Knowledge Representation: Logic To CNF To Prolog

Clausal Normal Form

Any FOL (Situation Calculus) formulae can be converted to clausal form:

1. Eliminate implications
2. Move negation inwards
3. Standardize variables apart
4. Skolemize existentials
5. Move universal quantifiers outwards
6. Distribute "and" ∧ over "or" ∨

CNF Step 1: remove implication

$$P \rightarrow Q$$

becomes the logically equivalent

$$\neg P \vee Q$$

Example

$$\forall x\ man(x) \rightarrow human(x)$$

Becomes

$$\forall x\ \neg man(x) \vee human(x)$$

CNF Step 2: move negation inwards

Replace this	With this
$\neg(P \wedge Q)$	$\neg P \vee \neg Q$
$\neg(P \vee Q)$	$\neg P \wedge \neg Q$
$\neg \forall x\ P$	$\exists x\ \neg P$
$\neg \exists x\ P$	$\forall x\ \neg P$
$\neg\neg\ P$	P

CNF Step 3: Standardize Variables Apart

Rename variables such that each quantifier uses a different variable name:

Example:

$$\forall x \left(P(x) \lor \exists x Q(x) \right)$$

Becomes

$$\forall x \left(P(x) \lor \exists y Q(y) \right)$$

This is analogous to Java code (change red x to z):

```
{  int x = 4; int y = 9;
   {  int x = 2;
       x = y + x
     }
}
```

CNF Step 4: Skolemize

Replace existentially quantified variables with skolem functions

Each universally quantified variable whose scope includes the existential becomes a parameter of the skolem function

The idea is that if we know something exists, we can give it a name (any name that works for us is fine)

$\forall x \exists y P(x, y)$ becomes $\forall x P(x, a(x))$ where a depends on x

$\exists y \forall x P(x, y)$ becomes $\forall x P(x, a)$ a doesn't depend on x

Examples:

$\forall x \exists y \, mother(x, y)$ becomes $\forall x \, mother(x, mom(x))$

$\exists x \, moon(x)$ becomes moon(the_moon)

CNF Step 5: move universal quantifiers outwards

After skolemizing, we have only universal quantifiers

Because we've standardized variables apart, we can move them all to the outer left

In fact, we can now assume all variables are universally quantified, and just drop the universal quantifiers

CNF Step 6: Distribute "and" ∧ over "or" ∨

Replace this	With this
$(P \wedge Q) \vee R$	$(P \vee R) \wedge (Q \vee R)$
$P \vee (Q \wedge R)$	$(P \vee Q) \wedge (P \vee R)$

Write down the CNF

Now we have a formula of the form

$P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6 \wedge \cdots$

where each P_i might include one or more disjunctions

We write this as a set of clauses:

$$\{P_1, P_2, P_3, P_4, P_5, P_6, \ldots\}$$

These P_i in general each have this form

$$L_1 \vee L_2 \vee \cdots \vee \neg N_1 \vee \neg N_2 \vee \cdots$$

A Prolog program is a set of Horn clauses

A Horn clause is any clause that has at most one positive literal (0 or 1 positive literals). The positive literals are the L_i

When we write a Prolog program, we are writing down a set of Horn clauses with 0 or 1 L:

$$\{P_1, P_2, P_3, P_4, P_5, P_6, \ldots\}$$

When restricted to be Horn clauses, these P$_i$ in a Prolog program each have this form

One positive, several Negative $\longrightarrow$ $L \lor \neg N_1 \lor \neg N_2 \lor \cdots$

or

One positive, zero Negative $\longrightarrow$ L

or

Zero positive, several Negative $\longrightarrow$ $\neg N_1 \lor \neg N_2 \lor \cdots$

Converting CNF to Prolog

Each of the clauses P_n has this general form:

$$L_1 \vee L_2 \vee \cdots \vee \neg N_1 \vee \neg N_2 \vee \cdots$$

Where we have gathered all of the negative literals on the right side (since the order doesn't matter)

