Conversion to Clausal Form

Robert Keller
April 2007

To prepare formulas for resolution proving, conversion to an equivalent Prenex form is necessary. Prenex form simply means that all quantifiers are on the outside, and not within any sub-formulas. Once we have Prenex form, there may be some additional steps still needed to get to an equivalent clausal form.

Following these rules will yield an equivalent **Prenex form**:

1. **Rename bound variables** so that there are none that are the same as any free variables. Rename bound variables that have more than one distinct binding by introducing new variables, so that no two bound variables are the same.

2. **Replace implications** $\varphi \rightarrow \psi$ with $\neg\varphi \vee \psi$, so that the only connectives are $\wedge \vee \neg$.

3. **Push any negations inward**, starting at the outermost negations, using these rules:

   a. $\neg\neg\varphi$ becomes φ
   b. $\neg(\varphi\wedge\psi)$ becomes $\neg\varphi \vee \neg\psi$
   c. $\neg(\varphi\vee\psi)$ becomes $\neg\varphi \wedge \neg\psi$
   d. $\neg\exists x\, \varphi$ becomes $\forall x\, \neg\varphi$
   e. $\neg\forall x\, \varphi$ becomes $\exists x\, \neg\varphi$

4. **Push any quantifiers outward**, starting at the outermost negations, using these rules, or their symmetric counterparts where the left-hand side sub-formulas are exchanged:
   a. $(\forall x\, \varphi) \vee \psi$ becomes $\forall x\, (\varphi \vee \psi)$
   b. $(\exists x\, \varphi) \vee \psi$ becomes $\exists x\, (\varphi \vee \psi)$
   c. $(\forall x\, \varphi) \wedge \psi$ becomes $\forall x\, (\varphi \wedge \psi)$
   d. $(\exists x\, \varphi) \wedge \psi$ becomes $\exists x\, (\varphi \wedge \psi)$

   Caution: The correctness of these transformations is based on fact that, after renaming as in step 1, *variable x cannot be free in* ψ.

Note: Rules could be presented for dealing with $\rightarrow$ without converting to $\neg \vee$, but since we want to end up with conjunctive normal form eventually, it is simpler to replace $\rightarrow$ at the outset.

Example 1

$$\forall x\, \exists y\, (\neg P(x, y) \rightarrow \neg((\exists x\, Q(x, y)) \vee \forall y\, R(x, y)))$$

Step 1, rename bound variables:

$\forall x\ \exists y\ (\neg P(x, y) \rightarrow \neg((\exists u\ Q(u, y)) \vee \forall v\ R(x, v)))$

Step 2, replace $\rightarrow$:

$\forall x\ \exists y\ (\neg\neg P(x, y) \vee \neg((\exists u\ Q(u, y)) \vee \forall v\ R(x, v)))$

Step 3, push $\neg$ inward:

$\forall x\ \exists y\ (P(x, y) \vee ((\forall u\ \neg Q(u, y)) \wedge \exists v\ \neg R(x, v)))$

Step 4, push quantifiers outward:

$\forall x\ \exists y\ \forall u\ \exists v\ (P(x, y) \vee (\neg Q(u, y) \wedge \neg R(x, v)))$

Skolemization

In order to eliminate any existentially-quantified variables, we use the convenient device of *Skolem functions*: Introduce a new function symbol having arguments that are the free variables in the sub-formula with the $\exists$ quantifier.

In the above example, we would introduce a Skolem function, say f(x) in place of y, and one g(x, u) in place of v, giving:

$\forall x\ \forall u\ (P(x, f(x)) \vee (\neg Q(u, f(x)) \wedge \neg R(x, g(x, u))))$

If there are no free variables in the sub-formula, then we use a new *constant* symbol instead of a function symbol.

The rationale behind Skolem functions is that their introduction does not change the Satisfiability of a formula. The original formula was satisfiable (by some interpretation) iff there is an interpretation satisfying the new formula. The new interpretation must interpret the skolem functions.

Conversion to Conjunctive Normal Form

The final step is to remove the $\forall$ quantifiers and convert to conjunctive normal form. In the current example, there are two clauses, obtained by distributing $\vee$ over $\wedge$:

1. $P(x, f(x)) \vee \neg Q(u, f(x))$
2. $P(x, f(x)) \vee \neg R(x, g(x, u)$

Resolution Example 1:

Assess the validity of:

$$\forall x\, \exists y\, (P(y, y) \rightarrow P(x, y))$$

We attempt to determine that the negation is unsatisfiable:

$$\neg \forall x\, \exists y\, (P(y, y) \rightarrow P(x, y))$$

There are no renamings required in this case. Replace $\rightarrow$:

$$\neg \forall x\, \exists y\, (\neg P(y, y) \vee P(x, y))$$

Push $\neg$ in:

$$\exists x\, \forall y\, (P(y, y) \wedge \neg P(x, y))$$

Introduce a Skolem constant for x:

$$\forall y\, (P(y, y) \wedge \neg P(c, y))$$

Convert to conjunctive normal form. There are two clauses:

1. $P(y, y)$
2. $\neg P(c, y)$

These immediately resolve, giving the empty clause $\perp$. Thus the original clause was valid.

