Final Report for Deliverable Nr. 1.2

SSReflect Library and Documentation

Responsible: Georges Gonthier, Georges.Gonthier@inria.fr
Written by: Enrico Tassi, enrico.tassi@inria.fr
Site: INRIA, France

Deliverable Date: February 2011
Software Release Date: March 2011

Version 1.3 of the Small Scale Reflection extension [1] for Coq 8.3 and its library has been made available to the public on the 11th of March 2011.

The name SSReflect stands for "small scale reflection", a style of proof that evolved from the computer-checked proof of the Four Colour Theorem and which leverages the higher-order nature of Coq's underlying logic to provide effective automation for many small, clerical proof steps. This is often accomplished by restating ("reflecting") problems in a more concrete form, hence the name. For example, in the SSReflect library arithmetic comparison is not an abstract predicate, but a function computing a boolean.

SSReflect is both a proof shell extension for the Coq system and a library of formalized mathematics. It is the tool of choice for the Formath Project. The release of version 1.3 consisted in:

- Porting the extension to the new version of Coq, namely 8.3 patch level 1
- Extending the proof language, supporting new linguistic idioms
- Enriching the library of formalized mathematics shipped with SSReflect
- Updating and improving the documentation of the proof language and of the libraries of formalized mathematics
- Providing a new tutorial to the SSReflect proof language and its base libraries

1 SSReflect library

The previous version of SSReflect's library, named 1.2, provided 28 distinct components amounting to a bit less than 30 thousands lines. Version 1.3 provides 54 components for a total of 70 thousands lines of definitions and formal proofs. These new components have been validated on the ongoing proof of the Odd Order theorem, that at the actual state amounts to other 20 thousands lines of of formal proofs developed on top of the SSReflect library.

In the following we mention the components that have been added to the SSReflect library. Every module includes a detailed documentation concerning the notations, the definitios and the main results available to the user.

Here we briefly describes their content only.

- `finalg`: algebraic hierarchy for finite types
- `perm`: finite permutation groups
- `presentation`: generator and relation group presentations
- `action`: finite group actions, orbits, stabilizers
- `gproduct`: direct, semidirect and central group products
- `gfunctor`: (characteristic) group functor hierarchy
- `zmodp`: properties of $\mathbf{Z}/n\mathbf{Z}$
- `center`: properties of group centers
- `commutator`: properties of commutator subgroups
- `gseries`: normal, central, chief group series
- `jordanholder`: sections, factors and the Jordan-Holder theorem
- `nilpotent`: nilpotent and solvable groups, lower and upper series
- `pgroup`: π-groups, π-cores, Hall and Sylow subgroups
- `sylow`: the Sylow theorems and its consequences; the Baer-Suzuki theorem
- `primitive_action`: primitive and n-transitive actions
- `alt`: symmetric and alternating groups
- `abelian`: structure of abelian groups; homocyclic and elementary abelian groups; group p-rank and rank
- `finmodule`: finite modules, transfer, and the Gaschütz theorems
- `maximal`: Frattini, Fitting, special, extraspecial, and critical subgroups
- `hall`: the Schur-Zassenhaus and Hall theorems; coprime action
- `extremal`: classification of extremal p-groups (modular or (generalized) dihedral/quaternion)
- `extraspecial`: classification of extraspecial groups
- `frobenius`: Frobenius groups, semiregular and semiprime action

- `mxalgebra`: matrix rank, row spaces, and subalgebras
- `vector`: finite dimensional abstract linear algebra
- `mxpoly`: minimal and characteristic polynomials, resultants
- `mxrepresentation`: (modular) finite group representation theory
- `mxabelem`: representation induced by normal elementary abelian subgroups

2 SSReflect proof language

The proof language provided by the SSReflect extension has been improved in many respects. Here we summarize the most relevant ones, for a complete account of the new linguistic constructs refer to [3] section 12.

contextual rewrite patterns The user can now resort to a more expressive language to drive the main proof command of the SSReflect proof language: `rewrite`. In particular the subterms of the goal to be rewritten can be identified mentioning their context. This is of particular interest when the subterm to be rewritten is large and very similar, if not identical, to other subterms, but the context in which it occurs is not.

