Worksheet 7

Clausal Form Logic

In this worksheet, we present a *normal form* for first order predicate calculus (FOPC), known as *clausal form logic* (CFL). The aim of the worksheet is that

- you learn how to convert wffs of FOPC into CFL.

7.1 Introduction

Over the next few worksheets we introduce the topics necessary to an understanding of the proof theory at the heart of a lot of automated theorem-proving, logic programming, expert systems and deductive databases. First, in this worksheet, we introduce a normal form (or standard form or canonical form) for wffs, then in subsequent worksheets, we introduce unification, resolution and issues in the search for a proof.

The canonical representation is called *clausal form*. Using clausal form means we need only one inference rule, *resolution*.[1] By cutting the number of inference rules and logical axiom schemata, we get a system that is much more amenable to automation (although don't take this to imply that clausal form theorem proving is efficient *per se*).

7.2 Conversion to Clausal Form

Clausal form logic is a variant of FOPC. Its syntax is equally expressive as that of FOPC but it is canonical. By this we mean that despite the apparent restrictiveness of CFL, for any wff in FOPC there is a corresponding set of

[1] This is only true if we prove using refutation. If not, we need another rule and some logical axiom schemata —see worksheet 10.

clauses in CFL such that the original wff is satisfiable iff the set of clauses is satisfiable. (Note that we are not saying that they are equivalent, i.e. we are not saying that they are true in exactly the same worlds; we are only saying that if the original wff is (un)satisfiable, then the set of clauses will be (un)satisfiable and vice versa. The translation to CFL preserves (un)satisfiability. This turns out to be OK because we will do proof by contradiction, where (un)satisfiability is what matters.)

Here is a procedure for converting an ordinary FOPC wff into a set of clauses.

7.2.1 The Conversion Algorithm

Here's the algorithm in summary:

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

Here it is 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.

$\forall X \forall Y (\mathrm{p}(X,Y) \rightarrow \exists Y \mathrm{q}(Y,Y))$
becomes
$\forall X \forall Y (\mathrm{p}(X,Y) \rightarrow \exists Z \mathrm{q}(Z,Z))$

(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:

$(\alpha \rightarrow \beta) \equiv (\neg\alpha \vee \beta)$
$(\alpha \leftrightarrow \beta) \equiv ((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha))$

$\forall X \forall Y(\mathrm{p}(X, Y) \rightarrow \exists Z \mathrm{q}(Z, Z))$
becomes
$\forall X \forall Y(\neg\mathrm{p}(X, Y) \vee \exists Z \mathrm{q}(Z, Z))$

Move $\neg$ inwards

Keep moving negation operators until we are only negating atoms. Use:

$\neg\forall V \alpha \equiv \exists V \neg\alpha$
$\neg\exists V \alpha \equiv \forall V \neg\alpha$
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ (de Morgan)
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ (de Morgan)
$\neg\neg\alpha \equiv \alpha$

Skolemise existentially quantified variables

Replace existentially quantified variables with Skolem terms.

If $\exists V$ occurs in the wff and n $(n \geq 0)$ universal quantifiers $\forall V_1 \forall V_2, \ldots, \forall V_n$ appear 'before' the $\exists V$ (strictly, if the $\exists V$ is in the scope of the n universal quantifiers), then choose a new n-ary function symbol f distinct from all the others in the wff, replace all occurrences of V with $f(V_1, V_2, \ldots, V_n)$, and strike out the $\exists V$. For the case where $n = 0$, a 0-ary function is chosen: a 0-ary function is the same as a constant.

$\forall X \forall Y(\neg\mathrm{p}(X, Y) \vee \exists Z \mathrm{q}(Z, Z))$
becomes
$\forall X \forall Y(\neg\mathrm{p}(X, Y) \vee \mathrm{q}(\mathrm{h}(X, Y), \mathrm{h}(X, Y)))$

(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 we've only got 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.

