Lecture notes 11.1

Propositional logic: resolution

COMP 2411, session 1, 2004

Introduction

Though semantic tableaux provide a relatively efficient proof method, they do not scale up to the predicate calculus, which is the logical language we will eventually be interested in.

Hilbert-style proof systems and Natural deduction have to explore a bigger search space, which makes them unsuitable for mechanical theorem proving, despite their good features.

Resolution is a relatively efficient method that can be extended to the predicate calculus.

It is all the more interesting that Prolog uses a refinement of the Resolution method.

Conjunctive normal form (1)

Contrary to the previous proof systems, Resolution cannot process any kind of formula, but just disjunctions of literals.

The following are examples of disjunctions of literals.

- p
- $p \vee \neg p$
- $\neg p \vee \neg q \vee \neg r$
- $p \vee \neg q \vee \neg r \vee s \vee \neg r \vee \neg r$

This is not a limitation because every propositional formula φ can be put into conjunctive normal form (CNF): φ is logically equivalent to a conjunction of disjunctions of literals.

Conjunctive normal form (2)

To put a formula into conjunctive normal for, it suffices to:

1. eliminate all boolean operators except negation, disjunction and conjunction;
2. pull all negations inwards using De Morgan's laws;
3. eliminate double negations;
4. distribute disjunction over conjunction.

The previous steps applied to $(\neg p \rightarrow \neg q) \rightarrow \neg(p \wedge \neg q)$ yield:

1. $\neg(\neg\neg p \vee \neg q) \vee \neg(p \wedge \neg q)$
2. $(\neg\neg\neg p \wedge \neg\neg q) \vee (\neg p \vee \neg\neg q)$
3. $(\neg p \wedge q) \vee (\neg p \vee q)$
4. $(\neg p \vee \neg p \vee q) \wedge (q \vee \neg p \vee q)$

Clauses

A disjunction of literals can be represented as a set of literals, called a clause.

- Identical literals in a disjunction is literals are obviously reduced to one representative in the associated clause. *E.g.*, the clause associated with $q \vee \neg p \vee q$ is $\{q, \neg p\}$.
- The empty clause is a particular clause, that does not corresponding to any disjunction of literals, but to a contradiction (like $p \wedge \neg p$). It is denoted $\Box$.
- A unit clause is a clause consisting of a single literal.
- Alternative notation for clauses is sometimes used. For instance, $p\bar{q}\bar{r}$ is an alternative notation for $\{p, \neg q, \neg r\}$.
- A conjunctive normal form is sometimes represented as a set of clauses. *E.g.*, the CNF on p. 4, line 4, can be represented as $\{\{\neg p, q\}, \{q, \neg p\}\}$.

Satisfiability of clauses/sets of clauses

A clause/set of clauses is satisfiable iff there exists an assignment (of truth values to the atomic formulas) that makes the formula in CNF associated with the clause/set of clauses true.

A clause/set of clauses is valid iff all assignments make the associated formula true.

For instance, $\{\{p, q, r\}, \{\neg p, q\}, \{\neg q, \neg r\}, \{r\}\}$ is satisfiable, thanks to the assignment that maps p to false, q to false, and r to true.

The empty clause is not satisfiable (it is not possible to make any member of $\Box$ true since $\Box$ has no member.

The empty set of clauses is valid (all members of $\emptyset$ are always true since $\emptyset$ has no member).

The resolution rule

Given three clauses C_1 and C_2 and C, the Resolution rule infers C (conclusion/ resolvent) from C_1 and C_2 (premises/parent clauses) iff there exists an atom φ such that:

- φ belongs to C_1, $\neg\varphi$ belongs to C_2, and $C = (C_1 \setminus \{\varphi\}) \cup (C_2 \setminus \{\neg\varphi\})$, or
- φ belongs to C_2, $\neg\varphi$ belongs to C_1, and $C = (C_1 \setminus \{\neg\varphi\}) \cup (C_2 \setminus \{\varphi\})$.

For example, from $C_1 = \{a, b, \neg c\}$ and $C_2 = \{b, c, \neg e\}$, and taking $\varphi = c$, the resolution rule derives $C = \{a, b, \neg e\}$.

Property: The Resolution rule can be applied in more than one way to the same pair of clauses C_1 and C_2 iff any resolvent of C_1, C_2 is valid.

Key property of resolution rule

Property: For all clauses C_1, C_2, C such that C is a resolvent of C_1, C_2, $C_1 \cup C_2$ is satisfiable iff C is satisfiable; moreover, any model of $C_1 \cup C_2$ is a model of C.

