Computation-as-deduction in Abella: work in progress

Kaustuv Chaudhuri, Ulysse Gérard and Dale Miller
LFMTP, July 7, 2018

Inria Saclay
Palaiseau France
Abella is an interactive theorem prover in which relations, and not functions, are defined by (co)induction.

It has rather limited forms of automation.

Recent work on focused proof systems for the logic underlying Abella allows us to propose various extensions.
Notions of $G$-logic and focusing
The $\mathcal{G}$-logic in Abella [Baelde et al., 2014]

An extension of intuitionistic first-order logic with

- Higher-order $\lambda$-terms with $\alpha\beta\eta$-equivalence
- Inductive and coinductive fixed point definitions
- Nominals, nominal abstraction and generic ($\nabla$) quantification.
$\mathcal{G}$'s terms are well-typed terms of Church's simple theory of types, a given type signature declares:

- basic types (keyword \texttt{Kind})
- constants which are constructors for these basic types (\texttt{Type}).

\begin{verbatim}
Kind bool  type.
Type tt, ff bool.

Kind nat  type.
Type z nat.
Type s nat \rightarrow nat.
\end{verbatim}
Two ways to build atomic formulas:

- With **Type** declarations of target type **prop**
- Using inductively or coinductively defined fixed points:

Define `is_nat : nat → prop` by

\[
\begin{align*}
\text{is\_nat } z; \\
\text{is\_nat } (s\ X) & := \text{is\_nat } X.
\end{align*}
\]
Define plus : nat → nat → nat → prop by
    plus z X X ;
    plus (s X) Y (s Z) := plus X Y Z.

Theorem plus_z2 : forall X, is_nat X → plus X z X.

Proved by induction on the first antecedent of the chain of implications : is_nat X.
Organize search for proofs in an alternation of two phases:

- **Invertible** (asynchronous): invertible rules, can be applied in any order (intros, split and case tactics)
- **Synchronous**: other rules, require choices from the user to progress (unfold, left/right, witness, instantiating variables or inventing and using lemmas)
Invertible phases are functionally determined by their conclusion. A definition can be fully discharged in one invertible phase if:

- It appears as an hypothesis and is made of positive connectives \((=, \land, \lor, \text{false}, \text{and exists})\)
- Or it appears as a goal and is made of negative connectives \((\land, \text{true}, \rightarrow, \text{and forall})\)
1st proposal: Compute and suspend
The `compute` tactic performs unfolding and subsequent asynchronous steps for assumptions involving fully positive definition predicates.

\[ \text{forall } X, \text{ plus } (s \; z) \; (s \; z) \; X \rightarrow X = s \; (s \; z) \]

---

**Proof completed.**
The \texttt{compute} tactic can lead to multiple subgoals: predicates.

\texttt{forall} X Y, plus X Y (s (s z)) $\rightarrow$ something X Y

Variables: X Y
H1 : plus X Y (s (s z))

\begin{align*}
\text{Subgoal 1} & \quad \text{something} \quad X \quad Y \\
\text{Subgoal 2 is:} & \quad \text{something} \quad z \quad s \quad (s \quad z) \\
\text{Subgoal 3 is:} & \quad \text{something} \quad (s \quad (s \quad z)) \quad z
\end{align*}
Imagine we have the following hypothesis:

\[ H_1 : \text{is_nat} \ (s \ (s \ X)) \]

\( H_1 \) cannot be eagerly solved:

\[ \sim \ \text{is_nat} \ X \quad \Rightarrow \quad X = z \]
\[ \lor \]
\[ X = (s \ X_1) \]
\[ \text{is_nat} \ (s \ X_1) \sim \text{is_nat} \ X_1 \quad \Rightarrow \quad X_1 = z \]
\[ \lor \]
\[ \ldots \]

We need a way to prevent unproductive unfoldings.
New **Suspend** declarations to make Abella stop the asynchronous phase prematurely.

**Suspend** nat X on X.

means "(nat X) should not be unfolded if X is a variable"

\[
\text{nat (s (s X))} \rightarrow \text{nat X}
\]

**Suspend** plus X Y _ on X, Y.
2\textsuperscript{nd} proposal: Deterministic computation
The polarity ambiguity of singleton

If \( p \) is a singleton (that is a monadic predicate that holds for exactly one argument) then:

\[
\forall x, p\,x \rightarrow Q\,x \equiv \exists x, p\,x \land Q\,x
\]

In Abella, a definition for singleton would be:

Define \text{singleton} : (A \rightarrow \text{prop}) \rightarrow \text{prop} by
singleton \( P \) :=
\[
\big( \exists X, P\,X \big) \\
\land \big( \forall X\,Y, P\,X \rightarrow P\,Y \rightarrow X = Y \big).
\]
The polarity ambiguity of singleton

We admit the definition singleton to Abella.

Trying to prove $\text{exists } x, \ p \ x \land Q \ x,$ if singleton $p$ holds then the problem of guessing a witness term $t$ becomes:

- Transforming the goal $\text{exists } x, \ p \ x \land Q \ x$ into $\text{forall } x, \ p \ x \rightarrow Q \ x$
- Introducing the variable and its hypothesis ($\text{intros}$)
- Using $\text{compute}$ on that hypothesis

It allows use to switch between to paradigms :

$$\text{Guess and check } \rightarrow \text{ Compute}$$
Singleton and functions

Singleton actually arise whenever a relation is actually a function:

**Theorem** plus_funct:

\[
\forall X \ Y, \text{is_nat} \ X \rightarrow \text{is_nat} \ Y \rightarrow \text{singleton} \ (\text{plus} \ X \ Y).
\]

This theorem is an ordinary Abella theorem that can be readily proved by induction on (\text{is_nat} \ X).
Witness compute

When the goal has the form:

\[ \exists X, \, P(X) \land Q(X) \]

**witness compute will**

1. Try to prove (singleton P)
2. Switch \( \exists \) and \( \forall \)

\[ \forall X, \, P(X) \rightarrow Q(X) \]

3. Use **intros**:

   \( H1 : P(X) \)  \( \text{(with X an eigenvariable)} \)

   \[ Q(X) \]

4. Use **compute** \( H1 \) to actually compute the witness
Dually, whenever we have a hypothesis of the form:

\[ H : \forall X, P X \rightarrow Q X \]

then invoking `apply compute \ H` has the effect of first trying to prove `(singleton P)` and then continuing with the new hypotheses where `X` is an eigenvariable:

\[ H1 : P X \]
\[ H : Q X \]

following up with `compute \ H1`.
This small extension to Abella is orthogonal to its core. No change was made to the underlying logic:

- `compute / Suspend`
- `singleton / witness compute / apply compute`

These proposals could be generalized:

- Default suspend declarations?
- The notion of singleton could be relaxed to a notion of singleton up to equivalence
- Deal with data defined by higher-order type signatures.
Thank you.
Baelde, D. (2012). **Least and greatest fixed points in linear logic.**

*ACM Trans. on Computational Logic*, 13(1).


*Journal of Formalized Reasoning*, 7(2).

Gérard, U. and Miller, D. (2017). **Separating functional computation from relations.**