Specification & Verification Assignment – Theorem Provers

Matthew Dyson

Introduction

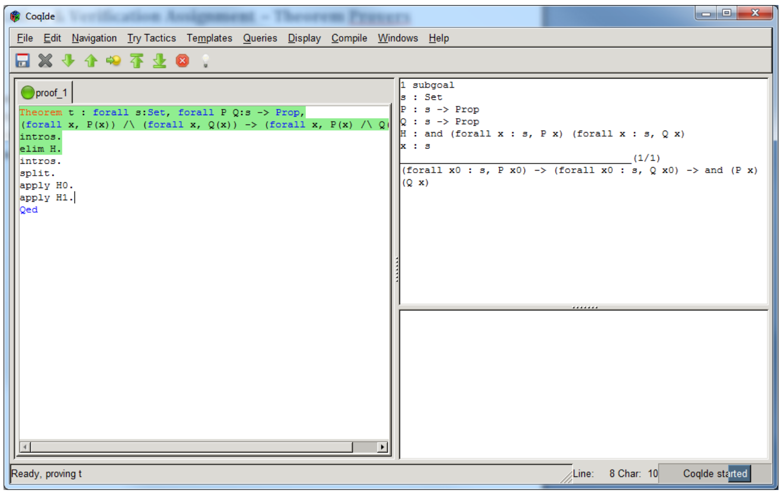
This assignment asked to examine 2 different software theorem provers, one automatic and one assisted, in order to form a comparative analysis. I chose to use Coq and Prover9 for this task, giving a recognisable contrast on which to compare their functionality, ease of use, key features, and other strengths & weaknesses. As part of this, the predicate logic formula given below was to be proved. For each of the provers, I have given the necessary input for the tool to handle the formula, and any pertinent output.

$$\forall x(\, P(x) \wedge Q(x)\,) \vdash \forall x. P(x) \wedge \forall x. Q(x)$$

The first two sections of this document describe the software in general, and give details of the steps necessary to input and prove the formula above. The third section compares and contrasts the two programs, and finally a conclusion is given.

Coq

Coq is an implementation of the high-level mathematical language Gallina (1) that allows the creation of formal proofs. It is implemented in Objective Caml based around a command line client, but comes bundled with a front-end GUI (CoqIDE) which simplifies the functionality available within the program to allow the user to access more advanced features easily.


CoqIde
File Edit Navigation Try Tactics Templates Queries Display Compile Windows Help
proof_1
Ready, proving t
Line: 8 Char: 10
CoqIde started

```
Theorem t : forall s:Set, forall P Q:s -> Prop,
(forall x, P(x)) /\ (forall x, Q(x)) -> (forall x, P(x) /\ Q(
intros.
elim H.
intros.
split.
apply H0.
apply H1.
Qed
```

```
1 subgoal
s : Set
P : s -> Prop
Q : s -> Prop
H : and (forall x : s, P x) (forall x : s, Q x)
x : s
______________________________________(1/1)
(forall x0 : s, P x0) -> (forall x0 : s, Q x0) -> and (P x)
(Q x)
```

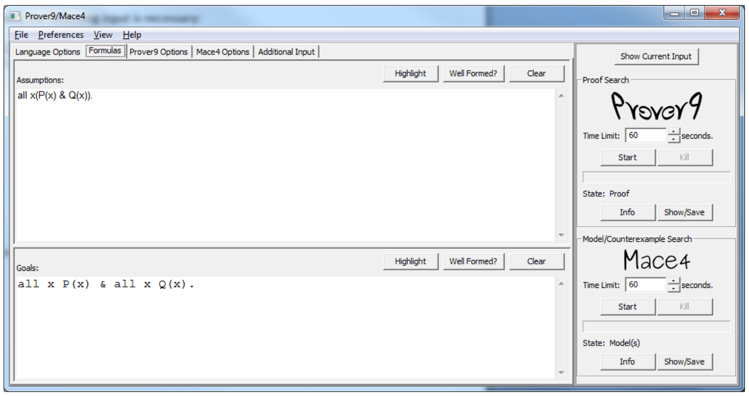
When the user has inputted the proof they wish to test into the proof window (left hand side), the proof can be broken down and executed step-by-step, with the interface showing the variables, assumptions and progress within the right hand window. As Coq is a proof assistant, it does not offer any suggestion to the user as to how the input should be solved, instead giving a library of common functions that the user must choose how to apply.

