Clausal Form*

Graeme Taylor

February 10, 2004

In a given first order language, there may be many ways to translate a statement into a valid formula- thus given a set of seemingly different formula, it is desirable to be able to transform them into a standard form of notation which preserves logical equivalence. Then, if this form is the same for each formula, then they are in fact different formulations of the same idea. This standardisation is achieved through the generation of the normal form, of which there are many types.

However, it is also desirable to have a normal form which is in some sense the "simplest" rendering of a formula. Clausal form is the closest current approach to this; although ingenuity may offer up a more elegant version of a particular formula, there is no systematic, mechanical method for this (assuming a common agreement on simple can even be made).

The simplicity of clausal form is due to it's composition- a clausal form, rather than being a single long expression such as with CNF, consists of a number of clauses, which have an extremely specific and easily understood structure-

Definition: A formula is said to be a clause if it is of the form-

$$(A_1 \wedge A_2 \wedge ... \wedge A_m) \Rightarrow (B_1 \vee B_2 \vee ... \vee B_n)$$

Where $\wedge$ *denotes AND,* $\vee$ *denotes OR,* $\Rightarrow$ *denotes IMPLIES;* $A_1, ..., B_n$ *are atomic formulae*

Hence, a clause states that if a list of conditions A are met, then at least one other condition from list B must also hold. It is very important to note that A's and B's are atomic formulae, not formulae- thus they contain no logical connectives and in particular no negations. For instance, if you wanted a clause to encapsulate *"I work on weekdays other than tuesdays"* then *"weekday* $\wedge \neg$ *Tuesday* $\Rightarrow$ *I work"* wouldn't do. Instead, you'd need to move the negated atomic formulae to the other side of the implication: *"weekday* $\Rightarrow$ *I work* $\vee$ *Tuesday"* is a correct clause which, if you think about it, has the same meaning. This approach will always work due to the following theorem-

*First appeared on Everything2, at *http://www.everything2.com/index.pl?node_id=1519076*

A disjunction of literals (an atomic formula or its negation) is logically equivalent to a clause.

To prove this, consider that the disjunction can be written as (with appropriate labelling A's and B's and possibly some reordering, which does not alter the meaning of a disjunction)
$\neg A_1 \vee \neg A_2 \vee ... \vee A_m \vee B_1 \vee B_2 \vee ... \vee B_n$
Then this is the same as $\neg(A_1 \wedge A_2 \wedge ... \wedge A_m) \vee (B_1 \vee B_2 \vee ... \vee B_n)$ by DeMorgan's Laws. But if you have $\neg C \vee D$, then this is precisely the definition of $C \Rightarrow D$. So we have the clause structure $(A_1 \wedge A_2 \wedge ... \wedge A_m) \Rightarrow (B_1 \vee B_2 \vee ... \vee B_n)$ as desired.

There are some particularly useful types of clause:

- Clauses with $m = 0$ i.e. of the form $\Rightarrow (B_1 \vee B_2 \vee ... \vee B_n)$
  This means that there are no preconditions for one of the B's to hold- so we can conclude that at least one of them must hold true. If furthermore $n = 1$, then we can accept B_1 as a fact (Its truth is always implied).

- Clauses with $n = 0$ i.e. of the form $(A_1 \wedge A_2 \wedge ... \wedge A_m) \Rightarrow$
  This means that $A_1, ..., A_m$ cannot all hold at the same time (Consider the disjunction of literal form of this clause; with no B's, we have $\neg A_1 \vee \neg A_2 \vee ... \vee A_m$ i.e. the negation of at least one is true- they cannot all be asserted at once).