For example, the pattern `[in X in _ = X]` forces the rewrite to happen only on the right hand side of an equational goal. Note that the right hand side of the goal may be a very complex and large expression, the user is not asked to type. Moreover contextual pattern almost always allow to avoid identifying subterms by means of occurrence numbers, that are fragile in face of proof script refactoring and also uninformative for the reader.

views inside introduction patterns View lemmas are probably the most characterizing feature of the SSReflect proof language and formalization style. They relate multiple presentation of same concept, like the computational and the propositional ones. For example decidable predicates, represented by computable functions to `bool`, are composed together with boolean, computable, connectives. Simple views relate boolean connectives to the propositional ones. This is necessary to use standard Coq standard proof commands, that are designed to work in the propositional fragment of the logic. Examples are the commands to apply elimination rules. The SSReflect proof language enables most proof commands to pre-process their argument with views, so that their correct presentation can be obtained with a minimal effort. For example given two decidable predicates `P` and `Q`, and an assumption that their boolean conjunction `P && Q` holds, one can get that `P` and `Q` hold with the following proof commands:

```
Lemma example:  forall P Q, P && Q -> ...
move=> P Q; case/andP; move=> P_holds Q_holds
```

where `andP` is a view relating boolean conjunction (`&&`) and propositional conjunction ($\wedge$), and `case` applies the elimination rule. The `move` command, here used to introduce assumptions, has a concise syntax to perform elimination, but since a view is needed, the user has to explicitly mention it as the flag of the case analysis command.

SSReflect 1.3 allows views to be mentioned inside introduction patterns, reducing the verbosity of proofs. The former proof line can now be replaced by the more compact:
`move=> P Q /andP [P_holds Q_holds]`
Although very simple in its idea, this change has a great impact on all proof commands, since introduction patterns are used whenever a fact is named. Most notably, forward reasoning steps can now easily state an intermediate result in its presentation more suitable to be proved, but introduce it into the context under a different presentation, possibly more suitable to be reused later on.

second order predicate synthesis In the higher order logic of Coq, elimination principles can be expressed as second order lemmas quantified of a predicate. For example induction over natural numbers is usually performed applying a lemma of type:
`forall P : nat -> Prop, P 0 -> (forall n, P n -> P n.+1) -> forall n, P n`
Since the conclusion of the lemma has to match the goal the user is trying to prove, higher order unification has to figure out what `P` is, that is usually omitted being in general an expression as big as the goal itself. Since higher order unification problems do not admit a single, most general, solution the user may be forced to specify `P` by hand.

SSReflect 1.3 enriched the `elim` proof command with a language of patterns that allows the user to succinctly specify how to synthesize `P` without having to type it as a whole. In addition to that, `elim` also supports general second order lemmas, not necessarily elimination principles for inductive types. This greatly improves the compactness of proof scripts dealing with iterated operations, since they are equipped with higher order elimination principles that the standard `elim` proof command is unable to handle.

3 User manual

The user manual, published as technical report [3], has been updated according to the additions to the SSReflect proof language and library described above. The extensive section 12 (Changes) details all the changes to the proof language, pointing to the relevant parts of the manual that were updated.

4 Tutorial

A tutorial one the basic features of the distributed libraries has been published as an article in the Journal of Formalized Reasoning [2]. This tutorial is accompanied by online solutions to the exercises, accessible at: `http://www.msr-inria.inria.fr/Projects/math-components/index_html`

References

[1] Small Scale Reflection extension for the Coq system. `https://www.msr-inria.inria.fr/Projects/math-components/index_html#download`.

[2] Georges Gonthier and Assia Mahboubi. An introduction to small scale reflection in Coq. *Journal of Formalized Reasoning*, 3(2):95–152, 2010.

[3] Georges Gonthier, Assia Mahboubi, and Enrico Tassi. A small scale reflection extension for the coq system. Technical Report RR-6455 inria-00258384, version 5 or above, INRIA, 2011. `http://hal.inria.fr/inria-00258384`.