$\forall X \forall Y(\neg \mathrm{p}(X,Y) \vee \mathrm{q}(\mathrm{h}(X,Y), \mathrm{h}(X,Y)))$
becomes
$(\neg \mathrm{p}(X,Y) \vee \mathrm{q}(\mathrm{h}(X,Y), \mathrm{h}(X,Y)))$

Distribute disjunction over conjunction

Convert the wffs to *conjunctive normal form.* A wff is in conjunctive normal form iff it is a conjunction of disjunctions of atoms or negated atoms, i.e. of the form $((P_1 \vee \ldots \vee P_i) \wedge (Q_1 \vee \ldots \vee Q_j) \wedge \ldots \wedge (R_1 \vee \ldots \vee R_k))$, where each P_i, Q_i and R_i is an atom or a negated atom.[2]

Use:

$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$
$(\alpha \wedge \beta) \vee \gamma)) \equiv ((\alpha \vee \gamma) \wedge (\beta \vee \gamma))$
Also
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$
Also
$(\alpha \vee \beta \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$
$(\alpha \wedge \beta \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$

Rewrite as clauses

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

$(\neg \mathrm{p}(X,Y) \vee \mathrm{q}(\mathrm{h}(X,Y), \mathrm{h}(X,Y)))$
needs no changing: a single clause.

Standardise variables apart

Rename variables so that no variable appears in more than one clause.

7.2.2 Another Example

The original wff is $\exists V \mathrm{p}(V) \vee(\forall X(\forall Y\ \mathrm{p}(X,Y) \rightarrow \neg \forall Y(\mathrm{q}(X,Y) \rightarrow \mathrm{r}(X,Y))))$.

1. Relabel clashing variables: $\exists V \mathrm{p}(V) \vee(\forall X(\forall Y\ \mathrm{p}(X,Y) \rightarrow \neg \forall Z(\mathrm{q}(X,Z) \rightarrow \mathrm{r}(X,Z))))$.
2. Eliminate $\rightarrow$ and $\leftrightarrow$: $\exists V \mathrm{p}(V) \vee(\forall X(\neg \forall Y\ \mathrm{p}(X,Y) \vee \neg \forall Z(\neg(\mathrm{q}(X,Z) \vee \mathrm{r}(X,Z))))$.

[2]The next section introduces the terminology "literal". A literal is an atom or the negation of an atom. So this step of the conversion algorithm puts the wff into the form of a conjunction of disjunction of literals.

3. Move ¬ inwards: $\exists V \mathrm{p}(V) \vee (\forall X(\exists Y \neg \mathrm{p}(X, Y) \vee \exists z(\mathrm{q}(X, Z) \wedge \neg \mathrm{r}(X, Z))))$.

4. Skolemise existentials: $\mathrm{p(c)} \vee (\forall X(\neg \mathrm{p}(X, \mathrm{f}(X)) \vee (\mathrm{q}(X, \mathrm{g}(X)) \wedge \neg \mathrm{r}(X, \mathrm{g}(X))))$.

5. Drop universal quantifiers: $\mathrm{p(c)} \vee (\neg \mathrm{p}(X, \mathrm{f}(X)) \vee (\mathrm{q}(X, \mathrm{g}(X)) \wedge \neg \mathrm{r}(X, \mathrm{g}(X))))$.

6. Distribute disjunction over conjunction: $(\mathrm{p(c)} \vee \neg \mathrm{p}(X, \mathrm{f}(X)) \vee \mathrm{q}(X, \mathrm{g}(X))) \wedge (\mathrm{p(c)} \vee \neg \mathrm{p}(X, \mathrm{f}(X)) \vee \neg \mathrm{r}(X, \mathrm{g}(X)))$.

7. Rewrite as clauses: $\mathrm{p(c)} \vee \neg \mathrm{p}(X, \mathrm{f}(X)) \vee \mathrm{q}(X, \mathrm{g}(X))$, $\mathrm{p(c)} \vee \neg \mathrm{p}(X, \mathrm{f}(X)) \vee \neg \mathrm{r}(X, \mathrm{g}(X))$.

8. Standardise variables apart: $\mathrm{p(c)} \vee \neg \mathrm{p}(X_1, \mathrm{f}(X_1)) \vee \mathrm{q}(X_1, \mathrm{g}(X_1))$, $\mathrm{p(c)} \vee \neg \mathrm{p}(X_2, \mathrm{f}(X_2)) \vee \neg \mathrm{r}(X_2, \mathrm{g}(X_2))$.