We can put parenthesis in without changing meaning:

$$(L_1 \vee L_2 \vee \cdots) \vee (\neg N_1 \vee \neg N_2 \vee \cdots)$$

Now put in an implication (note reverse direction):

$$(L_1 \vee L_2 \vee \cdots) \leftarrow (N_1 \wedge N_2 \wedge \cdots)$$

Prolog is limited to clauses with 0 or 1 L_i on the left side

Horn Clauses

A horn clause with one L would look like this

$$(L_1) \leftarrow (N_1 \wedge N_2 \wedge \cdots)$$

In our familiar prolog syntax this would would be a rule:

L :- N1,N2,…

A horn clause with just one positive literal is a fact;

L.

A horn clause with no L is a query!

?- N1,N2…

Prolog is limited to Horn Clauses

A clause of this form, not allowed in Prolog:

$$(L_1 \vee L_2) \leftarrow (N_1 \wedge N_2)$$

would be read "If N1 is true and N2 is true, then L1 is true or L2 is true."

A clause of this form, not allowed in Prolog:

$$(L_1 \vee L_2)$$

would be read "L1 is true or L2 is true."

So in a Prolog program we cannot say "the sky is blue or the sea is green":

color(sky,blue);color(sea,green).

Example: Sitcalc to Prolog

Recall we have two types of axiom:

1. Action precondition Axioms (what can happen)

   $$Poss(A(\vec{x}), s) \ \equiv \ \Phi(\vec{x}, s)$$

2. Successor State Axioms (if it happens, then what's true afterwards?)

   $$Poss(A(\vec{x}), s) \supset \ R(\vec{y}, do(A(\vec{x}, s)) \equiv$$
   $$\gamma_R^+(\vec{y}, A(\vec{x}), s) \vee$$
   $$R(\vec{y}, s) \wedge \neg\gamma_R^-(\vec{y}, A(\vec{x}), s)$$

Precondition Axioms

Long story short, when translating equivalence ≡, because of Clark's Completion Semantics, CWA, Negation as Failure, we can simply make the implication go one way only:

$Poss(A(\vec{x}), s) \leftarrow \Phi(\vec{x}, s)$

In actual prolog, we might have this:

```
poss(open_door,S) :-
    near_door(S),
    door_closed(S).
```

No $\vec{x}$ in this case, and $\Phi(s)$is $near\_door(s) \wedge door\_closed(s)$

Successor State Axioms

$$Poss(A(\vec{x}), s) \supset R(\vec{y}, do(A(\vec{x}, s)) \equiv$$
$$\gamma_R^+(\vec{y}, A(\vec{x}), s) \vee$$
$$R(\vec{y}, s) \wedge \neg\gamma_R^-(\vec{y}, A(\vec{x}), s)$$

Becomes

$$R(\vec{y}, do(A(\vec{x}, s)) \leftarrow Poss(A(\vec{x}), s) \wedge$$
$$(\gamma_R^+(\vec{y}, A(\vec{x}), s) \vee$$
$$R(\vec{y}, s) \wedge \neg\gamma_R^-(\vec{y}, A(\vec{x}), s))$$

Not so fast?

Let's replace the three parts of that first big formula on the previous slide with A, B, C:

A -> B <-> C

(-A or B) <-> C

(-A or B) <- C (make implication go one way)

-C or –A or B

A,C->B

B<-A,C

When you look carefully, this is the second formula

(reminder) Steps to Axiomatize a Domain

In the following steps, the word "determine" implies "write down"

1. Understand the domain by reading about it, studying it, and thinking about it
2. Determine the set of fluents that are sufficient to represent a state in the domain
3. Determine the set of actions that effect (bring about) change in the state (fluent truth values)
4. Determine the precondition axioms for actions in terms of fluents
5. Determine the successor state axiom for each fluent
6. Determine the fluent values in the initial situation s0 (we use [] for s0).