Clausal Form Logic

1 Introduction

Over the next few lectures we introduce the topics necessary to an understanding of the proof theory at the heart of a lot of knowledge-based systems. It's also the proof theory used by many automated theorem-proving systems, by logic programming languages such as Prolog and by many deductive databases. In this lecture, we introduce a canonical form (a standard form) for wffs.

The canonical representation is called *clausal form*. Using a canonical form we can get a sound and complete proof theory which is amenable to automation. In the worst case, its performance is no better than any other sound & complete proof theory, but it seems to perform well in practice. To be more precise, for CFL, if we use refutation instead of deduction, we can get a proof theory that is sound & complete but has

- only one inference rule, and
- no logical axiom schemata.

Clausal Form Logic is as expressive as FOPL. And, in fact, there's even an algorithm for taking wffs in FOPL and converting them to *clauses* in CFL. But wffs of FOPL and their corresponding clauses in CFL are not necessarily logically equivalent. In other words, the original wff and the corresponding clauses are not necessarily true in exactly the same models.

Instead, the translation preserves a weaker property: satisfiability and unsatisfiability. If the original wff is (un)satisfiable, then the clauses will also be (un)satisfiable. This will turn out to be OK because we will build refutation proofs (proof by contradiction), where finding inconsistency (unsatisfiability) is what matters.

2 The Conversion Algorithm

Here, in summary, is the algorithm for converting an ordinary FOPL wff into a set of clauses in CFL:

1. Relabel clashing variables
2. Eliminate $\Rightarrow$ and $\Leftrightarrow$
3. Move $\neg$ inwards
4. Skolemise existentially quantified variables
5. Drop universal quantifiers
6. Distribute disjunction over conjunction
7. Rewrite as clauses
8. Standardise variables apart

— in that order!

Here's the algorithm again with explanation and an example.

Relabel clashing variables

Rename variables so that each quantifier has a unique variable, i.e. make sure that the same variable is not quantified more than once within the same wff.

$$\forall x \forall y \neg(p(x,y) \Rightarrow \forall y\, q(y,y))$$

becomes

$$\forall x \forall y \neg(p(x,y) \Rightarrow \forall z\, q(z,z))$$

(To avoid having to remember this step, it's a good idea never to write wffs with clashing variables in them!)

Eliminate $\Rightarrow$ and $\Leftrightarrow$

Use:

$$\begin{aligned}(W_1 \Rightarrow W_2) &\equiv (\neg W_1 \vee W_2)\\ (W_1 \Leftrightarrow W_2) &\equiv ((W_1 \Rightarrow W_2) \wedge (W_2 \Rightarrow W_1))\end{aligned}$$

In our example,

$$\forall x \forall y \neg(p(x,y) \Rightarrow \forall z\, q(z,z))$$

becomes

$$\forall x \forall y \neg(\neg p(x,y) \vee \forall z\, q(z,z))$$

Move $\neg$ inwards

Keep moving negation operators until we are only negating atoms.

Use:

$$\begin{aligned}\neg\forall X\, W &\equiv \exists X \neg W\\ \neg\exists X\, W &\equiv \forall X \neg W\\ \neg(W_1 \vee W_2) &\equiv (\neg W_1 \wedge \neg W_2) \text{ (de Morgan)}\\ \neg(W_1 \wedge W_2) &\equiv (\neg W_1 \vee \neg W_2) \text{ (de Morgan)}\\ \neg\neg W &\equiv W\end{aligned}$$

Our example

$$\forall x \forall y \neg(\neg p(x,y) \vee \forall z\, q(z,z))$$

becomes

$$\forall x \forall y (\neg\neg p(x,y) \wedge \neg\forall z\, q(z,z))$$

which then becomes

$$\forall x \forall y (p(x,y) \wedge \exists z \neg q(z,z))$$

Skolemise existentially quantified variables

Replace existentially quantified variables with Skolem terms.

If $\exists Y$ occurs in the wff and it appears within the scope of n ($n \geq 0$) universal quantifiers $\forall X_1 \forall X_2, \ldots, \forall X_n$, then choose a new n-ary function symbol F distinct from all the others in the wff, replace all occurrences of Y with $F(X_1, X_2, \ldots, X_n)$, and strike out the $\exists Y$. For the case where $n = 0$, a 0-ary function is chosen: a 0-ary function is the same as a constant symbol.

Schematically, this is what this step does:

a. $\exists Y \forall X_1, \ldots, \forall X_n p(X_1, \ldots, X_n, Y)$
becomes
$\forall X_1, \ldots, \forall X_n p(X_1, \ldots, X_n, c)$

b. $\forall X_1, \ldots, \forall X_n \exists Y p(X_1, \ldots, X_n, Y)$
becomes
$\forall X_1, \ldots, \forall X_n p(X_1, \ldots, X_n, f(X_1, \ldots, X_n))$

where c is a new constant symbol and f is a new function symbol

Our example:

$$\forall x \forall y (p(x, y) \wedge \exists z \neg q(z, z))$$

becomes

$$\forall x \forall y (p(x, y) \wedge \neg q(f(x, y), f(x, y)))$$

Here's another example:

$$\exists w (p(w) \wedge \forall x \exists y \exists z\, q(w, x, y, z))$$

becomes

$$p(c) \wedge \forall x\, q(c, x, g(x), h(x))$$

Note that this is the step that means that we are only preserving (un)satisfiability. This is the step that is not based on equivalences.

Drop universal quantifiers

Now our wffs only contain universally quantified variables. And because of the renaming of variables done in step 1, all the variables are different. We can drop the quantifiers and have the variables as implicitly universally quantified.

