PROPOSITIONAL LOGIC (2)

based on

Huth & Ruan
Logic in Computer Science:
Modelling and Reasoning about Systems
Cambridge University Press, 2004

Russell & Norvig
Artificial Intelligence:
A Modern Approach
Prentice Hall, 2010

Clauses

- Clauses are formulas consisting only of $\vee$ and $\neg$

$$p \vee q \vee \neg r$$
$$\neg p \vee \neg q$$

(brackets within a clause are not allowed!)

they can also be written using →, $\vee$(after →) and $\wedge$ (before →)

$$r \to p \vee q$$
$$p \wedge q \to \bot$$
$$\top \to p \vee q$$
$$\top \to \bot$$

Clause without positive literal

Clause without negative literal

Empty clause is considered *false*

an atom or its negation is called a *literal*

Conjunctive & Disjunctive Normal Form

- A formula is in **conjunctive normal form** if it consists of a conjunction of clauses

$$(p \lor q \lor \neg r) \land (p \lor \neg q) \land (p \lor r)$$

$$(r \to p \lor q) \land (q \to p) \land (\top \to p \lor r)$$

  - “conjunction of disjunctions”

- A formula is in **disjunctive normal form** if it consists of a disjunction of conjunctions

$$(p \land q \land \neg r) \lor (p \land \neg q) \lor (p \lor r)$$

Conjunctive & Disjunctive Normal Form

- The transformation from CNF to DNF is exponential

$$
(p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge (p_3 \vee q_3) = \begin{array}{l}
(p_1 \wedge p_2 \wedge p_3) \vee \\
(p_1 \wedge p_2 \wedge q_3) \vee \\
(p_1 \wedge q_2 \wedge p_3) \vee \\
(p_1 \wedge q_2 \wedge q_3) \vee \\
(q_1 \wedge p_2 \wedge p_3) \vee \\
(q_1 \wedge p_2 \wedge q_3) \vee \\
(q_1 \wedge q_2 \wedge p_3) \vee \\
(q_1 \wedge q_2 \wedge q_3)
\end{array}
$$

Conjunctive Normal Form

- Any formula can be written in CNF

$$
\begin{array}{rcl}
(p \vee q \rightarrow r) \vee (q \rightarrow p) & = & \neg(p \vee q) \vee r \vee \neg q \vee p \\
& = & (\neg p \wedge \neg q) \vee r \vee \neg q \vee p \\
& = & (\neg p \vee r \vee \neg q \vee p) \\
& & \wedge(\neg q \vee r \vee \neg q \vee p) \\
& = & (\neg q \vee r \vee p)
\end{array}
$$

(consequently, any formula can also be written in DNF, but the DNF formula may be exponentially larger)

Checking Satisfiability of Formulas in DNF

- Checking DNF satisfiability is easy: process one conjunction at a time; if at least one conjunction is not a contradiction, the formula is satisfiable

  → DNF satisfiability can be decided in polynomial time

$$(p_1 \wedge p_3 \wedge \neg p_3)\vee$$
$$(p_1 \wedge \neg p_2 \wedge \neg p_3)\vee$$
$$(p_1 \wedge \neg p_2 \wedge p_3)\vee$$
$$(\neg p_1 \wedge p_3 \wedge \neg p_3)\vee$$

Conversion to DNF is not feasible in most cases (exponential blowup)

Checking Satisfiability of Formulas in CNF

- No polynomial algorithm is known for checking the satisfiability of arbitrary CNF formulas

Example:

we could use such an algorithm to solve graph coloring with *k* colors

- for each node *i*, create a formula

$$\phi_i = p_{i1} \vee p_{i2} \vee \cdots \vee p_{ik}$$

indicating that each node *i* must have a color

- for each node *i* and different pair of colors c_1 and c_2, create a formula

$$\phi_{ic_1c_2} = \neg(p_{ic_1} \wedge p_{ic_2}) = \neg p_{ic_1} \vee \neg p_{ic_2}$$

indicating a node may not have more than 1 color

- for each edge, create *k* formulas

$$\phi_{ijc} = \neg(p_{ic} \wedge p_{jc}) = \neg p_{ic} \vee \neg p_{jc}$$

indicating that a pair connected nodes *i* and *j* may not both have color *c* at the same time

“At-most-once” constraint

- Let us have variables $x_1, \ldots, x_n$ and require that at most one of these variables is one
- Constraints on the previous slide:

$$(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_n) \wedge \cdots \wedge (\neg x_{n-1} \vee \neg x_n)$$

→ $n(n-1)/2$ clauses in total

- We can do better…

"At-most-once" constraint

- Introduce additional variables $a_1, \ldots a_n$
- Idea: let a_i be true if one of $x_1, \ldots, x_i$ is true
- Formally:

  $\neg a_i \vee \neg x_{i+1}$ (a_i and x_{i+1} may not be true at the same time)

  $\neg a_i \vee a_{i+1}$ (if a_i is true, then a_{i+1} is true)

  $\neg x_i \vee a_i$ (if x_i is true, then a_i is true)

  for all $1 \leq i \leq n-1$
- *3(n-1)* clauses in total!