Resolution Example 2:

Assess the validity of:

$$(\forall x\, P(x) \vee \forall x\, Q(x)) \rightarrow \forall x\, (P(x) \vee Q(x))$$

Negate:

$$\neg((\forall x\, P(x) \vee \forall x\, Q(x)) \rightarrow \forall x\, (P(x) \vee Q(x)))$$

Rename variables:

$$\neg((\forall u\, P(u) \vee \forall v\, Q(v)) \rightarrow \forall x\, (P(x) \vee Q(x)))$$

Eliminate →:

$$\neg(\neg(\forall u\, P(u) \vee \forall v\, Q(v)) \vee \forall x\, (P(x) \vee Q(x)))$$

Push ¬ inward:

$$((\forall u\, P(u)) \vee (\forall v\, Q(v))) \wedge \exists x\, (\neg P(x) \wedge \neg Q(x))$$

Push quantifiers outward:

$$\exists x\, \forall u\, \forall v\, ((P(u) \vee Q(v)) \wedge (\neg P(x) \wedge \neg Q(x)))$$

Introduce Skolem constant:

$$\forall u \forall v\, (P(u) \vee Q(v)) \wedge (\neg P(c) \wedge \neg Q(c))$$

Remove quantifiers and convert to conjunctive normal form:

1. $P(u) \vee Q(v)$
2. $\neg P(c)$
3. $\neg Q(c)$

Resolvents:

4. $Q(v)$ from 1 and 2
5. $\perp$ from 3 and 4

Resolution Example 3:

Assess the validity of:

$$\exists x\, (P(x) \vee Q(x)) \rightarrow (\exists x\, P(x) \vee \exists x\, Q(x))$$

Negate:

$$\neg(\exists x\, (P(x) \vee Q(x)) \rightarrow (\exists x\, P(x) \vee \exists x\, Q(x)))$$

Rename variables:

$$\neg(\exists x\, (P(x) \vee Q(x)) \rightarrow (\exists y\, P(y) \vee \exists z\, Q(z)))$$

Remove →:

$$\neg(\neg\exists x\, (P(x) \vee Q(x)) \vee (\exists y\, P(y) \vee \exists z\, Q(z)))$$

Push ¬ inward:

$$\exists x\, (P(x) \vee Q(x)) \wedge (\forall y\, \neg P(y) \wedge \forall z\, \neg Q(z)))$$

Push quantifiers outward:

$$\exists x\ \forall y\ \forall z\ (P(x) \vee Q(x)) \wedge (\neg P(y) \wedge \neg Q(z)))$$

Introduce Skolem constant c:

$$(P(c) \vee Q(c)) \wedge (\neg P(y) \wedge \neg Q(z)))$$

Convert to conjunctive normal form:

1. $P(c) \vee Q(c)$
2. $\neg P(y)$
3. $\neg Q(z)$

Resolvents:

4.	$Q(c)$	1, 2
5.	$\perp$	3, 4

Using Otter-λ

Otter-λ is a resolution theorem prover, the child of Otter as developed at the Argonne National Laboratory over many years. (The latest version at Argonne is now called Prover9. http://www.cs.unm.edu/~mccune//prover9/) The λ part alludes to the fact that Otter-λ includes better support for proofs about functional programs. It can be downloaded or used on-line at:

```
http://michaelbeeson.com/research/otter-lambda/
```

To ask Otter-λ to find a proof, the input can either be in clausal form or as a list of formulas with quantifiers. The setup is slightly different in these two cases.

Clausal form input:

```
set(auto).

list(usable).
```

… List of premises …

… Negation of the conclusion …

```
end_of_list.
```

One can either use explicit axioms for equality, or use the built-in handling of equality.

