Pearls of Computer Science 3 - WS 2012/13
Tutorial 1 (for Monday, 10.12.2012)

Corneliu Popeea (popeea@model.in.tum.de)

This tutorial will guide us through implementing a boolean satisfiability solver based on the DPLL algorithm. We will write a procedure that takes as input a boolean formula in conjunctive normal form (CNF) and either returns a satisfying assignment for the given formula or a value signifying that the input formula is unsatisfiable.

1. Install the OCaml compiler (Windows / Linux / Mac). `http://caml.inria.fr/download.en.html`
2. Download and install the Emacs editor (Windows / Linux / Mac). `http://www.gnu.org/software/emacs/`
3. Download an archive tutorial1.tgz containing the files types.mli, sat.ml, homework.ml, parser.ml, Makefile. `http://www7.in.tum.de/um/courses/pearl3/ws1213/tutorial1.tgz`
4. Discuss the types used for representing formulas in conjunctive normal form (types.mli)
5. Discuss the functions that check whether a formula is satisfiable (sat.ml)

   ```
   val sat:  formula -> asgn option
   val sat_and_print:  formula -> unit
   ```
6. Compile the solution using the provided makefile. Is the result for the formula phi0 correct? If not, why not?
7. A manual for the OCaml language and documentation for the standard library are available:
   `http://caml.inria.fr/pub/docs/manual-ocaml/libref/index.html`
   `http://caml.inria.fr/pub/docs/manual-ocaml/`

Homework

Submit your solution (homework.ml) via email to popeea@model.in.tum.de. Deadline 17Dec, 10am.

1. Write two functions that update a clause/formula to set a literal to true. Fill in the function definitions given in `homework.ml`.

   - `val set_clause :  Types.clause -> Types.lit -> Types.clause option`
   - `val set_formula :  Types.formula -> Types.lit -> Types.formula option`

   Now you have a complete satisfiability solver! Check for satisfiability the test formulas, phi0, phi1, phi2, phi3.
2. Write two functions that implement unit propagation. Fill in the function definitions given in `homework.ml`. The function `sat` invokes these functions, but the code is commented out. Uncomment the code that calls these functions and check phi0, phi1, phi2, phi3.

   - `val find_unit_clause :  Types.formula -> Types.lit option`

  - `val unit_propagate :  Types.formula -> (Types.formula * Types.lit) option`

3. Bonus topic: benchmark problems for SAT solvers are widely available. A parser for these benchmarks (in DIMACS format) is included (parser.ml). My implementation is quite fast for the first set of problems from:
   `http://www.cs.ubc.ca/ hoos/SATLIB/benchm.html`

   For the first 1000 problems, it takes approximately 7s without unit propagation or 4s with unit propagation on my Intel Core 2 Duo laptop. Can you find examples where this naive implementation takes long time?