Resolution Rule

Essential in most satisfiability solvers for CNF formulas is the **resolution rule** for clauses:

Given two clauses $l_1 \vee \cdots \vee l_k$ and $m_1 \vee \cdots \vee m_n$, where $l_1, \ldots, l_k, m_1, \ldots, m_n$ represent literals and it holds that $l_i = \neg m_j$, then it holds that

$$l_1 \vee \cdots \vee l_k, m_1 \vee \cdots \vee \cdots m_n \vdash_R$$
$$l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots m_n$$

Example: $p \vee q \vee \neg r, r \vee s \vdash_R p \vee q \vee s$

$r \rightarrow p \vee q, r \vee s \vdash_R p \vee q \vee s$

Proof for Resolution

on an example

1.	$p \vee q$	premise
2.	$q \rightarrow r$	premise
3.	p	assumption
4.	$p \vee r$	$\vee$i 3
5.	q	assumption
6.	r	→e 2,5
7.	$p \vee r$	$\vee$i 6
8.	$p \vee r$	$\vee$e 1,3-4, 5-7

$\neg q \vee r$

Completeness of Resolution

- If it holds that $C_1, \ldots, C_n \models \bot$ for clauses $C_1, \ldots, C_n$ (i.e. the clauses are a contradiction), then we can derive $\bot$ from $C_1, \ldots, C_n$ by repeated application of the resolution rule

$$
\begin{aligned}
p, p \to q \vee r, q \to \bot, r \to \bot \quad & \vdash_R \quad q \vee r, q \to \bot, r \to \bot \\
& \vdash_R \quad r, r \to \bot \\
& \vdash_R \quad \bot
\end{aligned}
$$

How to find the resolution steps in general?
For some types of clauses it is easier…

Definite clauses & Horn clauses

- A **definite clause** is a clause with exactly one positive literal

$$p, q, p \wedge q \to t$$

- A **horn clause** is a clause with at most one positive literal

$$p, q, p \wedge q \to t, p \wedge q \to \perp$$

A clause with one positive literal is called a **fact**

Forward chaining for Definite clauses

- The **forward chaining algorithm** calculates facts that can be entailed from a set of definite clauses

C = initial set of definite clauses
repeat
    if there is a clause $p_1,\ldots,p_n \rightarrow q$ in ***C*** where $p_1,\ldots,p_n$ are
        facts in ***C*** **then**
        add fact *q* to ***C***
    end if
until no fact could be added
return all facts in ***C***

Resolution

This algorithm is complete for facts: any fact that is entailed, will be derived.

Forward chaining for Horn clauses

- We now also allow to add $\perp$ and other clauses without positive literals to ***C***
- We stop immediately $\perp$ when is found, and return that the set of formulas is contradictory.

$$\mathbf{C}_1 = \{p, p \to q, p \wedge q \to r, r \to \perp\}$$
$$\mathbf{C}_2 = \{p, q, p \to q, p \wedge q \to r, r \to \perp\}$$
$$\mathbf{C}_3 = \{p, q, r, p \to q, p \wedge q \to r, r \to \perp\}$$
$$\mathbf{C}_4 = \{p, q, r, \perp, p \to q, p \wedge q \to r, r \to \perp\}$$

Note:

1) a set of definite clauses is always satisfiable.

2) we can decide in linear time whether a set of Horn clauses is satisfiable.

Deciding entailment for Horn clauses

- Suppose we would like to know whether

  $$C_1, \ldots, C_n \models p_1, \ldots, p_n \rightarrow q$$

  where $C_1, \ldots, C_n$ are Horn clauses; then it suffices to determine whether

  $$C_1, \ldots, C_n, p_1, \ldots, p_n \vdash_R q$$

  (we can show this by means of → introduction)
- As entailment of facts can be decided in linear time, Horn clause entailment can be determined in linear time as well