcal{L} \qquad \frac{\Gamma, B \longrightarrow C}{\Gamma \longrightarrow B \supset C}\ \supset\mathcal{R}$$

$$\frac{\Gamma, B[h/x] \longrightarrow C}{\Gamma, \exists x.B \longrightarrow C}\ \exists\mathcal{L} \qquad \frac{\Gamma \longrightarrow B[t/x]}{\Gamma \longrightarrow \exists x.B}\ \exists\mathcal{R}$$

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

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

We can encode this in a single definitional clause:

$$\begin{aligned} eval\ T\ V \triangleq\ & (\exists A, R.\ T = (fun\ A\ R) \wedge V = (fun\ A\ R)) \vee \\ & (\exists M, N, A, R.\ T = (app\ M\ N) \wedge \\ & \qquad eval\ M\ (fun\ A\ R) \wedge eval\ (R\ N)\ V) \end{aligned}$$

Logical Rules for Definitions

Let p be defined by

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

$$\frac{\Gamma, B\ p\ \vec{t} \longrightarrow C}{\Gamma, p\ \vec{t} \longrightarrow C}\ \mathit{def}\mathcal{L} \qquad \frac{\Gamma \longrightarrow B\ p\ \vec{t}}{\Gamma \longrightarrow p\ \vec{t}}\ \mathit{def}\mathcal{R}$$

We also have rules for induction and co-induction for appropriate definitions

Formally Proving Determinacy of Evaluation

Theorem

$\forall t, v, w.\ (eval\ \ t\ \ v \wedge eval\ \ t\ \ w) \supset v = w$

Proof.

Apply rules for $\forall$, $\wedge$, and $\supset$

$\boxed{eval\ \ t\ \ v, eval\ \ t\ \ w \longrightarrow v = w}$

Case analysis on $eval\ \ t\ \ v$

- $t = v = (fun\ \ a\ \ r)$

  $\boxed{eval\ \ (fun\ \ a\ \ r)\ \ w \longrightarrow (fun\ \ a\ \ r) = w}$

  Case analysis on $eval\ \ (fun\ \ a\ \ r)\ \ w$

  $\boxed{\longrightarrow (fun\ \ a\ \ r) = (fun\ \ a\ \ r)}$

- $t = (app\ \ m\ \ n)\ \ldots$

$\square$

Dynamic Aspects of Binding

Consider a typing judgment for Mini-ML

$$\frac{\mathtt{x} : a \in \Gamma}{\Gamma \vdash \mathtt{x} : a} \qquad\qquad \frac{\Gamma \vdash m : a \to b \qquad \Gamma \vdash n : a}{\Gamma \vdash m\ n : b}$$

$$\frac{\Gamma, \mathtt{x} : a \vdash r : b}{\Gamma \vdash \mathtt{(fn\ \ x:}a\mathtt{\ \ =>\ \ }r\mathtt{)} : a \to b}\ \mathtt{x} \notin dom(\Gamma)$$

$$of\ \Gamma\ X\ A \triangleq member\ (X : A)\ \Gamma$$

$$of\ \Gamma\ (app\ M\ N)\ B \triangleq \exists A.\ of\ \Gamma\ M\ (arrow\ A\ B) \wedge of\ \Gamma\ N\ A$$

$$of\ \Gamma\ (fun\ A\ R)\ (arrow\ A\ B) \triangleq \nabla x.\ of\ ((x : A) :: \Gamma)\ (R\ x)\ B$$

Some Properties of the ∇ Quantifier

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

We have the following structural properties for ∇:

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

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

If we allow ∇ quantification at a type, then we assume there are infinitely many fresh names at that type

Logical Rules for the ∇ Quantifier

$$\frac{B[a/x], \Gamma \longrightarrow C}{\nabla x.B, \Gamma \longrightarrow C} \nabla\mathcal{L} \qquad\qquad \frac{\Gamma \longrightarrow B[a/x]}{\Gamma \longrightarrow \nabla x.B} \nabla\mathcal{R}$$

