Formale Methoden der Softwaretechnik
Formal methods of software engineering

Till Mossakowski, Christoph Lüth

SoSe 2011

Propositional Logic

- at the core of many logics, formalisms, programming languages
- used as kind of assembly language for coding problems
- available tools:
  - Boole — learning about truth tables
  - Tarski's world — Henkin-Hintikka game
  - Fitch — natural deduction proofs
  - SPASS — resolution proofs
  - Jitpro — tableau proofs
  - minisat, zChaff — SAT solvers using DPLL
  - Hets — friendly interface to SAT solvers and SPASS

Recall: Conjunctive Normal Form (CNF)

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 *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

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

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

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

$$Home(claire) \wedge Home(max) \wedge \\ (\neg Home(max) \vee \neg Home(max))$$

Examples of non-Horn sentences

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

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

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

Alternative notation for the conjuncts in Horn sentences

$\neg A_1 \vee \ldots \vee \neg A_n \vee B$	$(A_1 \wedge \ldots \wedge A_n) \to B$
$\neg A_1 \vee \ldots \vee \neg A_n$	$(A_1 \wedge \ldots \wedge A_n) \to \bot$
B	$\top \to B$
$\bot$	$\Box$

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 $\top \to B$, assign TRUE to B.
2. If for some conjunct $(A_1 \wedge \ldots \wedge A_n) \to B$, you have assigned TRUE to $A_1, \ldots, A_n$ then assign TRUE to B.
3. Repeat step 2 as often as possible.
4. If there is some conjunct $(A_1 \wedge \ldots \wedge A_n) \to \bot$ with TRUE assigned to $A_1, \ldots, A_n$, 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.

Propositional Prolog

$$
\begin{array}{l}
AncestorOf(a,b) :- MotherOf(a,b). \\
AncestorOf(b,c) :- MotherOf(b,c). \\
AncestorOf(a,b) :- FatherOf(a,b). \\
AncestorOf(b,c) :- FatherOf(b,c). \\
AncestorOf(a,c) :- AncestorOf(a,b), AncestorOf(b,c). \\
MotherOf(a,b). \qquad FatherOf(b,c). \qquad FatherOf(b,d).
\end{array}
$$

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

Clauses

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

$$C_1 = \{Small(a), Cube(a), BackOf(b, a)\}$$
$$C_2 = \{Small(a), Cube(b)\}$$
$$C_3 = \emptyset \quad (\text{ also written } \Box)$$

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.

Resolution

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 $\Box$ at some point, then $\mathcal{S}$ is not satisfiable. Otherwise, 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:

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

Soundness and completeness

Theorem Resolution is sound and complete. That is, given a set $\mathcal{S}$ of clauses, it is possible to arrive at $\Box$ by successive resolutions if and only if $\mathcal{S}$ is not satisfiable.
This gives us an alternative sound and complete proof calculus by putting

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

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

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 (see below)
  - use "Check consistency" menu of a node, select *darwin*
- available at `http://www.dfki.de/sks/hets`.
  - available for Linux
  - use the virtual machine (see homepage)

Common Algebraic Specification Language

- nice syntax for propositional logic

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

SAT solving

Davis-Putnam-Logemann-Loveland (DPLL) 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, minisat
- Crucial: design of the literal selection function

Optimizations in DPLL

- 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(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 l in Φ
        Φ=unit-propagate(l, Φ);
    for every literal l that occurs pure in Φ
        Φ=pure-literal-assign(l, Φ);
    l := select-literal(Φ);
    return DPLL(Φ∧l) OR DPLL(Φ∧not(l));
```

Tableau provers

- checks unsatisfiability
- break complex formulas into simpler ones
- nodes of the same branch = conjunction
- different branches = disjunction
- a conjunction is split into the conjuncts, added to is branch
- a disjunction splits the branch into two
- a branch is closed of it contains a literal and its negation
- Jitpro: `http://ps.uni-sb.de/jitpro/prover.php`