Logik für Informatiker Logic for computer scientists

Tools for propositional logic

Till Mossakowski

WiSe 2009/10

Till Mossakowski Logic

For each propositional sentence, there is an equivalent sentence of form

 $(\varphi_{1,1} \vee \ldots \vee \varphi_{1,m_1}) \wedge \ldots \wedge (\varphi_{n,1} \vee \ldots \vee \varphi_{n,m_n})$

where the $\varphi_{i,j}$ are $\mathit{literals},$ i.e. atomic sentences or negations of atomic sentences.

A sentence in CNF is called a *Horn sentence*, if each disjunction of literals contains *at most one positive literal*.

 $\neg Home(claire) \land (\neg Home(max) \lor Happy(carl))$ $Home(claire) \land Home(max) \land \neg Home(carl)$ $Home(claire) \lor \neg Home(max) \lor \neg Home(carl)$ $Home(claire) \land Home(max) \land$ $(\neg Home(max) \lor \neg Home(max))$

 \neg Home(claire) \land (Home(max) \lor Happy(carl)) (Home(claire) \lor Home(max) $\lor \neg$ Happy(claire)) \land Happy(carl)

 $Home(claire) \lor (Home(max) \lor \neg Home(carl)$

Alternative notation for the conjuncts in Horn sentences

$$\neg A_1 \lor \ldots \lor \neg A_n \lor B \qquad (A_1 \land \ldots \land A_n) \to B$$
$$\neg A_1 \lor \ldots \lor \neg A_n \qquad (A_1 \land \ldots \land A_n) \to \bot$$
$$B \qquad \qquad \top \to B$$
$$\bot \qquad \qquad \Box$$

Any Horn sentence is equivalent to a conjunction of conditional statements of the above four forms.

Satisfaction algorithm for Horn sentences

- **1** For any conjunct $\top \rightarrow B$, assign TRUE to B.
- **2** If for some conjunct $(A_1 \land \ldots \land A_n) \rightarrow B$, you have assigned TRUE to A_1, \ldots, A_n then assign TRUE to B.
- Seperat step 2 as often as possible.
- If there is some conjunct (A₁ ∧ ... ∧ A_n) → ⊥ with TRUE assigned to A₁,..., A_n, the Horn sentence is not satisfiable. Otherwise, assigning FALSE to the yet unassigned atomic sentences makes all the conditionals (and hence also the Horn sentence) true.

Theorem The algorithm for the satisfiability of Horn sentences is correct, in that it classifies as tt-satisfiable exactly the tt-satisfiable Horn sentences.

To ask whether this database entails B, Prolog adds $\perp \leftarrow B$ and runs the Horn algorithm. If the algorithm fails, Prolog answers "yes", otherwise "no".

A *clause* is a finite set of literals. Examples:

$$C_1 = \{Small(a), Cube(a), BackOf(b, a)\}$$

$$C_2 = \{Small(a), Cube(b)\}$$

$$C_3 = \emptyset \text{ (also written }\Box\text{)}$$

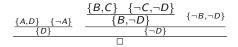
Any set \mathcal{T} of sentences in CNF can be replaced by an equivalent set \mathcal{S} of clauses: each conjunct leads to a clause.

A clause R is a *resolvent* of clauses C_1, C_2 if there is an atomic sentence A with $A \in C_1$ and $(\neg A) \in C_2$, such that

 $R = C_1 \cup C_2 \setminus \{A, \neg A\}.$

Resolution algorithm: Given a set S of clauses, systematically add resolvents. If you add \Box at some point, then S is not satisfiable. Otherwise, it is satisfiable.

We start with the CNF sentence: $\neg A \land (B \lor C \lor B) \land (\neg C \lor \neg D) \land (A \lor D) \land (\neg B \lor \neg D)$ In Clause form: $\{\neg A\}, \{B, C\}, \{\neg C, \neg D\}, \{A, D\}, \{\neg B, \neg D\}$ Apply resolution:



Theorem Resolution is sound and complete. That is, given a set S of clauses, it is possible to arrive at \Box by successive resolutions if and only if S is not satisfiable.

This gives us an alternative sound and complete proof calculus by putting

$\mathcal{T}\vdash S$

iff with resolution, we can obtain \Box from the clausal form of $\mathcal{T} \cup \{\neg S\}$.

Davis-Putnam-Logemann-Loveland algorithm

- *backtracking* algorithm:
 - select a literal,
 - assign a truth value to it,
 - simplify the formula,
 - recursively check if the simplified formula is satisfiable
 - if this is the case, the original formula is satisfiable;
 - otherwise, do the recursive check with the opposite truth value.
- Implementations: mChaff, zChaff, darwin
- Crucial: design of the literal selection function

- If a clause is a *unit clause*, i.e. it contains only a single unassigned literal, this clause can only be satisfied by assigning the necessary value to make this literal true ⇒ reduction of search space
- Pure literal elimination: If a propositional variable occurs with only one polarity in the formula, it is called *pure* ⇒ the assignment is clear

function DPLL(Φ) if Φ is a consistent set of literals then return true; if Φ contains an empty clause then return false; for every unit clause I in Φ Φ =unit-propagate(I, Φ); for every literal I that occurs pure in Φ Φ =pure-literal-assign(I, Φ); I := select-literal(Φ); return DPLL($\Phi \land I$) OR DPLL($\Phi \land not(I)$);

```
• nice syntax for propositional logic
```

```
logic Propositional
spec Props =
    props A,B,C
    A
    not (A /\ B)
    C => B
    not C %implied
```

end

- Reads and checks CASL specifications
- Can prove %implied sentences using resolution provers and SAT solvers
 - use "Prove" menu of a node
- Can find models of sets of sentences using DPLL
 - use "Check consistency" menu of a node, select darwin
- available at http://www.dfki.de/sks/hets.
 - availabel for Linux
 - Windows users: use the live CD