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

$$\frac{}{\Gamma, B \longrightarrow B'} \; id, \text{ if } B = \pi.B'$$

Typing Example with ∇

$$of\ \Gamma\ X\ A \triangleq member\ (X : A)\ \Gamma$$
$$of\ \Gamma\ (app\ M\ N)\ B \triangleq \exists A.\ of\ \Gamma\ M\ (arrow\ A\ B) \wedge of\ \Gamma\ N\ A$$
$$of\ \Gamma\ (fun\ A\ R)\ (arrow\ A\ B) \triangleq \nabla x.\ of\ ((x : A) :: \Gamma)\ (R\ x)\ B$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\longrightarrow member\ (c : int)\ ((d : int) :: (c : int) :: nil)}}{\longrightarrow of\ ((d : int) :: (c : int) :: nil)\ c\ int}}{\longrightarrow \nabla x. of\ ((x : int) :: (c : int) :: nil)\ c\ int}}{\longrightarrow of\ ((c : int) :: nil)\ (fun\ int\ (\lambda y.\ c))\ (arrow\ int\ int)}}{\dfrac{\longrightarrow \nabla x.\ of\ ((x : int) :: nil)\ (fun\ int\ (\lambda y.\ x))\ (arrow\ int\ int)}{\longrightarrow of\ nil\ (fun\ int\ (\lambda x.\ fun\ int\ (\lambda y.\ x)))\ (arrow\ int\ (arrow\ int\ int))}}$$

Reasoning about Type Uniqueness

$$\forall t, a, b.\ (of\ \ nil\ \ t\ \ a \wedge of\ \ nil\ \ t\ \ b) \supset a = b$$

$$\forall \Gamma, t, a, b.\ (of\ \ \Gamma\ \ t\ \ a \wedge of\ \ \Gamma\ \ t\ \ b) \supset a = b$$

$$\forall \Gamma, t, a, b.\ (cntx\ \Gamma \wedge of\ \ \Gamma\ \ t\ \ a \wedge of\ \ \Gamma\ \ t\ \ b) \supset a = b$$

$cntx\ \Gamma$ should enforce

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

Definitions can serve to capture such meta-level properties

$$cntx\ \ nil \triangleq \top$$

$$cntx\ \ ((X : A) :: L) \triangleq \text{“}X \text{ atomic and not occurring in } L\text{”} \wedge cntx\ \ L$$

Analyzing Occurrences of Nominal Constants

We introduce the device of *nominal abstraction*:

$$(\lambda x_1 \cdots \lambda x_n.s) \trianglerighteq t$$

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"

  $(\lambda z.z) \trianglerighteq X$

- "X is atomic and does not occur in L"

  $(\lambda z.\mathit{fresh}\ z\ L) \trianglerighteq \mathit{fresh}\ X\ L$

Nominal Abstraction as a Modular Extension of Equality

$$\frac{}{\Gamma \longrightarrow t = t} =\mathcal{R}$$

$$\frac{\{\Gamma[\theta] \longrightarrow C[\theta] \mid \text{all } \theta \text{ such that } (s = t)[\theta]\}}{s = t, \Gamma \longrightarrow C} =\mathcal{L}$$

$$\frac{}{\Gamma \longrightarrow s \trianglerighteq t} \trianglerighteq\mathcal{R}, \text{ if } s \trianglerighteq t \text{ holds}$$

$$\frac{\{\Gamma[\![\theta]\!] \longrightarrow C[\![\theta]\!] \mid \text{all } \theta \text{ such that } (s \trianglerighteq t)[\![\theta]\!]\}}{s \trianglerighteq t, \Gamma \longrightarrow C} \trianglerighteq\mathcal{L}$$

$\cdot[\![\cdot]\!]$ 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 λ-terms for representation
- atomic formulas for encoding judgments
- fixed-point definitions for encoding rules
- induction (and co-induction) over appropriate fixed-point definitions
- ∇ quantifier for introducing fresh names
- nominal abstraction for analyzing occurrences of names

Cut and Cut-elimination

$$\frac{\Gamma \longrightarrow B \quad B, \Gamma \longrightarrow C}{\Gamma \longrightarrow C}\ \mathit{cut}$$

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*

$$
\dfrac{\dfrac{\stackrel{\Pi_1}{\Gamma \longrightarrow B_1} \quad \stackrel{\Pi_2}{\Gamma \longrightarrow B_2}}{\Gamma \longrightarrow B_1 \wedge B_2} \wedge\mathcal{R} \qquad \dfrac{\stackrel{\Pi}{B_1, \Gamma \longrightarrow C}}{B_1 \wedge B_2, \Gamma \longrightarrow C} \wedge\mathcal{L}_1}{\Gamma \longrightarrow C} \; cut
$$

$$
\dfrac{\stackrel{\Pi_1}{\Gamma \longrightarrow B_1} \quad \stackrel{\Pi}{B_1, \Gamma \longrightarrow C}}{\Gamma \longrightarrow C} \; cut
$$

The difficulty is then showing that this procedure always terminates

Proving Cut-elimination for $\mathcal{G}$

Tiu and Momigliano prove cut-elimination for 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 $\Gamma \longrightarrow C$ has a proof then $\Gamma[\theta] \longrightarrow C[\theta]$ has a simpler proof

---

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

The following two lemmas are key:

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

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

Adequacy

How do we connect results in $\mathcal{G}$ to results about the object system?

