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

WS 2008/09
31. 10. 2008

Logic

Exercise Sheet 3

Exercise 3.1 Warm up

a) Put the following formula in conjunctive normal form:

$$\neg(((C \Rightarrow A) \wedge (A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow ((C \Leftrightarrow A) \vee (B \wedge C)))$$

b) Use resolution to find out if it is satisfiable.

Solution

a) We get the clauses $\{\neg C, A\}, \{\neg A, B\}, \{\neg B, C\}, \{\neg C, \neg A\}, \{C, A\}, \{\neg B, \neg C\}$

b) Resolving $\{\neg C, A\}$ with $\{C, A\}$ yields $\{A\}$.
Resolving $\{\neg A, B\}$ and $\{\neg B, C\}$ yields $\{\neg A, C\}$.
Resolving $\{A\}$ with $\{\neg A, C\}$ yields $\{C\}$.
Resolving $\{C\}$ with $\{\neg C, \neg A\}$ yields $\{\neg A\}$.
Resolving $\{A\}$ with $\{\neg A\}$ yields $\emptyset$.
Thus the formula is unsatisfiable.

Exercise 3.2 Pure Literal Rule

The *pure literal rule* is the following transformation of clause sets:

> If a literal p occurs only positively or only negatively, then we remove all clauses containing it.

Show that applying this rule to a set of clauses yields a clause-set that is equisatisfiable.

Solution

See book, pp. 81–82.

Exercise 3.3 Proof-Producing Resolution

Section 2.9 in the book gives the code for a resolution-based SAT procedure, using the resolution rule, the pure literal rule and a unit propagation rule which is just a special case of resolution.

Extend this implementation to make it proof-producing: Instead of just returning *false* if the clauses are unsatisfiable, your procedure should return a *proof* of unsatisfiability.

A proof has a tree-like structure, which represents a derivation of the empty clause from the clauses given initially:

```
type proof =
    Input of int
  | Resolve of (prop formula * proof * proof)
```

The proof *Input* i, represents the trivial derivation of the i-th input clause. *Resolve* (l, p, p') represents a resolution step, where two clauses are resolved with the literal l. The proof of the two clauses is given by p and p'.

You can find this type definition in the file `proofcheck.ml` on the course homepage, together with a function

check :: prop formula list list $\rightarrow$ *proof* $\rightarrow$ *bool*

that checks proofs for correctness.

Your job is to extend the function *dp* to a function

dp_proof :: prop formula list list $\rightarrow$ *proof*

which returns a proof when the formula is unsatisfiable. Otherwise it may throw an exception.

Use the checker to test the correctness of your little prover on various examples.

What can you say about the efficiency of the representation of proofs by trees? How could it be improved?

Solution

See code in `ex3.ml`

Recommended setup for this exercise:

```
#use "init.ml";;                          (* load everything *)
let default_parser = parse_prop_formula;; (* set parser for propositional logic *)
#use "proofcheck.ml";;                    (* load the proof checker *)
```