- Clauses with $n = 1$ i.e. of the form $(A_1 \wedge A_2 \wedge ... \wedge A_m) \Rightarrow B_1$
  These are known as headed horn clauses or just horn clauses and are central to logic programming systems such as PROLOG or Cyc. They are particularly desirable for such reasoning systems as you can assert the truth of B_1 if you have the A_i's all true. Hence if the A's are simpler predicates to test, more complex ideas can be constructed from simpler building blocks; and complex goals can be matched with a probably larger but generally simpler set of sub-goals. Sadly, as the example of working on Tuesdays described above shows, you may not always be able to get a horn clause. (You may be able to work around it by defining a "not tuesday" predicate, but negation of existing predicates can get messy in logic programming- certainly in my experience with prolog!)

So a clausal form breaks down a formula into a list of the rules it gives for the universe or domain described by the first order language. Testing individual clauses can give insight into predicates contained within the original formula, and this will generally be a lot simpler than trying to infer their truth value direct from that formula. It is also handy when faced with trying to say something useful if presented with incomplete data, as some predicates may have no bearing on the values of others. How then to find a clausal form?

Clausal Form Algorithm

This requires a sentence A- that is, there can be no free variables. It may therefore be necessary to use the universal closure of A (as if that holds then the original statement A does; although not the converse.)

1. Find the Prenex normal form of A.
2. Find Skolem form. This has some effect on logical equivalence which carries through to the clausal form (see below).
3. We now have something of the form $(\forall X_1)...(\forall X_n)M(X_1, ..., X_n)$ where M is quantifier free. Remove the universal quantifiers- literally, this is a typographical operation, just delete them and keep M.
4. Obtain the CNF of M. This gives $C_1 \wedge C2 \wedge ... \wedge C_j$ with the C's being disjunctions of literals. This is computationally intensive, being of exponential order relative to the number of predicates.
5. Rewrite the conjunction as a list $\Gamma = [\ C_1, ..., C_j\ ]$.
6. Rewrite each C_i as a clause (as discussed in the above theorem). The resultant list Γ of clauses is the clausal form of sentence A.

As noted, when A is skolemized we alter the equivalence. The end result is that any interpretation for which the clausal form is true makes A true; if we have an interpretation I in which A holds then by appropriate definition of skolem functions we can extend I to I^* in which the clausal form is true. Thus, the clausal form is satisfiable if and only if the original sentence is satisfiable.

A worked example of clausal form

Some people like other people who do not like anyone

- looks quite a mess! We can put this into a first order language using predicates $p(A)$ to mean A is a person, $likes(A, B)$ to mean A likes B, our domain is all the objects in the universe. This yields:

$$(\exists X)(p(X) \wedge (\exists Y)(p(Y) \wedge likes(X, Y) \wedge (\forall Z)(p(Z) \Rightarrow \neg likes(Y, Z))))$$

That is, Y is a person who doesn't like any things that happen to be people; and X likes Y. We can immediately pull quantifiers through for a prenex form:

$$(\exists X)(\exists Y)(\forall Z)(p(X) \wedge p(Y) \wedge likes(X, Y) \wedge (p(Z) \Rightarrow \neg likes(Y, Z)))$$

Now we introduce skolem constants- use α for the person in our universe who satisfies X (a person liking Y) and β for the person who likes no other people (i.e. the person who satisfies Y). We'll also drop the universal quantifier on Z as directed:

$$(p(\alpha) \wedge p(\beta) \wedge likes(\alpha, \beta) \wedge (p(Z) \Rightarrow \neg likes(\beta, Z))$$

Rewriting the implication gives us

$$(p(\alpha) \wedge p(\beta) \wedge likes(\alpha, \beta) \wedge (\neg p(Z) \vee \neg likes(\beta, Z))$$

This is conjunctive normal form; we get a list of clauses as follows:

- $\Rightarrow p(\alpha)$
- $\Rightarrow p(\beta)$
- $\Rightarrow likes(\alpha, \beta)$
- $p(Z) \wedge likes(\beta, Z) \Rightarrow$

Which is a very simple formulation- it tells us that α is a person; so is β , α likes β ; and it's impossible to find a Z which is both a person and liked by β .