

## Assignment 7: a miniproject

Choose one of the following miniprojects. Project 2 and 3 are more or less open ended. The projects may be done in groups up to three persons. Reports (short) are due by December 17. A satisfactorily completed project is worth 30 points.

### Project 1: Automated Theorem Proving in Lattice Theory

The automated theorem prover **Prover 9** and the counter model constructor **Mace** may be downloaded from [www.cs.unm.edu/~mccune/mace4](http://www.cs.unm.edu/~mccune/mace4). The project consists in solving some problems in lattice theory using these tools.

Recall that a *partially ordered set*  $P = (P, \leq)$  consists of a set  $P$  with a binary relation  $\leq$  which is reflexive, transitive and antisymmetric.  $P$  is a *linearly ordered set* if in addition satisfies

$$x \leq y \vee y \leq x$$

for all  $x, y$ .

A *lattice*  $A = (A, \leq, \vee, \wedge, 0, 1)$  is a structure consisting of a partially ordered set  $(A, \leq)$ , and where 0 and 1 are elements of  $A$  and  $\vee$  and  $\wedge$  are two binary operations on  $A$  satisfying

- (a)  $0 \leq x$ ,
- (b)  $x \vee y \leq z$  if and only if  $x \leq z$  and  $y \leq z$ ,
- (c)  $x \leq 1$ ,
- (d)  $z \leq x \wedge y$  if and only if  $z \leq x$  and  $z \leq y$ .

The lattice is *distributive* if it satisfies

$$\text{(dist)} \quad x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

A *Heyting algebra* is a structure  $H = (H, \leq, \wedge, \vee, \rightarrow, 0, 1)$  where  $(H, \leq, \wedge, \vee, 0, 1)$  forms a lattice and  $(\rightarrow)$  is a binary operation on  $H$  such that

$$\text{(imp)} \quad x \wedge y \leq z \iff x \leq (y \rightarrow z).$$

A *Boolean algebra* is a structure  $H = (H, \leq, \wedge, \vee, \neg, 0, 1)$  where  $(H, \leq, \wedge, \vee, 0, 1)$  forms a distributive lattice and  $(\neg)$  is an operation on  $H$  (the *complement operation*) such that

$$\text{(comp)} \quad x \vee \neg x = 1 \quad x \wedge \neg x = 0.$$

The laws of a Heyting algebra are closely connected to intuitionistic propositional logic, just as those of a Boolean algebra are connected to those of classical propositional logic.

## The Problems

The assignment consists in either finding proofs of the following statements using the automatic theorem prover **Prover9** or to find counter examples to them using **Mace4**. Some of the problems might be too hard for the machine, and you may have to find ways to guide its search. See the on-line documentation.

1. Decide: Is every lattice distributive?
2. Prove: Every lattice satisfying

$$x \leq y \vee z \implies x \leq y \vee x \leq z$$

is linearly ordered.

3. Prove: Every Heyting algebra is a distributive lattice.
4. Prove: In a distributive lattice  $L$  complements are unique in the following sense: If  $a, b, c \in L$ , then there exists at most one  $x \in L$  with

$$x \wedge a = b \quad x \vee a = c.$$

Is this true if the distributivity condition is dropped?

5. Prove: Every Boolean algebra becomes a Heyting algebra when letting  $a \rightarrow b = \neg a \vee b$ .
6. Prove: Every linearly ordered lattice is a Heyting algebra.
7. Prove: Every linearly ordered lattice is distributive.
8. Determine the Boolean algebras which are linearly ordered.

## Examples of use

Suppose we want to investigate whether a distributive lattice  $L$  always have complements, i.e. whether

$$(*) \quad (\forall x)(\exists y)(x \vee y = 1 \ \& \ x \wedge y = 0).$$

We then make a text file as containing the axioms for a distributive lattice written in the Otter/Prover9 syntax. The formula (\*) is then written down as a goal formula. See below.

