Propositional logic: Normal forms

CS242 Formal Specification and Verification

University of Warwick

Autumn term 2006

Semantic equivalence

$$\phi \equiv \psi$$

if and only if

$$\phi \models \psi \text{ and } \psi \models \phi$$

By soundness and completeness, same as provable equivalence, i.e. $\phi \dashv\vdash \psi$.

Exercise 1.5.2.

Adequate set of connectives . . .

. . . is such that, for every formula, there is an equivalent formula with only connectives from that set.

Example: $\{\neg, \vee\}$.

Exercise 1.5.3.

Validity and satisfiability

Validity: $\models \phi$.

Lemma

$$\phi_1, \phi_2, \ldots, \phi_n \models \psi$$

if and only if

$$\models \phi_1 \rightarrow (\phi_2 \rightarrow (\cdots \rightarrow (\phi_n \rightarrow \psi) \cdots))$$

Satisfiability: there exists an assignment of truth values to ϕ's propositional atoms such that ϕ is true.

Proposition

ϕ is satisfiable if and only if $\neg\phi$ is not valid.

Conjunctive normal form

Literal: p or $\neg p$.
CNF:

$$\psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_n$$

such that, for each i, ψ_i is a disjunction of literals.
Some examples:

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

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

$$p \vee q$$

$$p \wedge (\neg p \vee r)$$

$$\top$$

$$\bot \wedge (p \vee q)$$

Validity of CNF

Lemma

A disjunction of literals $L_1 \vee L_2 \vee \cdots \vee L_m$ is valid iff there are i and j such that L_i is $\neg L_j$.

A conjunction $\psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_n$ is valid iff, for all i, ψ_i is valid.

From truth table to CNF

Suppose we have a truth table of ϕ.

For any row in which ϕ is F, form a disjunction as follows: for any propositional atom p, include p if p is F in that line, or $\neg p$ if p is T in that line.

A conjunction of all those disjunctions is a CNF for ϕ.

Example:

$$(p \rightarrow q) \wedge (q \rightarrow p)$$

From CNF to truth table

Example:

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

Conversion to CNF

Specification of `CNF` algorithm:

1. it takes (a parse tree of) a formula ϕ of propositional logic as input and rewrites it to another formula of propositional logic; such rewrites might call the algorithm `CNF` recursively;
2. each computation, or rewrite step, of CNF results in an equivalent formula;
3. `CNF` terminates for all inputs ϕ which are formulas of propositional logic; and
4. the final formula computed by `CNF` is in CNF.

function $\texttt{CNF}(\phi)$:
begin function
  return $\texttt{CNF}'(\texttt{NNF}(\texttt{IMPL\_FREE}(\phi)))$
end function

function $\texttt{CNF}'(\phi)$:
/* precond.: ϕ implication free and in NNF */
/* postcond.: returns an equivalent CNF */
begin function
  case
    ϕ is a literal: **return** ϕ
    ϕ is $\phi_1 \wedge \phi_2$: **return** $\texttt{CNF}'(\phi_1) \wedge \texttt{CNF}'(\phi_2)$
    ϕ is $\phi_1 \vee \phi_2$: **return** $\texttt{DISTR}(\texttt{CNF}'(\phi_1), \texttt{CNF}'(\phi_2))$
  end case
end function

function $\texttt{DISTR}(\eta_1, \eta_2)$:
/* precond.: η_1 and η_2 are in CNF */
/* postcond.: returns a CNF for $\eta_1 \vee \eta_2$ */
begin function
  case
    η_1 is $\eta_{11} \wedge \eta_{12}$:
      return $\texttt{DISTR}(\eta_{11}, \eta_2) \wedge \texttt{DISTR}(\eta_{12}, \eta_2)$
    η_2 is $\eta_{21} \wedge \eta_{22}$:
      return $\texttt{DISTR}(\eta_1, \eta_{21}) \wedge \texttt{DISTR}(\eta_1, \eta_{22})$
    otherwise (= no conjunctions):
      return $\eta_1 \vee \eta_2$
  end case
end function

```
function NNF(φ):
/* precond.: φ is implication free */
/* postcond.: returns a NNF for φ */
begin function
  case
    φ is a literal: return φ
    φ is ¬¬φ1: return NNF(φ1)
    φ is φ1 ∧ φ2: return NNF(φ1) ∧ NNF(φ2)
    φ is φ1 ∨ φ2: return NNF(φ1) ∨ NNF(φ2)
    φ is ¬(φ1 ∧ φ2): return NNF(¬φ1 ∨ ¬φ2)
    φ is ¬(φ1 ∨ φ2): return NNF(¬φ1 ∧ ¬φ2)
  end case
end function
```

Horn formula . . .

. . . is of the form

$$\psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_n$$

such that each ψ_i is of the form

$$p_1 \wedge p_2 \wedge \cdots \wedge p_{k_i} \to q_i$$

where p_1, p_2, . . . , p_{k_i}, q_i are atoms, $\bot$, or $\top$.
We call each ψ_i a *Horn clause*.
Some examples:

$$(p_2 \wedge p_3 \wedge p_5 \to p_{13}) \wedge (\top \to p_5) \wedge (p_5 \wedge p_{11} \to \bot)$$

$$(p \wedge q \wedge s \to \bot) \wedge (\neg q \wedge r \to p) \wedge (\top \to s)$$

$$p_2 \wedge p_3 \wedge p_5 \to p_{13} \wedge p_{27}$$

Exercise 1.5.17.

Deciding satisfiability

function HORN(ϕ):
/* precond.: ϕ is a Horn formula */
/* postcond.: decides satisfiability of ϕ */
begin function
  mark all atoms p where $\top \rightarrow p$
    is a subformula of ϕ;
  while there is a subformula $p_1 \wedge \cdots \wedge p_{k_i} \rightarrow q_i$
    of ϕ such that all p_j are marked but
    q_i is either $\bot$ or an unmarked atom **do**
    if q_i is $\bot$ **then return** 'unsatisfiable'
    else mark q_i for all such subformulas
  end while
  return 'satisfiable'
end function

We assume that, whenever $k_i \geq 2$, all of $p_1, p_2, \ldots, p_{k_i}$ are atoms.

Exercise 1.5.15.

Theorem

The algorithm `HORN` is correct for the satisfiability decision problem of Horn formulas and has no more than n cycles in its while-loop if n is the number of atoms in ϕ. In particular, `HORN` always terminates on correct input.