Take C_1, C_2, C as in previous page. Let ν be an assignment that makes $C = \{a, b, \neg e\}$ true, hence $\nu(a) = T$ or $\nu(b) = T$ or $\nu(e) = F$. Suppose that $\nu(a) = T$. Note that $a \in C_1$ and $c \in C_2$ and neither c nor $\neg c$ belongs to C. Put $\nu^\star(\varphi) = \nu(\varphi)$ for all atoms φ distinct from c, and $\nu^\star(c) = T$. Clearly, $\nu^\star$ is a model of $C_1 \cup C_2$.

Conversely, let ν be an assignment that makes both C_1 and C_2 true. Suppose that $\nu(c) = T$. Since $\neg c \in C_1$ it follows that ν is a model of $C_1 \setminus \{\neg c\}$. Since $C_1 \setminus \{\neg c\} \subseteq C$, ν is also a model of C.

Proof by resolution

Starting from a finite set S of clauses, only finitely many clauses can be iteratively generated using the Resolution rule.

Moreover, the (unsatisfiable) empty clause $\Box$ is generated iff S is unsatisfiable.

More precisely, the Resolution algorithm takes as argument a set of clauses S, puts $S_0 = S$ and inductively defines S_{i+1} from S_i as the union of S_i with the set of all resolvents of pairs of clauses in S_i.

Let $n \in \mathbb{N}$ be such that $S_{n+1} = S_n$ (no new clause can be generated). Then S is unsatisfiable iff $\Box \notin S_n$.

Examples (1)

`[p] [q, ~p] [~r] [r, ~p, ~q]` is unsatisfiable, since the empty clause can be generated:

```
[p]  [q, ~p]  [~r]  [r, ~p, ~q]
[q]  [p]  [q, ~p]  [~r]  [r, ~p, ~q]
[r, ~p]  [q]  [p]  [q, ~p]  [~r]  [r, ~p, ~q]
[r]  [r, ~p]  [q]  [p]  [q, ~p]  [~r]  [r, ~p, ~q]
[]  [r]  [r, ~p]  [q]  [p]  [q, ~p]  [~r]  [r, ~p, ~q]
```

Note that the above output of the program `resolution_tests` does not generate $S_1, S_2, \ldots$ as defined on previous slide, but (equivalently) generates one new clause at each step, until either $\Box$ (`[]`) is generated or no more clause can be generated.

Examples (2)

The program does not guarantee that the shortest derivation of $\Box$ is obtained. Different orderings of clauses might yield shorter proofs:

```
[r, ~p, ~q]  [q, ~p]  [p]  [~r]
[r, ~p]  [r, ~p, ~q]  [q, ~p]  [p]  [~r]
[r]  [r, ~p]  [r, ~p, ~q]  [q, ~p]  [p]  [~r]
[]  [r]  [r, ~p]  [r, ~p, ~q]  [q, ~p]  [p]  [~r]
```

Since `[p] [q, ~p] [~r] [r, ~p, ~q]` is a set of clauses associated with the formula $\varphi = \neg((p{\rightarrow}q{\rightarrow}r){\rightarrow}(p{\rightarrow}q){\rightarrow}p{\rightarrow}r)$, what has been shown is that φ is not satisfiable, hence that $(p{\rightarrow}q{\rightarrow}r){\rightarrow}(p{\rightarrow}q){\rightarrow}p{\rightarrow}r$ is valid.

Examples (3)

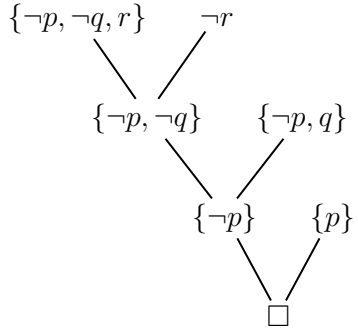
Like semantic tableaux, Resolution is a top-down proof procedure that tries to generate a contradiction.

For another example, `[q, ~p] [q, ~p]` make up a satisfiable set of clauses, since no new clause can be generated from it.

`resolution_tests` also gives examples of formulas that are obviously valid because one of their CNF equivalent yields an empty set of clauses, obtaining by removing valid clauses (containing both an atom and its negation).

Resolution trees

Derivations of $\square$ from a set S of clauses can be represented by a derivation tree, whose root is labeled with $\square$, and whose leaves are labeled with the members of S used in the derivation:



Soundness and completeness

Like all the proof procedures we have studied, Resolution is a sound and complete proof procedure.
Soundness: If $\square$ is derived from a set S of clauses by derivation, then S is unsatisfiable.

The proof of the contrapositive is by induction on heights of trees using the second part of the property on page 8.

Completeness: If a set S of clauses is unsatisfiable, then the resolution procedure generates $\square$ from S.

See Textbook for proof.