Assume the text file is called `example1.in`. To try prove (\*) we write

```
prover9 -f example1.in > example1.out
```

To try to find a counter example to (\*) write

```
mace4 -c -f example1.in > example1.out
```

File `example1.in`:

```
assign(max_seconds, 30). % limit of search tume
```

```
formulas(sos).
```

```
% axioms for a partial order
```

```
x <= x.  
x <= y & y <= x -> x=y.  
x <= y & y <= z -> x <= z.
```

```
% lattice axioms
```

```
x <= 1.  
z <= x ^ y <-> z <= x & z <= y.
```

```
0 <= x.  
x v y <= z <-> x <= z & y <= z.
```

```
% distributivity
```

```
(x v y) ^ z = (x^z) v (y^z).
```

```
end_of_list.
```

```

formulas(goals).

% do complements exist?

all x exists y (x v y = 1 & x ^ y = 0).

end_of_list.

```

We see that mace4 gives a counter model to the statement. The file example1.out contains the following structure.

```

===== MODEL =====

interpretation( 3, [number=1, seconds=0], [

    function(c1, [ 2 ]),

    function(^(_,_), [
0, 0, 0,
0, 1, 2,
0, 2, 2 ]),

    function(v(_,_), [
0, 1, 2,
1, 1, 1,
2, 1, 2 ]),

    relation(<=(_,_), [
1, 1, 1,
0, 1, 0,
0, 1, 1 ]))
]).

===== end of model =====

```

The counter example is a lattice consisting of the elements  $\{0, 1, 2\}$  and ordered as  $0 < 2 < 1$ .

Suppose next that we want to prove that  $x \wedge y = y \wedge x$ , commutativity of the meet operation, follows from the axioms of a distributive lattice. We then replace the goal formula above with

```
% commutativity of meet
```

```
x ^ y = y ^ x.
```

The resulting file `example2.in` is fed to `prover9`. The resulting output contains:

```
===== PROOF =====

% Proof 1 at 0.02 (+ 0.00) seconds.
% Length of proof is 14.
% Level of proof is 5.
% Maximum clause weight is 11.
% Given clauses 46.

1 x <= y & y <= x -> x = y # label(non_clause). [assumption].
3 z <= x ^ y <-> z <= x & z <= y # label(non_clause). [assumption].
5 x ^ y = y ^ x # label(non_clause) # label(goal). [goal].
6 x <= x. [assumption].
7 -(x <= y) | -(y <= x) | y = x. [clausify(1)].
10 -(x <= y ^ z) | x <= y. [clausify(3)].
11 -(x <= y ^ z) | x <= z. [clausify(3)].
12 x <= y ^ z | -(x <= y) | -(x <= z). [clausify(3)].
19 c2 ^ c1 != c1 ^ c2. [deny(5)].
20 x ^ y <= x. [hyper(10,a,6,a)].
21 x ^ y <= y. [hyper(11,a,6,a)].
57 x ^ y <= y ^ x. [hyper(12,b,21,a,c,20,a)].
484 x ^ y = y ^ x. [hyper(7,a,57,a,b,57,a)].
485 $F. [resolve(484,a,19,a)].

===== end of proof =====
```

Only a subset of the lattice axioms are used in the proof above: 1,3,6. The goal is denied and everything is put on Skolem normal form (clausified): 7,10,11,12,19. Note the Skolem constants `c1` and `c2`. A contradiction (`$F`) is derived using resolution with factorisation (hyper resolution).

## Project 2: Modelling Sokoban

The old computer game Sokoban can be regarded as a problem of motion planning. The game pictures a one floor warehouse with a number large boxes. The keeper of the warehouse (controlled by the player) is required to move all the boxes to certain goals in the rooms. This is complicated by the fact that the boxes can only be pushed, not dragged or lifted. The directions of the pushes are limited to north, east, west and south and are always a unit's length (= length of the boxes). Moreover, the keeper must stand on a free square and of course push the box into a free square.