In solving the given equation, the following input is necessary:

```
Theorem proof : forall s:Set, forall P Q:s -> Prop, (forall x, P(x)) /\ (forall x,
Q(x)) -> (forall x, P(x) /\ Q(x)).
intros.
elim H.
intros.
split.
apply H0.
apply H1.
Qed.
```

Prover9

Prover9 is a completely automated proof solver, based upon the Otter theorem prover (2), that accepts first-order and equational logic and comes bundled with Mace4, a first-order logic model searcher which is used to attempt to find counter-examples (3). It is still under production as part of an open-source project, with regular releases.



Assumptions:
all x(P(x) & Q(x)).

Goals:
all x P(x) & all x Q(x).

Time Limit: 60 seconds.
State: Proof

Time Limit: 60 seconds.
State: Model(s)

The graphical user interface is split into two text entry boxes, which allow the user to input their assumptions and goals, which can then be automatically hi-lighted in order to identify any grammatical errors, and also be checked for sense. Limitations can be set on execution time, and then the input can be run through either the Prover9 or Mace4 engine as appropriate. Due to the nature of automatic solving the validation of generated proofs is difficult, since the engine that the prover is based on is extremely hard to formally verify. Other tools have been developed in order to meet the purpose of validating proofs in order to overcome this problem, one such being Ivy (4).

To solve the given equation, the following input is necessary:

Assumptions:
```
all x(P(x) & Q(x)).
```

Goals:
```
all x P(x) & all x Q(x).
```

This gives the following output:

```
1     (all x (P(x) & Q(x))) # label(non_clause).  [assumption].
2     (all x P(x)) & (all x Q(x)) # label(non_clause) # label(goal).  [goal].
3     -P(c1) | -Q(c2).  [deny(2)].
4     P(x).  [clausify(1)].
5     -Q(c2).  [resolve(3,a,4,a)].
6     Q(x).  [clausify(1)].
7     $F.  [resolve(5,a,6,a)].
```

Analysis & Comparison

In terms of usability, there is not a great deal of difference between the two tools. Both are well known in the field of theorem provers, and hence have fairly large online communities and support websites through which a user can learn everything from the general working right down to the more advanced features. The user interface of both tools is fairly minimalist, with the more advanced features being hidden away in menus rather than displayed to the user at first glance. To the novice user this may be appealing, although personally I found this frustrating when trying to find certain commands.

The output from Prover9 for larger proofs can at times be extremely hard to follow, as it adopts an exhaustive method in order to prove the goal. Because of this, other programs have been developed (discussed earlier) in order to validate the output of Prover9 (4) as the output may not always be human-readable or even human-interpretable. This severely limits the usability of the program in aiding the users understanding of formal logic.

Coq uses the specialist high-level language Gallina (1) in order to generate input sequences, as previously mentioned. This method of pseudo-programming would make the formulation process a lot easier to people with even the most basic programming skills, however the actual syntax is not immediately obvious to the user, necessitating further reading into the reference manual (5).

Prover9 and Coq are designed and built to appeal to two different audiences with contrasting requirements from theorem prover software. The ability to use Prover9 in a completely automated manner is obviously of more use to the less-experienced user, or even people wishing to learn how to formulate proofs, allowing them to see the entire process broken down step-by-step. In contrast to this, Coq allows the user to formulate the entire proof themselves, meaning it would appeal more to the more experienced user, who would be able to interpret the program output and manipulate this into larger proofs, essentially becoming a method of validating existing proofs, rather than a tool to build them with. For this reason, I would choose Coq over Prover9 for validating formal correctness of any self-written algorithm, although Prover9 would prove more useful in the design stage.

Conclusion

In conclusion, both of the tools I have examined in this report have various strengths and weaknesses, whether these be intentional or accidental, that gear them towards a certain user group. For this reason, someone wishing to use a theorem prover in their line of work should consider their options very carefully, to avoid wasting time reading too far into a program when others are available.

References

1. **ADT Coq.** What is Coq? *The Coq Proof Assistant.* [Online] [Cited: 30 04 2010.] http://coq.inria.fr/what-is-coq.
2. **Prover9.** *Prover9 & Mace4.* [Online] [Cited: 30 04 2010.] http://www.cs.unm.edu/~mccune/mace4/.
3. **Mathematics & Computer Science Division.** Mace4 Model Searcher. *Argonne National Laboratory.* [Online] [Cited: 30 04 2010.] http://www.mcs.anl.gov/research/projects/AR/mace4/.
4. **Matlin, William McCune & Olga Shumsky.** Ivy Checker. [Online] [Cited: 30 04 2010.] http://www.cs.unm.edu/~mccune/papers/ivy/.
5. **ADT Coq.** Reference Manual. *The Coq Proof Assistant.* [Online] [Cited: 02 05 2010.] http://www.lix.polytechnique.fr/coq/refman/.