The Abella Interactive Theorem Prover (System Description)

Andrew Gacek

Department of Computer Science and Engineering
University of Minnesota

IJCAR ’08
August 12, 2008

Characteristics of the Abella System

Abella is a theorem proving system that

- is geared towards reasoning about formal systems specified via structural rules
- uses higher-order abstract syntax in a fundamental way
- based on a two-level logic approach
    - (executable) specification logic for describing formal systems
    - meta-logic for reasoning about specification logic descriptions
- exploits key specification logic properties as lemmas in the meta-logic

Formal Systems Specified via Structural Rules

$$\frac{x : a \in \Gamma}{\Gamma \vdash x : a}$$

$$\frac{\Gamma \vdash t_1 : a \rightarrow b \quad \Gamma \vdash t_2 : a}{\Gamma \vdash (t_1\ t_2) : b}$$

$$\frac{\Gamma, x : a \vdash t : b}{\Gamma \vdash (\lambda x{:}a.\ t) : a \rightarrow b}\ x \notin dom(\Gamma)$$

Formal Systems Specified via Structural Rules

$$\frac{x : a \in \Gamma}{\Gamma \vdash x : a}$$

$$\frac{\Gamma \vdash t_1 : a \to b \quad \Gamma \vdash t_2 : a}{\Gamma \vdash (t_1\ t_2) : b}$$

$$\frac{\Gamma, x : a \vdash t : b}{\Gamma \vdash (\lambda x{:}a.\ t) : a \to b}\ x \notin dom(\Gamma)$$

Type uniqueness

If $\Gamma \vdash t : a$ and $\Gamma \vdash t : b$ then $a = b$

Type preservation

If $\Gamma, x : a \vdash t_1 : b$ and $\Gamma \vdash t_2 : a$ then $\Gamma \vdash t_1[x := t_2] : b$

Higher-order Abstract Syntax

Higher-order abstract syntax uses meta-level abstraction to represent object-level binding

$$\begin{aligned} \overline{x} &\longrightarrow (\mathit{var}\ x) \\ \overline{(t_1\ t_2)} &\longrightarrow (\mathit{app}\ \overline{t_1}\ \overline{t_2}) \\ \overline{\lambda x : a.\ t} &\longrightarrow (\mathit{abs}\ a\ (\lambda x.\ \overline{t})) \end{aligned}$$

Higher-order Abstract Syntax

Higher-order abstract syntax uses meta-level abstraction to represent object-level binding

$$\begin{aligned} \overline{x} &\longrightarrow (var\ x) \\ \overline{(t_1\ \ t_2)} &\longrightarrow (app\ \overline{t_1}\ \overline{t_2}) \\ \overline{\lambda x{:}a.\ t} &\longrightarrow (abs\ a\ (\lambda x.\ \overline{t})) \end{aligned}$$

Benefits

- α-equivalence completely handled by the meta-level

$$(abs\ a\ (\lambda x.t)) = (abs\ a\ (\lambda y.t[x := y]))$$

Higher-order Abstract Syntax

Higher-order abstract syntax uses meta-level abstraction to represent object-level binding

$$\begin{aligned} \overline{x} &\longrightarrow (var\ x) \\ \overline{(t_1\ t_2)} &\longrightarrow (app\ \overline{t_1}\ \overline{t_2}) \\ \overline{\lambda x : a.\ t} &\longrightarrow (abs\ a\ (\lambda x.\ \overline{t})) \end{aligned}$$

Benefits

- α-equivalence completely handled by the meta-level

  $$(abs\ a\ (\lambda x.t)) = (abs\ a\ (\lambda y.t[x := y]))$$

- capture-avoiding substitution realized via β-reduction

  $$(app\ (abs\ a\ t_1)\ t_2) \Longrightarrow (t_1\ t_2)$$

Two-level Logic Approach

Advocated by McDowell, Miller, and Tiu

Structure

- specification logic for describing formal systems
- meta-logic for reasoning about specification logic descriptions

