COMP2411 Lecture 6: Conjunctive Normal Form

Reading: Huth and Ryan, Section 1.5.1—1.5.2

Motivation

So far we have seen two approaches to determining/proving validity of arguments:

1. Truth Tables
   - simple and mechanical,
   - large proofs of validity (n propositions means 2^n rows.)
   - unsuited to human use
2. Natural Deduction
   - suitable for human use
   - often allows short proofs of validity
   - arguably too flexible for machine use: not mechanical, large number of rules, infinite search space

We now consider an approach that has been argued to be more suitable for automated determinations of validity. It also forms part of the basis for logic programming.

The first step is to reduce the complexity of formulae we need to deal with by:

1. reducing the number of operators (to the set $\wedge, \vee, \neg$)
2. restricting to formulae of a very simple syntactic form.

Literal and Clauses

A *literal* is either a propositional atom p or its negation $\neg p$.

A *clause* is a disjunction $l_1 \vee l_2 \vee \ldots \vee l_n$ of one or more literals l_i.

Examples of clauses:

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

Conjunctive Normal Form (CNF)

A formula is in *conjunctive normal form* if it is a conjunction of one or more clauses.

Examples:

- $\neg p$
- $p \vee \neg q$
- $(\neg p \vee q) \wedge (r \vee \neg t \vee \neg p)$
- $(\neg p \vee q) \wedge (r \vee \neg t \vee \neg p) \wedge p$

Testing validity of a formula in CNF is particularly simple:

Theorem:

- A clause $l_1 \vee l_2 \vee \ldots \vee l_n$ is valid iff there exist i, j such that $l_i = \neg l_j$.
- A CNF formula $c_1 \wedge c_2 \wedge \ldots \wedge c_n$ is valid if each of its clauses c_i is valid.

Examples:

- $\neg p \vee q \vee p \vee r$ is valid
- $(\neg p \vee q \vee p) \wedge (r \vee \neg r)$ is valid
- $(\neg p \vee q \vee p) \wedge (r \vee s)$ is not valid

Theorem: For every formula ϕ of propositional logic, there exists an equivalent formula ψ in conjunctive normal form (i.e., a formula ψ such that $\phi \equiv \psi$.)

Proof: we show how to apply the logical equivalences already introduced to convert any given formula to an equivalent one in CNF.

Step 1: Eliminate $\longrightarrow$

Using the rule

$$A \longrightarrow B \;\equiv\; \neg A \vee B$$

we may eliminate all occurrences of $\longrightarrow$.

Example:

$$\begin{aligned} p \longrightarrow ((q \longrightarrow r) \vee \neg s) &\equiv p \longrightarrow ((\neg q \vee r) \vee \neg s) \\ &\equiv \neg p \vee ((\neg q \vee r) \vee \neg s) \end{aligned}$$

Step 2: Push negations down

Using De Morgan's Laws and the double negation rule

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$
$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$
$$\neg\neg A \equiv A$$

we push negations down towards the atoms until we obtain a formula that is formed from literals using only $\wedge$ and $\vee$.

Example:

$$\begin{aligned}
&\neg(\neg p \wedge (q \vee \neg(r \wedge s))) \\
&\quad \equiv \neg\neg p \vee \neg(q \vee \neg(r \wedge s))) \\
&\quad \equiv p \vee (\neg q \vee \neg\neg(r \wedge s)) \\
&\quad \equiv p \vee (\neg q \vee (r \wedge s))
\end{aligned}$$

Step 3: Use distribution to convert to CNF

Using the distribution rules

$$A \vee (B_1 \wedge \ldots \wedge B_n) \equiv (A \vee B_1) \wedge \ldots \wedge (A \vee B_n)$$
$$(B_1 \wedge \ldots \wedge B_n) \vee A \equiv (B_1 \vee A) \wedge \ldots \wedge (B_n \vee A)$$

we obtain a CNF formula.

Example:

$$\begin{aligned}
&(p \wedge q) \vee (p \wedge \neg q) \\
&\quad \equiv ((p \wedge q) \vee p) \wedge ((p \wedge q) \vee \neg q) \\
&\quad \equiv ((p \vee p) \wedge (q \vee p)) \wedge ((p \vee \neg q) \wedge (q \vee \neg q))
\end{aligned}$$

Note: we use distribution from the "tips" of the parse tree, up to the root. Each tip is a literal, which is a CNF formula.

If α and β are already in CNF then

- $\alpha \wedge \beta$ is also in CNF
- $\alpha \vee \beta$ is converted to CNF as follows:
  1. If α and β are literals then $\alpha \vee \beta$ is already in CNF
  2. If $\alpha = \alpha_1 \wedge \ldots \wedge \alpha_1$ then

$$\alpha \vee \beta = (\alpha_1 \vee \beta) \wedge \ldots \wedge (\alpha_1 \vee \beta)$$

     If β is a literal the RHS is in CNF, otherwise $\beta = \beta_1 \wedge \ldots \wedge \beta_k$ and we take one more distribution step to convert each $\alpha_i \wedge \beta_i$ to CNF.

A more complicated example:

$$
\begin{aligned}
&((p \wedge q) \vee (r \wedge s)) \vee (\neg q \wedge (p \vee t)) \\
&\quad \equiv (((p \wedge q) \vee r) \wedge ((p \wedge q) \vee s)) \vee (\neg q \wedge (p \vee t)) \\
&\quad \equiv ((p \vee r) \wedge (q \vee r) \wedge (p \vee s) \wedge (q \vee s)) \vee (\neg q \wedge (p \vee t)) \\
&\quad \equiv \quad ((p \vee r) \vee (\neg q \wedge (p \vee t)) \wedge \\
&\qquad\quad ((q \vee r) \vee (\neg q \wedge (p \vee t)) \wedge \\
&\qquad\quad ((p \vee s) \vee (\neg q \wedge (p \vee t)) \wedge \\
&\qquad\quad ((q \vee s) \vee (\neg q \wedge (p \vee t)) \\
&\quad \equiv \quad (p \vee r \vee \neg q) \wedge (p \vee r \vee p \vee t) \wedge \\
&\qquad\quad (q \vee r \vee \neg q) \wedge (q \vee r \vee p \vee t) \wedge \\
&\qquad\quad (p \vee s) \vee \neg q) \wedge (p \vee s \vee p \vee t) \wedge \\
&\qquad\quad (q \vee s \vee \neg q) \wedge (q \vee s \vee p \vee t)
\end{aligned}
$$

We can apply the fact that every formula can be converted to CNF and the fact that CNF formulas have an easy test for validity as follows:

To determine if $\phi_1, \ldots, \phi_n \models \psi$:

- reduce the problem to testing if $\models \alpha$, where α is the formula $\phi_1 \longrightarrow (\phi_2 \longrightarrow \ldots (\phi_n \longrightarrow \psi) \ldots)$.
- convert α to CNF
- test the CNF formula for validity

An observation about complexity

We have now seen three methods for showing that a formula is valid (if indeed it is):

1. Truth tables: 2^n lines for n propositions
2. Natural Deduction Proofs: could be short, but we have no guarantee of this. The construction used in the proof of completeness simulates truth tables, so uses proofs of size at least 2^n
3. conversion to CNF: it can be shown that this can produce a formula of size 2^n

i.e. all take exponential time!

Question: Is there a method that does not take an exponential amount of time in the worst case?

Answer: Good question! Nobody knows. Solve it and you will immediately win the Turing award, the equivalent in Computer Science of the Nobel prize.

More on this topic in the 4th year theory of computing course, or perhaps in the 3rd year algorithms course ...