Otter-λ Example 1:
This is a result about groups. Equality axioms are explicit.

```
set(auto).
list(usable).

% Group axioms, with left- and right-identity relationships.

e(f(x, f(y, z)), f(f(x, y), z)).
e(f(x, 1), x).
e(f(1, x), x).
e(f(x, i(x)), 1).
e(f(i(x), x), 1).

% Equality axioms

e(x, x).
-e(x, y) | -e(v, w) | e(f(x, v), f(y, w)).
-e(x, y) | e(y, x).
-e(x, y) | -e(y, z) | e(x, z).

% Denial of conclusion

-e(i(i(b)), b).

end_of_list.
```

Disjunction (∨) is represented as |. Negation is represented as -. Implication is represented as ->.

The output of Otter-λ includes the proof search steps, and a proof, if one is found. In the present example, a proof is found and is shown as follows:

```
---------- PROOF ----------

1 [] -e(x,y)| -e(z,u)|e(f(x,z),f(y,u)).
2 [] -e(x,y)|e(y,x).
3 [] -e(x,y)| -e(y,z)|e(x,z).
4 [] -e(i(i(b)),b).
5 [] e(f(x,f(y,z)),f(f(x,y),z)).
6 [] e(f(x,1),x).
7 [] e(f(1,x),x).
8 [] e(f(x,i(x)),1).
16 [hyper,6,3,6] e(f(f(x,1),1),x).
19 [hyper,6,2] e(x,f(x,1)).
26 [hyper,7,3,6] e(f(1,f(x,1)),x).
52 [hyper,8,1,19] e(f(x,f(y,i(y))),f(f(x,1),1)).
58 [hyper,8,1,19] e(f(f(x,i(x)),y),f(1,f(y,1))).
10906 [hyper,52,3,16] e(f(x,f(y,i(y))),x).
10996 [hyper,58,3,26] e(f(f(x,i(x)),y),y).
11017 [hyper,10906,2] e(x,f(x,f(y,i(y)))).
11055 [hyper,10996,3,5] e(f(x,f(i(x),y)),y).
11081 [hyper,11055,3,11017] e(x,i(i(x))).
11228 [hyper,11081,2] e(i(i(x)),x).
11229 [binary,11228.1,4.1] $F.

---------- end of proof ----------
```

The step numbers (on the left) are steps from the search. In general, there will be many derived formulas that are not used in the final proof.

The rationale for the step, along with previous steps on which this step depends, appears inside [. . .]. For hypotheses and the negated conclusion, the rationale is empty.

The rules hyper and binary refer to two different refinements of resolution. Hyper resolves several clauses together in one step, whereas binary is basic resolution of two formulas.

Contradiction (⊥, the empty clause), is shown as `$F` (for false).

Otter-λ Example 2:

This is the same example as above, expressed using the built-in equality rather than as predicate e with supporting axioms:

```
set(auto).
list(usable).

% Group axioms, with left- and right-identity relationships.

f(x, f(y, z)) = f(f(x, y), z).
f(x, 1) = x.
f(1, x) = x.
f(x, i(x)) = 1.
f(i(x), x) = 1.

- (i(i(b)) = b).

end_of_list.
```

The proof is much shorter, due to the use of built-in rules (paramodulation and demodulation) for dealing with equality.

```
---------- PROOF ----------

1 [] i(i(b))!=b.
2 [] f(x,f(y,z))=f(f(x,y),z).
3 [copy,2,flip.1] f(f(x,y),z)=f(x,f(y,z)).
6,5 [] f(x,1)=x.
8,7 [] f(1,x)=x.
9 [] f(x,i(x))=1.
20 [para_from,9.1.1,3.1.1.1,demod,8,flip.1] f(x,f(i(x),y))=y.
26 [para_into,20.1.1.2,9.1.1,demod,6,flip.1] i(i(x))=x.
28 [binary,26.1,1.1] $F.

---------- end of proof ----------
```

Otter-λ Example 3:

This example uses quantifiers. Otter-λ automates the conversion to clausal form. Note that we must use

```
formula_list(usable).
```

instead of just

```
list(usable).
```

to enable this feature.

```
set(auto).
formula_list(usable).

-((all x (s(x) -> t(x))) ->
  ((exists x s(x)) -> (exists x t(x)))).

end_of_list.
```

Otter-λ shows the result of conversion to clausal form:

```
list(usable).
0 [] -s(x)|t(x).
0 [] s($c1).
0 [] -t(x1).
end_of_list.
```

Here `$c1` is a Skolem constant it has introduced and `x1` is a renamed variable.

The proof in this case is very short and direct:

```
---------- PROOF ----------

1 [] -s(x)|t(x).
2 [] -t(x).
3 [] s($c1).
4 [hyper,3,1] t($c1).
5 [binary,4.1,2.1] $F.

------------ end of proof -------------
```