We may model the set up of the game as follows. The grid of possible positions (of boxes, keeper) is modelled by the integer points of the Euclidean plane, i.e. an element of the set  $G = \mathbb{Z} \times \mathbb{Z}$ . The walls are specified by a boolean-valued function indicating whether this a particular square is filled with bricks:

$$W : G \rightarrow \text{Bool.}$$

The goals are likewise indicated by a boolean valued function

$$D : G \rightarrow \text{Bool.}$$

The state of the game is given by specifying the position of the keeper  $k \in G$  and the positions of the boxes  $b_1, \dots, b_n \in G$ . Some obvious consistency conditions on the state of the game model that should be true at any time are the following

- (C1) The keeper and the boxes are never inside a wall.
- (C2) The positions of all boxes must be distinct.
- (C3) The keeper's position is distinct from the boxes' positions.

The different directions that the keeper can try to move in are  $A = \{\mathbf{n}, \mathbf{s}, \mathbf{e}, \mathbf{w}\}$  and these change the coordinates as indicated by the function  $m : A \times G \rightarrow G$

$$\begin{aligned}m(\mathbf{n}, (x, y)) &= (x, y + 1) \\m(\mathbf{s}, (x, y)) &= (x, y - 1) \\m(\mathbf{e}, (x, y)) &= (x + 1, y) \\m(\mathbf{w}, (x, y)) &= (x - 1, y)\end{aligned}$$

The action  $a \in A$  of the keeper at position  $k$  on a box at position  $b$  is given by the function  $f : A \times G \times G \rightarrow G$  defined by  $f(a, k, b) = m(a, b)$  if  $m(a, k) = b$  and  $= b$  otherwise. Note  $m(a, k) = b$  holds precisely when  $k$  can push  $b$  with action  $a$ .

A move  $a \in A$  is *legal* at a consistent state  $(k, b_1, \dots, b_n)$  if also the resulting next state

$$\delta(a, k, b_1, \dots, b_n) =_{\text{def}} (m(a, k), f(a, k, b_1), \dots, f(a, k, b_n))$$

is consistent. A consistent state at which there is a legal move is called *non-blocked*.

The game is solved when every box has reached a goal position.

A transition system may now be obtained from these data. Let  $S = \{(k, \vec{b}) \in G \times G^n : (k, \vec{b}) \text{ is non-blocked}\}$ . Define a transition relation  $\rightarrow$  on  $S$  by  $(k, \vec{b}) \rightarrow (\ell, \vec{c})$  iff there is  $a \in A$  with  $\delta(a, k, \vec{b}) = (\ell, \vec{c})$ . Note that the relation  $\rightarrow$  is total on  $S$ .

Alternatives:

- Formalize the notion of Sokoban model in Coq and formulate the property of solvable and "stuck" game (i.e. some boxes cannot be moved to any goal area).
- Formalize a very simple Sokoban game in NuSMV and investigate some properties of it using model checking, e.g. solvability. You may have to replace  $\mathbb{Z}$  with a finite set  $\mathbb{Z}_n = \{0, \dots, n-1\}$  and use modulo arithmetic for defining  $m : A \times G \rightarrow G$ .

Some references

1. The Sokoban Homepage: <http://webdocs.cs.ualberta.ca/~games/Sokoban/>
2. Gihwon Kwon and Taehoon Lee: Solving Box-Pushing Games via Model Checking with Optimizations. *Automated technology for verification and analysis*. Lecture Notes in Computer Science, 2004, Volume 3299/2004, 491-494.
3. Coqoban - implementation of Sokoban in Coq by Jasper Stein.  
<http://coq.inria.fr/V8.2p11/contribs/Coqoban.html>

## **Project 3: make your own project.**

This should be a project that is amenable to the methods introduced in the course. The problems may concern mathematics, computer science or puzzles. Remember to provide references and credit to other work done.

---