

Logik für Informatiker  
Logic for computer scientists  
Tools for propositional logic

Till Mossakowski, Lutz Schröder

WiSe 2011/12

# Recall: Conjunctive Normal Form (CNF)

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

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

where the  $\varphi_{i,j}$  are *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*.

# Examples of Horn sentences

$\neg \text{Home}(\text{claire}) \wedge (\neg \text{Home}(\text{max}) \vee \text{Happy}(\text{carl}))$

$\text{Home}(\text{claire}) \wedge \text{Home}(\text{max}) \wedge \neg \text{Home}(\text{carl})$

$\text{Home}(\text{claire}) \vee \neg \text{Home}(\text{max}) \vee \neg \text{Home}(\text{carl})$

$\text{Home}(\text{claire}) \wedge \text{Home}(\text{max}) \wedge$   
 $(\neg \text{Home}(\text{max}) \vee \neg \text{Home}(\text{max}))$

# Examples of non-Horn sentences

$\neg \text{Home}(\text{claire}) \wedge (\text{Home}(\text{max}) \vee \text{Happy}(\text{carl}))$

$(\text{Home}(\text{claire}) \vee \text{Home}(\text{max}) \vee \neg \text{Happy}(\text{claire}))$   
 $\wedge \text{Happy}(\text{carl})$

$\text{Home}(\text{claire}) \vee (\text{Home}(\text{max}) \vee \neg \text{Home}(\text{carl}))$

# Alternative notation for the conjuncts in Horn sentences

$$\neg A_1 \vee \dots \vee \neg A_n \vee B$$

$$(A_1 \wedge \dots \wedge A_n) \rightarrow B$$

$$\neg A_1 \vee \dots \vee \neg A_n$$

$$(A_1 \wedge \dots \wedge A_n) \rightarrow \perp$$

$$B$$

$$\top \rightarrow B$$

$$\perp$$

$$\square$$

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  $(A_1 \wedge \dots \wedge A_n) \rightarrow B$ , you have assigned `TRUE` to  $A_1, \dots, A_n$  then assign `TRUE` to  $B$ .
- 3 Repeat step 2 as often as possible.
- 4 If there is some conjunct  $(A_1 \wedge \dots \wedge A_n) \rightarrow \perp$  with `TRUE` assigned to  $A_1, \dots, A_n$ , 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.

# Correctness of the satisfaction algorithm

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

*AncestorOf(a, b) : -MotherOf(a, b).*

*AncestorOf(b, c) : -MotherOf(b, c).*

*AncestorOf(a, b) : -FatherOf(a, b).*

*AncestorOf(b, c) : -FatherOf(b, c).*

*AncestorOf(a, c) : -AncestorOf(a, b), AncestorOf(b, c).*

*MotherOf(a, b).      FatherOf(b, c).      FatherOf(b, d).*

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:

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

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

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

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  $\mathcal{S}$  of clauses, systematically add resolvents. If you add  $\square$  at some point, then  $\mathcal{S}$  is not satisfiable. Otherwise (i.e. if no further resolution steps are possible and  $\square$  has not been added), it is satisfiable.

# Example

We start with the CNF sentence:

$$\neg A \wedge (B \vee C \vee B) \wedge (\neg C \vee \neg D) \wedge (A \vee D) \wedge (\neg B \vee \neg D)$$

In Clause form:

$$\{\neg A\}, \{B, C\}, \{\neg C, \neg D\}, \{A, D\}, \{\neg B, \neg D\}$$

Apply resolution:

$$\frac{\frac{\frac{\{A, D\} \quad \{\neg A\}}{\{D\}} \quad \frac{\frac{\{B, C\} \quad \{\neg C, \neg D\}}{\{B, \neg D\}} \quad \{\neg B, \neg D\}}{\{\neg D\}}}{\square}}$$

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

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

$$\mathcal{T} \vdash \mathcal{S}$$

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

## 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  $\Rightarrow$  reduction of search space
- *Pure literal elimination*: If a propositional variable occurs with only one polarity in the formula, it is called *pure*  $\Rightarrow$  the assignment is clear

# DPLL in pseudo code

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

- nice syntax for propositional logic

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



# Heterogeneous Tool Set

- 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>.
  - available for Linux
  - Windows users: use the live CD