7.2.3 Useful Terminology

A *literal* is an atom or the negation of an atom. In the following clause,

$$\mathrm{p(c)} \vee \neg \mathrm{p}(X_1, \mathrm{f}(X_1)) \vee \mathrm{q}(X_1, \mathrm{g}(X_1)),$$

the literals are

$\mathrm{p(c)}$,
$\neg \mathrm{p}(X_1, \mathrm{f}(X_1))$ and
$\mathrm{q}(X_1, \mathrm{g}(X_1))$.

We can thus talk of positive and negative literals. The first and third of the literals just given are positive, the second is negative.

As you can see, a clause is a disjunction of literals.

The empty clause is written $\square$, i.e. the disjunction of no literals.

A *ground* clause, term, literal, atom, or wff is one without variables.

Since a clause is a disjunction of literals (i.e. of atoms or negated atoms) and since disjunction is associative and commutative, the order of the literals is immaterial. Furthermore, repeated literals do not affect the truth of a clause, e.g. $\neg \mathrm{p} \vee \mathrm{q} \vee \mathrm{q} \vee \mathrm{r}$ can be simplified to $\neg \mathrm{p} \vee \mathrm{q} \vee \mathrm{r}$. These properties (immaterial order and no need to keep duplicates) mean that we can think of clauses as *sets* of literals, e.g. the clause $\neg \mathrm{p} \vee \mathrm{q} \vee \mathrm{r}$ can be written as a set $\{\neg \mathrm{p}, \mathrm{q}, \mathrm{r}\}$, where the literals are implicitly conjoined. In fact, I shall continue to write clauses as disjunctions of literals, but now that I have established the idea that they can be considered as sets of literals, it will also be legitimate for me to union clauses together or to take subsets of their literals, etc. For example, if I have two clauses $\mathrm{p} \vee \mathrm{q}$ and $\mathrm{q} \vee \neg \mathrm{r} \vee \mathrm{s}$, then their union is $\mathrm{p} \vee \mathrm{q} \vee \neg \mathrm{r} \vee \mathrm{s}$ and this is legitimate because we know we can think of the two clauses as sets, so their union makes sense and we also know that the result will contain no duplicates.

If L is a set of literals (a clause), then let L' denote the set containing the complement of every literal in L. For example, if L is the set $\neg \mathrm{p} \vee \mathrm{q} \vee \mathrm{r}$ then L' is the set $\mathrm{p} \vee \neg \mathrm{q} \vee \neg \mathrm{r}$.

7.2.4 Exercises

Exercise 7.1 *Convert the following wffs to clausal form.*

1. $\forall X \forall Y (p(X,Y) \rightarrow q(X,Y))$
2. $\forall X \forall Y (\neg q(X,Y) \rightarrow \neg p(X,Y))$
3. $\forall X \forall Y (p(X,Y) \rightarrow (q(X,Y) \rightarrow r(X,Y)))$
4. $\forall X \forall Y ((p(X,Y) \wedge q(X,Y)) \rightarrow r(X,Y))$
5. $\forall X \forall Y (p(X,Y) \rightarrow (q(X,Y) \vee r(X,Y)))$
6. $\forall X \forall Y (p(X,Y) \rightarrow (q(X,Y) \wedge r(X,Y)))$
7. $\forall X \forall Y ((p(X,Y) \vee q(X,Y)) \rightarrow r(X,Y))$
8. $\forall X \exists Y (p(X,Y) \rightarrow q(X,Y))$
9. $\neg \forall X \exists Y (p(X,Y) \rightarrow q(X,Y))$
10. $(\neg \forall X\, p(X)) \rightarrow (\exists X\, p(X))$

Exercise 7.2 *(This exercise is not essential: try it only if you're interested.)*

In the file `translate`*, there is Prolog code for translating from first-order logic to clausal form, which you can look at and play with if you fancy. (Clearly you* could *use it to do the previous exercise, but then you wouldn't master the eminently examinable skill of conversion to CFL. Better is to use it to* check *your answers.) The code is well worth skimming over, as it provides examples of Prolog predicate definitions whose recursive structure reflects the recursive structure of the data they operate on (in this case, the data being expressions of first order logic.)*

There is plenty of further practice in conversion to clausal form in worksheet 9.

Acknowledgement: The exercise is from Genesereth & Nilsson.