Two-level Logic Approach

Advocated by McDowell, Miller, and Tiu

Structure

- specification logic for describing formal systems
- meta-logic for reasoning about specification logic descriptions

Some of the benefits

- clean separation between specification and reasoning so features of each logic can be tailored to needs (*e.g.*, executable vs rich)
- allows for different specification logics

Design of the Specification Logic

The specification logic should

- support rule-based descriptions

Design of the Specification Logic

The specification logic should

- support rule-based descriptions

- provide support for higher-order abstract syntax

Design of the Specification Logic

The specification logic should

- support rule-based descriptions

- provide support for higher-order abstract syntax

  permit explicit representations of binding

Design of the Specification Logic

The specification logic should

- support rule-based descriptions

- provide support for higher-order abstract syntax

  permit explicit representations of binding

  have mechanisms for logically analyzing binding

Design of the Specification Logic

The specification logic should

- support rule-based descriptions

- provide support for higher-order abstract syntax

  permit explicit representations of binding

  have mechanisms for logically analyzing binding

  contain declarative means for recursion over binding structure

Design of the Specification Logic

The specification logic should

- support rule-based descriptions
- provide support for higher-order abstract syntax

  permit explicit representations of binding

  have mechanisms for logically analyzing binding

  contain declarative means for recursion over binding structure
- be executable

Design of the Specification Logic

The specification logic should

- support rule-based descriptions
- provide support for higher-order abstract syntax

  permit explicit representations of binding

  have mechanisms for logically analyzing binding

  contain declarative means for recursion over binding structure
- be executable

Abella uses second-order hereditary Harrop formulas

Design of the Specification Logic

The specification logic should

- support rule-based descriptions

  Horn clause like descriptions of relations

- provide support for higher-order abstract syntax

  permit explicit representations of binding

  have mechanisms for logically analyzing binding

  contain declarative means for recursion over binding structure

- be executable

Abella uses second-order hereditary Harrop formulas

Design of the Specification Logic

The specification logic should

- support rule-based descriptions

  Horn clause like descriptions of relations

- provide support for higher-order abstract syntax

  permit explicit representations of binding

  lambda terms as data structures

  have mechanisms for logically analyzing binding

  contain declarative means for recursion over binding structure

- be executable

Abella uses second-order hereditary Harrop formulas

Design of the Specification Logic

The specification logic should

- support rule-based descriptions

  Horn clause like descriptions of relations

- provide support for higher-order abstract syntax

  permit explicit representations of binding

  lambda terms as data structures

  have mechanisms for logically analyzing binding

  unification over lambda conversion rules

  contain declarative means for recursion over binding structure

- be executable

Abella uses second-order hereditary Harrop formulas

Design of the Specification Logic

The specification logic should

- support rule-based descriptions

  Horn clause like descriptions of relations

- provide support for higher-order abstract syntax

  permit explicit representations of binding

  lambda terms as data structures

  have mechanisms for logically analyzing binding

  unification over lambda conversion rules

  contain declarative means for recursion over binding structure

  generic goals to move object level binding to the meta level

- be executable

Abella uses second-order hereditary Harrop formulas

Design of the Specification Logic

The specification logic should

- support rule-based descriptions

  Horn clause like descriptions of relations

- provide support for higher-order abstract syntax

  permit explicit representations of binding

  lambda terms as data structures

  have mechanisms for logically analyzing binding

  unification over lambda conversion rules

  contain declarative means for recursion over binding structure

  generic goals to move object level binding to the meta level

- be executable

  subset of λProlog which has an efficient implementation

  `http://teyjus.cs.umn.edu`

Abella uses second-order hereditary Harrop formulas

Design of the Meta-logic

The meta-logic should

- be able to encode the specification logic

Design of the Meta-logic

The meta-logic should

- be able to encode the specification logic

- allow descriptions of properties of specifications

Design of the Meta-logic

The meta-logic should

- be able to encode the specification logic
- allow descriptions of properties of specifications
- provide mechanisms for reasoning about the specification logic treatment of binding constructs

