Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Alexander Krauss, Sascha Böhme

WS 2008/09
19. 12. 2008

Logic

Mini-Project 2

Deadline: Monday, 11 January 2009, 8:00 a.m.
Submit to: `krauss@in.tum.de`

Finite Model Generation

Just like programs, formal specifications are often wrong in the first attempt. So, instead of trying a proof (say, using resolution or the sequent calculus), we may first want to find out if $\neg c$ is satisfiable. If it is, then c is invalid and we should not bother proving it. Furthermore, the model for $\neg c$ can give us hints about the bug.

This project is about automatic model generation. You will develop a procedure that, given a first-order sentence p, searches for a finite model M such that $M \vDash p$. If we are interested in proving c, we can search for models of $\neg c$ to make sure that there are none.

The general idea is to transform a first-order sentence p into a propositional formula f_p, such that f_p is propositionally satisfiable iff p has a model of size n, where n is a not-too-large natural number. We can then use SAT solving to construct a model of p from a satisfying valuation[1].

Exercise 2.1 Translating first-order logic without functions

At first we consider only formulas that do not contain function symbols, e.g.,

$p_1 = (\forall x.P(x)) \Rightarrow (\exists x.P(x))$.

a) Implement a function

prop_translate : int $\rightarrow$ *fol formula* $\rightarrow$ *prop formula*

such that *prop_translate n p* returns a propositional formula p' that is satisfiable iff p has a model of size n.

For example, *prop_translate* 3 p_1 should produce something like $\left(\bigwedge_{0 \leq i < 3} P_i\right) \Rightarrow \left(\bigvee_{0 \leq i < 3} P_i\right)$, where P_0, P_1, P_2 are propositional variables.

b) Implement a funtion *find_rel_model* that constructs a model M from a satisfying valuation for p', such that $M \vDash p$.

c) Extend your implementation, such that it only produces *normal models*. A model M is called *normal* iff the interpretation $=_M$ of the relation symbol $=$ is the normal equality on D_M.

[1] If you haven't done the first project, you can take a SAT solver that produces a satisfying valuation from the solution of that project.

Exercise 2.2 Compiling away functions

In order to be able to handle functions as well, we represent them as relations. For each n-ary function f we introduce an $(n+1)$-ary relation F, such that $F(x_1, \ldots, x_n, y)$ models that $f(x_1, \ldots, x_n) = y$.

a) When translating functions to relations we must make sure that the relations we produce really represent functions. Find suitable first-order formulas expressing that a relation F behaves like a function.

Hint: Make sure that your formulas map to a reasonable CNF when translated to propositional logic.

b) Implement a translation that removes functions and replaces them by relations.

For example,

$$P(f(x), g(h(y)))$$

should be translated to

$$\forall z_1 z_2 z_3 . F(x, z_1) \Rightarrow H(y, z_2) \Rightarrow G(z_2, z_3) \Rightarrow P(z_1, z_3).$$

c) Using the components that you have, write a function

find_model : int $\rightarrow$ *fol formula* $\rightarrow$ *int model option*

that constructs finite (normal) models for arbitrary first-order formulas.

Here, *'a model* abbreviates *'a list * (string* $\rightarrow$ *'a list* $\rightarrow$ *'a) * (string* $\rightarrow$ *'a list* $\rightarrow$ *bool)*, the type of finite models with domain from *'a*.

Also provide a function *refute* that searches for countermodels for a conjecture c.

d) Specify soundness and completeness for *find_model*.

Exercise 2.3 Experiments

Note: Since our refutation algorithm is quite naive, it will only handle small examples successfully. Don't expect too much if you do your own experiments.

a) Use *refute* to show that the cancellation law $x + z = y + z \Rightarrow x = y$ does not hold for monoids.

b) Write a function

min_size_axiom : int $\rightarrow$ *fol formula*

such that for any $n > 0$, *min_size_axiom* n has no model of size n, but one of size $n + 1$.

c) Show that the following algorithm is not a decision procedure for first-order logic:

> Given a conjecture c, run two processes in parallel: The first process tries to prove c using resolution, the second process tries to refute c by searching for countermodels of increasing size. When any of the processes has terminated, we know if c is valid.

Coding and Documentation Guidelines

- Follow the style of coding that you find in the book. You can also use any pieces of the existing code base, in particular the library.
- Keep your code purely functional. You will not need any imperative features of OCaml.
- Do not do low-level optimizations. Always prefer clarity and conciseness over efficiency.
- Find a suitable way of documenting your implementation. This can be comments in the code, a separate text, or a text that includes code snippets as you find it in the book.
- Keep in mind that extensive comments cannot make up for bad code. On the other hand, clear and readable code can make many comments obsolete.

Recommended setup for this project:

```
#use "init.ml";;                              (* load everything *)
```

In case you like to use the solution of the previous project, add:

```
#use "sudoku.ml";;
#use "proj1.ml";;
```