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

WS 2008/09
7. 11. 2008

Logic

Mini-Project 1

Deadline: Monday, 24 November 2008, 8:00 a.m.
Submit to: `krauss@in.tum.de`

Exercise 1.1 DPLL with explicit solutions

a) Modify the iterative DPLL implementation *dpli*, such that it returns a satisfying valuation if it finds out that the clauses are satisfiable. That is, write a function

dplic :: prop formula list list $\rightarrow$ *(prop* $\rightarrow$ *bool) option*

b) State the correctness property that you expect from *dplic*.
Implement appropriate tests.

Exercise 1.2 Sudoku solving with DPLL

A Sudoku is a 9×9 grid where each cell contains a digit from 1 to 9. In a valid solution every digit occurs exactly once in each row, each column, and each of the 3×3 blocks. A puzzle is given by an incomplete grid and the task is to complete it to a valid solution.

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

a) Write a function that can solve Sudoku puzzles by encoding them into propositional satisfiability problems. That is, given a puzzle, produce a set of clauses that is satisfiable iff the Sudoku has a solution. Then, use the modified solver *dplic* to find a solution, from which you read the completed Sudoku.

Note: You are not supposed to program your own Sudoku solving strategy. Instead, use the generic tools that we have already developed.

The file `sudoku.ml` has a definition of a Sudoku type and a simple parser that can read problems from a text file. We also provide a collection of Sudokus that you can use to test your implementation.

b) State the correctness property that you expect from your Sudoku solver.
Implement appropriate tests. Is our collection of sudokus a sufficient test suite?

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. It is perfectly fine if your solver takes a few seconds per Sudoku. If it takes much longer, think about your encoding and implementation.
- 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 *)
let default_parser = parse_prop_formula;;  (* set parser for propositional logic *)
#use "sudoku.ml";;                        (* sudoku tools *)
let sudokus = read_sudokus "sudokus.txt";; (* read sudokus from file *)
```