Our example:

$$\forall x \forall y (p(x, y) \wedge \neg q(f(x, y), f(x, y)))$$

becomes

$$p(x, y) \wedge \neg q(f(x, y), f(x, y)))$$

Distribute disjunction over conjunction

Convert the wff to *conjunctive normal form*. A wff is in conjunctive normal form iff it is a conjunction of disjunctions of literals. (Recall that a literal is an atom or a negated atom.). So a wff is in conjunctive normal form if it has the following form:

$$(P_1 \vee P_2 \vee \ldots \vee P_l) \wedge (Q_1 \vee Q_2 \ldots \vee Q_m) \wedge \ldots \wedge (R_1 \vee R_2 \vee \ldots \vee R_n)$$

where each P_i, Q_i and R_i is a literal (an atom or a negated atom).

Our example:

$$p(x, y) \wedge \neg q(f(x, y), f(x, y)$$

is already in conjunctive normal form, so there is no change resulting from this step.

In general, we would use the following:

$$\begin{array}{rcl} (W_1 \vee (W_2 \wedge W_3)) & \equiv & ((W_1 \vee W_2) \wedge (W_1 \vee W_3)) \\ (W_1 \wedge W_2) \vee W_3)) & \equiv & ((W_1 \vee W_3) \wedge (W_2 \vee W_3)) \end{array}$$

Also, associativity can be useful:

$$\begin{array}{rcccl} (W_1 \vee (W_2 \vee W_3)) & \equiv & ((W_1 \vee W_2) \vee W_3) & \equiv & W_1 \vee W_2 \vee W_3 \\ (W_1 \wedge (W_2 \wedge W_3)) & \equiv & ((W_1 \wedge W_2) \wedge W_3) & \equiv & W_1 \wedge W_2 \wedge W_3 \end{array}$$

Here's an example:

$$p \vee (q \wedge r \wedge s \wedge (t \vee u))$$

becomes

$$(p \vee q) \wedge (p \vee r) \wedge (p \vee s) \wedge (p \vee t \vee u)$$

Another example:

$$p \wedge (q \vee r \vee s \vee (t \wedge u))$$

becomes

$$p \wedge (q \vee r \vee s \vee t) \wedge (q \vee r \vee s \vee u)$$

Rewrite as clauses

Split the wff up into bits (one per conjunct). Each bit is referred to as a *clause*.

In general,

$$(P_1 \vee \ldots \vee P_l) \wedge (Q_1 \vee \ldots \vee Q_m) \wedge (R_1 \vee \ldots \vee R_n)$$

becomes

$$P_1 \vee \ldots \vee P_l$$

$$Q_1 \vee \ldots \vee Q_m$$

$$R_1 \vee \ldots \vee Q_n$$

Our example:

$$p(x, y) \wedge \neg q(f(x, y), f(x, y))$$

becomes two clauses

$$p(x, y)$$

$$\neg q(f(x, y), f(x, y))$$

Standardise variables apart

Rename *variables* so that no variable appears in more than one clause. (Note that this applies only to variables.)

Our example:

$$p(x, y)$$

$$\neg q(f(x, y), f(x, y))$$

becomes

$$p(x_1, y_1)$$

$$\neg q(f(x_2, y_2), f(x_2, y_2))$$

3 Clauses

The result of running the algorithm is that the original FOPL wff becomes one or more *clauses*. Separate clauses are implicitly conjoined together. (This follows from step 7 of the algorithm.)

Each clause is a disjunction of literals. Clauses can be thought of as *sets* of literals. As such, the order of the literals is irrelevant, and we can remove duplicates. (The fact that order is irrelevant follows from the fact that $W_1 \vee W_2 \equiv W_2 \vee W_1$; the fact that we can remove duplicates follows from the fact that $W \vee W \equiv W$.)

If a clause contains only one literal, it is called a *unit clause*.

We will sometimes want to write a clause that contains no literals. This is called the *empty clause* and it is written $\Box$.

4 Exercises

1. Convert the following wffs to clausal form.
   (a) $\forall x \forall y (p(x, y) \Rightarrow q(x, y))$
   (b) $\forall x \forall y (\neg q(x, y) \Rightarrow \neg p(x, y))$
   (c) $\forall x \forall y (p(x, y) \Rightarrow (q(x, y) \Rightarrow r(x, y)))$
   (d) $\forall x \forall y ((p(x, y) \wedge q(x, y)) \Rightarrow r(x, y))$
   (e) $\forall x \forall y (p(x, y) \Rightarrow (q(x, y) \vee r(x, y)))$
   (f) $\forall x \forall y (p(x, y) \Rightarrow (q(x, y) \wedge r(x, y)))$
   (g) $\forall x \forall y ((p(x, y) \vee q(x, y)) \Rightarrow r(x, y))$
   (h) $\forall x \exists y (p(x, y) \Rightarrow q(x, y))$
   (i) $\neg \forall x \exists y (p(x, y) \Rightarrow q(x, y))$
   (j) $(\neg \forall x\, p(x)) \Rightarrow (\exists x\, p(x))$
2. In this question, use the following 'key' for the unary predicate symbols s and c and the binary predicate symbol u:

$s(x)$	:	x is a student
$c(x)$	:	x is a computer
$u(x, y)$	:	x uses y

   (a) Convert the following sentence of English into FOPL:

   Every student uses some computer, but at least one (specific) computer is used by every student.

   (b) Convert your FOPL into clausal form.