Design of the Meta-logic

The meta-logic should

- be able to encode the specification logic
- allow descriptions of properties of specifications
- provide mechanisms for reasoning about the specification logic treatment of binding constructs
- support inductive arguments over the structure of specifications

Design of the Meta-logic

The meta-logic should

- be able to encode the specification logic
- allow descriptions of properties of specifications
- provide mechanisms for reasoning about the specification logic treatment of binding constructs
- support inductive arguments over the structure of specifications

Abella uses the logic $\mathcal{G}$ [LICS08] as a meta-logic

Design of the Meta-logic

The meta-logic should

- be able to encode the specification logic

  atomic judgments unraveled by definitions

- allow descriptions of properties of specifications
- provide mechanisms for reasoning about the specification logic treatment of binding constructs
- support inductive arguments over the structure of specifications

Abella uses the logic $\mathcal{G}$ [LICS08] as a meta-logic

Design of the Meta-logic

The meta-logic should

- be able to encode the specification logic
  atomic judgments unraveled by definitions
- allow descriptions of properties of specifications
  atomic judgments can be combined using meta-logic connectives
- provide mechanisms for reasoning about the specification logic treatment of binding constructs
- support inductive arguments over the structure of specifications

Abella uses the logic $\mathcal{G}$ [LICS08] as a meta-logic

Design of the Meta-logic

The meta-logic should

- be able to encode the specification logic

  atomic judgments unraveled by definitions

- allow descriptions of properties of specifications

  atomic judgments can be combined using meta-logic connectives

- provide mechanisms for reasoning about the specification logic treatment of binding constructs

  generic judgments to represent generic goals

- support inductive arguments over the structure of specifications

Abella uses the logic $\mathcal{G}$ [LICS08] as a meta-logic

Design of the Meta-logic

The meta-logic should

- be able to encode the specification logic

  atomic judgments unraveled by definitions

- allow descriptions of properties of specifications

  atomic judgments can be combined using meta-logic connectives

- provide mechanisms for reasoning about the specification logic treatment of binding constructs

  generic judgments to represent generic goals

- support inductive arguments over the structure of specifications

  natural number induction

Abella uses the logic $\mathcal{G}$ [LICS08] as a meta-logic

Exploiting Specification Logic Properties in Reasoning

Specification logic properties are encoded via lemmas in Abella

- The *context* lemma allows weakening, permutation, and contraction of the specification logic context

Exploiting Specification Logic Properties in Reasoning

Specification logic properties are encoded via lemmas in Abella

- The *context* lemma allows weakening, permutation, and contraction of the specification logic context

  if $pv(\Gamma_1, C)$ and $\Gamma_1 \subseteq \Gamma_2$ then $pv(\Gamma_2, C)$

Exploiting Specification Logic Properties in Reasoning

Specification logic properties are encoded via lemmas in Abella

- The *context* lemma allows weakening, permutation, and contraction of the specification logic context

  if $pv(\Gamma_1, C)$ and $\Gamma_1 \subseteq \Gamma_2$ then $pv(\Gamma_2, C)$

  if $\Gamma_1 \vdash t : a$ and $permute(\Gamma_1, \Gamma_2)$ then $\Gamma_2 \vdash t : a$

Exploiting Specification Logic Properties in Reasoning

Specification logic properties are encoded via lemmas in Abella

- The *context* lemma allows weakening, permutation, and contraction of the specification logic context

  if $pv(\Gamma_1, C)$ and $\Gamma_1 \subseteq \Gamma_2$ then $pv(\Gamma_2, C)$

  if $\Gamma_1 \vdash t : a$ and $permute(\Gamma_1, \Gamma_2)$ then $\Gamma_2 \vdash t : a$

- The *instantiation* lemma instantiates generic variables in the specification logic

