Computation-as-deduction in Abella: work in progress

Kaustuv Chaudhuri, **Ulysse Gérard** and Dale Miller

LFMTP, July 7, 2018

Inria Saclay
Palaiseau France

Introduction

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 $\mathcal{G}$-logic and focusing

The $\mathcal{G}$-logic in Abella [Baelde et al., 2014]

An extension of intuitionistic first-order logic with

- Higher-order λ-terms with $\alpha\beta\eta$-equivalence
- Inductive and coinductive fixed point definitions
- Nominals, nominal abstraction and generic (∇) quantification.

The $\mathcal{G}$-logic in Abella [Baelde et al., 2014]

$\mathcal{G}$'s terms are well-typed terms of Church's simple theory of types, a given type signature declares:

- basic types (keyword `Kind`)
- constants which are constructors for these basic types (`Type`).

```
Kind bool   type.
Type tt, ff bool.

Kind nat  type.
Type z    nat.
Type s    nat → nat.
```

The $\mathcal{G}$-logic in Abella

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
  is_nat z;
  is_nat (s X) := is_nat X.
```

The $\mathcal{G}$-logic in Abella

```
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`.

Focusing

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)

Focusing

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 (=, $\wedge$, $\vee$, `false`, and `exists`)
- Or it appears as a goal and is made of negative connectives ($\wedge$, `true`, $\rightarrow$, and `forall`)

1^{st} proposal: Compute and suspend

Compute

The `compute` tactic performs unfolding and subsequent asynchronous steps for assumptions involving fully positive definition predicates.

```
forall X, plus (s z) (s z) X → X = s (s z)
```

```
intros.
compute H1.
search.
```

```
==============================
 forall X, plus (s z) (s z) X
                → X = s (s z)


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


==============================
 s (s z) = s (s z)


Proof completed.
```

Compute can branch...

The `compute` tactic can lead to multiple subgoals: predicates.

```
forall X Y, plus X Y (s (s z)) → something X Y
```

```
intros.
compute H1.
```

```
Variables: X Y
H1 : plus X Y (s (s z))
============================
something X Y

Subgoal 1
============================
 something z s (s z)

Subgoal 2 is:
 something (s z) (s z)

Subgoal 3 is:
 something (s (s z)) z
```

Compute can loop...

Imagine we have the following hypothesis:

```
H1 : is_nat (s (s X))
```

`H1` cannot be eagerly solved:

```
⇝ is_nat X    > X = z
                ∨
             X = (s X1)
             is_nat (s X1) ⇝ is_nat X1    > X1 = z
                                             ∨
                                            ...
```

We need a way to prevent unproductive unfoldings.

… Suspend

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"

$$\texttt{nat (s (s X))} \rightarrow \texttt{nat X}$$

`Suspend plus X Y _ on X, Y.`

2^{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:

$$\texttt{forall x, p x} \rightarrow \texttt{Q x} \equiv \texttt{exists x, p x} \wedge \texttt{Q x}$$

In Abella, a definition for singleton would be:

```
Define singleton : (A → prop) → prop by
  singleton P :=
      (exists X, P X)
    ∧ (forall X Y, P X → P Y → X = Y).
```

The polarity ambiguity of singleton

We admit the definition `singleton` to Abella.

Trying to prove `exists x, p x ∧ Q x` , if `singleton p` holds then the problem of guessing a witness term `t` becomes:

- Transforming the goal `exists x, p x ∧ Q x` into `forall x, p x → Q x`
- Introducing the variable and its hypothesis (`intros`)
- Using `compute` on that hypothesis

It allows use to switch between to paradigms :

Guess and check $\longrightarrow$ **Compute**

Singleton and functions

Singleton actually arise whenever a relation is actually a function:

```
Theorem plus_funct:
 forall X Y, is_nat X → is_nat Y → singleton (plus X Y).
```

This theorem is an ordinary Abella theorem that can be readily proved by induction on (`is_nat X`).

Witness compute

When the goal has the form:

```
========================================
 exists X, P X ∧ Q X
```

`witness compute` will

1. Try to prove `(singleton P)`
2. Switch ∃ and ∀

   ```
   ========================================
    forall X, P X → Q X
   ```

3. Use `intros` :

   ```
   H1 : P X        (with X an eigenvariable)
   ========================================
    Q X
   ```

4. Use `compute H1` to actually compute the witness

Apply compute

Dually, whenever we have a hypothesis of the form:

```
H : forall X, P X → Q X
```

then invocating `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`.

Conclusion and perspectives

This small extension to Abella is orthogonal to it's 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).

Baelde, D., Chaudhuri, K., Gacek, A., Miller, D., Nadathur, G., Tiu, A., and Wang, Y. (2014).
Abella: A system for reasoning about relational specifications.
Journal of Formalized Reasoning, 7(2).

Gérard, U. and Miller, D. (2017).
Separating functional computation from relations.
In Goranko, V. and Dam, M., editors, *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *LIPIcs*, pages 23:1–23:17.