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

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

$$\forall T, V, A.\ (eval\ \ T\ \ V \wedge of\ \ nil\ \ T\ \ A) \supset of\ \ nil\ \ V\ \ A \tag{1}$$

Theorem

If $t \Downarrow v$ *and* $\vdash t : a$ *then* $\vdash v : a$

Proof.

- By adequacy we know $\longrightarrow eval\ \ulcorner t \urcorner\ \ulcorner v \urcorner$ and $\longrightarrow of\ \ nil\ \ulcorner t \urcorner\ \ulcorner a \urcorner$ have proofs in $\mathcal{G}$
- Using these with (1) and various rules of $\mathcal{G}$ (particularly *cut*) we can construct a proof of $\longrightarrow of\ \ nil\ \ulcorner v \urcorner\ \ulcorner a \urcorner$
- By adequacy we know $\vdash v : a$ $\square$

A Specification Logic

$$\frac{\Delta, A \Vdash G}{\Delta \Vdash A \supset G} \qquad \frac{\Delta \Vdash G[c/x]}{\Delta \Vdash \forall x.G}$$

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

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

$$\forall m, n, a, b.(of\ \ m\ (arrow\ \ a\ \ b) \supset of\ \ n\ \ a \supset of\ \ (app\ \ m\ \ n)\ \ b)$$

$$\forall r, a, b.((\forall x.of\ \ x\ \ a \supset of\ \ (r\ \ x)\ \ b) \supset of\ \ (fun\ \ a\ \ r)\ (arrow\ \ a\ \ b))$$

The Two-level Logic Approach to Reasoning

The specification logic sequent $\Delta, L \Vdash G$ is encoded as the atomic formula $seq \ulcorner L \urcorner \ulcorner G \urcorner$

$$
\begin{aligned}
seq\ L\ (imp\ A\ G) &\triangleq seq\ (A :: L)\ G \\
seq\ L\ (all\ B) &\triangleq \nabla x. seq\ L\ (B\ x) \\
seq\ L\ A &\triangleq member\ A\ L \\
seq\ L\ A &\triangleq \exists b. prog\ A\ b \wedge seq\ L\ b
\end{aligned}
$$

Where *prog* encodes the formulas of Δ:

$$
\begin{aligned}
&prog\ (of\ (fun\ A\ R)\ (arrow\ A\ B)) \\
&\qquad (all\ \lambda x.(imp\ (of\ x\ A)\ (of\ (R\ x)\ B))) \triangleq \top
\end{aligned}
$$

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

$\forall L, K, G.\ (\forall X. member\ X\ L \supset member\ X\ K) \supset seq\ L\ G \supset seq\ K\ G$

Instantiation

$\forall L, G.\ \nabla x.\ seq\ (L\ x)\ (G\ x) \supset \forall t.\ seq\ (L\ t)\ (G\ t)$

Cut admissibility

$\forall L, A, G.\ seq\ (A :: L)\ G \supset seq\ L\ A \supset seq\ L\ G$

Implementation

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`
- 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 π-calculus
- Church-Rosser property for λ-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' T', 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 (M' y) T'} ->
      exists M~, {Xs |- subst_rm Tx$ R` N M~ T'} /\ {Xs |- subst_m Ty$ M' L' M~})    /\
    (forall Xs N L L' R R` R', nabla x y,        %%%% rr vs. rr (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_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 Ty$ 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 Ty$ 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_rr Ty$ (x\ R x y) N (R' y)} ->
      exists M~, {Xs |- subst_m Ty$ M` N M~} /\ {Xs |- subst_rm Tx$ R' L' M~ T`})    /\
    (forall Xs N L L' R R` M' T', 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_rr Tx$ (y\ R x y) (L x) (R` x)} -> {Xs, var y |- subst_rm Ty$ (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` 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_rr Ty$ (x\ R x y) N (R' y)} ->
      exists R~, {Xs |- subst_rr Ty$ R` N R~} /\ {Xs |- subst_rr Tx$ R' L' R~}).
```

Conclusions & Future Work

Summary of contributions:

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