Exploiting Specification Logic Properties in Reasoning

Specification logic properties are encoded via lemmas in Abella

- The *context* lemma allows weakening, permutation, and contraction of the specification logic context

  if $pv(\Gamma_1, C)$ and $\Gamma_1 \subseteq \Gamma_2$ then $pv(\Gamma_2, C)$

  if $\Gamma_1 \vdash t : a$ and $permute(\Gamma_1, \Gamma_2)$ then $\Gamma_2 \vdash t : a$

- The *instantiation* lemma instantiates generic variables in the specification logic
- The *cut* lemma relieves a specification logic hypothesis with a proof of such a hypothesis

Exploiting Specification Logic Properties in Reasoning

Specification logic properties are encoded via lemmas in Abella

- The *context* lemma allows weakening, permutation, and contraction of the specification logic context

  if $pv(\Gamma_1, C)$ and $\Gamma_1 \subseteq \Gamma_2$ then $pv(\Gamma_2, C)$

  if $\Gamma_1 \vdash t : a$ and $permute(\Gamma_1, \Gamma_2)$ then $\Gamma_2 \vdash t : a$

- The *instantiation* lemma instantiates generic variables in the specification logic
- The *cut* lemma relieves a specification logic hypothesis with a proof of such a hypothesis

  if $pv(\Gamma, \forall x.H \Rightarrow C)$ and $pv(\Gamma, H[x := v])$ then $pv(\Gamma, C[x := v])$

Exploiting Specification Logic Properties in Reasoning

Specification logic properties are encoded via lemmas in Abella

- The *context* lemma allows weakening, permutation, and contraction of the specification logic context

  if $pv(\Gamma_1, C)$ and $\Gamma_1 \subseteq \Gamma_2$ then $pv(\Gamma_2, C)$

  if $\Gamma_1 \vdash t : a$ and $permute(\Gamma_1, \Gamma_2)$ then $\Gamma_2 \vdash t : a$

- The *instantiation* lemma instantiates generic variables in the specification logic
- The *cut* lemma relieves a specification logic hypothesis with a proof of such a hypothesis

  if $pv(\Gamma, \forall x.H \Rightarrow C)$ and $pv(\Gamma, H[x := v])$ then $pv(\Gamma, C[x := v])$

  if $\Gamma, x : a \vdash t_1 : b$ and $\Gamma \vdash t_2 : a$ then $\Gamma \vdash t_1[x := t_2] : b$

Exploiting Specification Logic Properties in Reasoning

Specification logic properties are encoded via lemmas in Abella

- The *context* lemma allows weakening, permutation, and contraction of the specification logic context

  if $pv(\Gamma_1, C)$ and $\Gamma_1 \subseteq \Gamma_2$ then $pv(\Gamma_2, C)$

  if $\Gamma_1 \vdash t : a$ and $permute(\Gamma_1, \Gamma_2)$ then $\Gamma_2 \vdash t : a$

- The *instantiation* lemma instantiates generic variables in the specification logic
- The *cut* lemma relieves a specification logic hypothesis with a proof of such a hypothesis

  if $pv(\Gamma, \forall x.H \Rightarrow C)$ and $pv(\Gamma, H[x := v])$ then $pv(\Gamma, C[x := v])$

  if $\Gamma, x : a \vdash t_1 : b$ and $\Gamma \vdash t_2 : a$ then $\Gamma \vdash t_1[x := t_2] : b$

The framework accommodates additional lemmas like these

Successful Applications of Abella

- Determinacy and type preservation of various evaluation strategies
- POPLmark Challenge 1a, 2a
- Cut admissibility for a sequent calculus
- Church-Rosser property for λ-calculus
- Tait-style weak normalizability proof [LFMTP08]

The code for all these examples is on the Abella website

Conclusion

The Abella website has tutorials, examples, downloads, papers, and documentation

`http://abella.cs.umn.edu/`

Ask me for a demo!