Reasoning with contexts in Abella

Olivier Savary-Belanger
McGill University
osavary@cs.mcgill.ca

Kaustuv Chaudhuri
INRIA Saclay Ile-de-France
kaustuv.chaudhuri@inria.fr

Abella (Gacek 2008) is an interactive theorem prover developed in collaboration at INRIA Saclay and U. of Minnesota. A formal development in consists of:

- a specification file, where the meta-theory of the studied systems are defined,
- a reasoning file, where theorems about the specifications are stated and proved using provided tactics to construct the theorem conclusion from its assumptions.

Abella allows for higher-order reasoning; terms may introduced assumption to a list representing the local context of assumption. It however doesn't provide specific facilities to define context, such as regular worlds in Twelf (Pfenning and Schürmann 1999) and schemas (Pientka and Dunfield 2010). Instead, users need to define a predicate characterizing which assumptions can appear in the context, and prove a number of structural theorems to reason about members of such context.

We developed a plugin for Abella introducing Schema, which restricts context definitions to a well-behaved fragment, and tactics stating, proving and applying theorems which always hold for context defined in our fragment. This greatly reduces the amount of boilerplate code needed for contextual reasoning in Abella.

Contexts, as defined in our plugin, can be described a star of finite sum of clauses, each of which adds a formula that can introduce some nominal and some existentially bound variables(talk about dependencies), on top of the context. N-ary context relations, where formulas introduced in n different contexts can depends on the same variables, can also be represented. N-ary context relations are represented using n lists of formulas, each clause corresponding to a constructor stating that, given such context relations, adding the formulas in the clauses in front of their respective list relate the extended lists. An additional constructor corresponding relating empty contexts is added to every context relation.

The tactics introduced by our plugin are:

- Inversion, which states that we can characterize members of the contexts by one of the clauses which could have introduced them
- Synchronize, which states that given a member of a context in a context relation, clauses which could have introduced that term introduced at the same time corresponding members in the other projections of the context relation.
- Unique, which states that two members of a context sharing nominal variables should be equal.
- Projection, which helps reusing contexts in different context relations by verifying projection and injection theorems.

These tactics eliminates most of the boilerplate lemmas present in the Abella example suite developments. For examples, using our plugin to rewrite the $bredure$ example eliminates half of the lines of code in the reasoning ($.thm$) file. Due additional context relations and mappings between context relations are present in bigger developments, we believe the provided tactics can significantly reduce the size of Abella development using reasoning with contexts.

An important design decision of our development is the theorem generated are proved using the provided tactics of Abella, and checked as regular theorems. This means that no logical inconsistency can be introduced by our plugin. This contrasts with theorem provers providing primitive notions of context, such as Twelf (Pfenning and Schürmann 1999) and Beluga (Pientka and Dunfield 2010), where the properties we prove with our new tactics are part of the trusted code base.

The extension we describe adds both top level commands and tactics to Abella. Instead of modifying the parser and main function to support the new functionality, we devised a plugin framework where an arbitrary expression can be dispatched to a particular plugin from an Abella source file. Plugins are implemented as modules implementing a processing function for string received from the top level of the source file, and another processing function for string received as reasoning command. Each processing function also receives a function which allows to recursively process strings as if their appeared in their respective position in the source file. Processing function at the reasoning command level can be related to tacticals languages such as Ltac (Delahaye 2000), which permits Coq user to develop procedures which applies tactics in function of current goal. In comparison, we allow arbitrary, potentially unsafe Ocaml code in the plugin, with the restriction that the only entry point back in the prover is the passed function. In the future, one would like to provide functions for plugins to safely interact with the prover, which could culminate in a language to define extensions and tacticals from within Abella source files.

References

D. Delahaye. A tactic language for the system coq. In *International conference on Logic for programming and automated reasoning*, pages 85–95, 2000.

A. Gacek. The Abella interactive theorem prover (system description). In *International Joint Conference on Automated Reasoning*, pages 154–161, 2008.

F. Pfenning and C. Schürmann. System description: Twelf - a meta-logical framework for deductive systems. In *International Conference on Automated Deduction*, pages 202–206, 1999.

B. Pientka and J. Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *International Joint Conference on Automated Reasoning*, pages 15–21, 2010.