32002: AI (Conversion clausal form, unification, mgu, theorem proving) **Spring 2015**

Lecture 5: January 15, 2015

Lecturer: K.R. Chowdhary *: Professor of CS (VF)*

5.1 Conversion to Clausal Form

Following are the steps to convert a predicate formula into clausal-form.

1. Eliminate all the implications: $p \rightarrow q \equiv \neg p \vee q$.
2. Move the outer negative symbol into the atom: replace $\neg \forall x p(x)$ by $\exists x \neg p(x)$.
3. Replace $\exists x \forall y(f(x) \rightarrow f(y))$ by $\forall y(f(a) \rightarrow f(y))$.
4. Rename the variables if necessary.
5. Replace existentially quantified variables with *Skolem* functions and drop corresponding quantifiers: for $\forall x \exists y[\neg p(x) \vee q(y)]$, we obtain $\forall x[\neg p(x) \vee q(f(x))$.
6. Move the universal quantifiers to the left of equation: $\exists x[\neg p(x) \vee \forall y\ q(y)]$ to $\exists x \forall y[\neg p(x) \vee q(y)]$
7. Terms should be connected by conjunctions only, vertically.
8. Eliminate the conjunctions.
9. Rename the variables, if necessary.
10. Drop all the universal quantifiers, and write each term in a separate line.

Example 5.1 *Convert the expression* $\exists x \forall y[\forall z\ p(f(x), y, z) \Rightarrow (\exists u\ q(x, u) \wedge \exists v\ r(y, v))]$ *to clausal form.*

Solution. Steps:

1. $$\exists x \forall y[\neg \forall z\ p(f(x), y, z) \vee (\exists u\ q(x, u) \wedge \exists v\ r(y, v))]$$
2. $$\exists x \forall y[\exists z \neg p(f(x), y, z) \vee (\exists u\ q(x, u) \wedge \exists v\ r(y, v))]$$
3. $$\forall y[\exists z \neg p(f(a), y, z) \vee (\exists u\ q(a, u) \wedge \exists v\ r(y, v))]$$
4. Rename variables (not required in this example.)

5.
$$\forall y[\neg p(f(a), y, g(y) \vee (q(a, h(y) \wedge r(y, l(y))]$$

6. Not required in this example.

7. Move disjunctions down to literals.

$$\forall y[(\neg p(f(a), y, g(y)) \vee (q(a, h(y))) \wedge (\neg p(f(a), y, g(y)) \vee r(y, l(y)))]$$

8. Eliminate conjunctions.

$$\forall y[\neg p(f(a), y, g(y)) \vee (q(a, h(y)), (\neg p(f(a), y, g(y)) \vee r(y, l(y))]$$

9. Not required in this example.

10. Drop all universal quantifiers and write each term on separate line.

$$\neg p(f(a), y, g(y)) \vee (q(a, h(y)),$$
$$\neg p(f(a), y, g(y)) \vee r(y, l(y)).$$

□

5.2 Substitutions and Unification

The following definitions are concerned with the operation of *instantiation*, i.e, substitutions of terms for variables in the well-formed expressions and in sets of well-formed expressions.

Substitutions. A substitution is any finite set (possibly empty) of substitution components, none of the variables of which are same. If P is any set of terms, and the terms of the components of the substitution θ are all in P, we say that θ is a substitution over P. We write the substitution where components are $T_1/V_1, \ldots, T_k/V_k$ as $\{T_1/V_1, \ldots, T_k/V_k\}$, with the understanding that order of components is immaterial. We will use lowercase Greek letters θ, λ, μ denote substitutions.

Instantiations. If E is any function string of symbols, and $\theta = \{T_1/V_1, \ldots, T_k/V_k\}$ is any substitution, then the instantiation of E by θ is the operation of replacing each occurrence of variable $V_i, 1 \leq i \leq k$, in E by occurrence of the term T_i. The resulting string denoted by $E\theta$ is called an instance of E by θ, i.e., if E is the string $E_0V_{i_1}E_1 \ldots V_{i_n}En$, then $E\theta$ is the string $E_0T_{i_1}E_1 \ldots T_{i_n}En$. Here, none of the substrings E_j of E contain occurrences of variables $V_1, \ldots, V_k$, some of E_j are possibly *null*, n is possibly 0, and each V_{i_j} is an occurrence of one of the variables $V_1, \ldots, V_k$. Any string $E\theta$ is called an instance of the string E.

If C is any set of strings and θ is substitution, then the instance of C by θ is the set of all strings $E\theta$, where E is in C. We denote this set by $C\theta$, and say that it is an instance of C.

Composition of Substitutions. If $\theta = \{T_1/V_1, \ldots, T_k/V_k\}$ and λ are any two substitutions, then the composition of θ and λ denoted by $\theta\lambda$ is union $\theta^{'} \cup \lambda^{'}$, defined as follows:

The $\theta^{'}$ is set of all components $T_i\lambda/V_i, 1 \leq i \leq k$, such that $T_i\lambda$ (λ substituted in θ) is different from V_i, and $\lambda^{'}$ is set of all components of λ whose variables are not among $V_1, \ldots, V_k$.

Within a given scope, once a variable is bound, it may not be given a new binding in future unifications and inferences. If θ and λ are two substitution sets, then the composition of θ and λ, i.e., $\theta\lambda$, is obtained by applying $\theta\lambda$ to the elements of θ and adding the result to λ.

Example 5.2 *Find out the composition of* $\{x/y, w/z\}, \{v/x\}$, *and* $\{A/v,\ f(B)/w\}$.

Solution. Let us assume that $\theta = \{x/y, w/z\}$, $\lambda = \{v/x\}$ and $\mu = \{A/v, f(B)/w\}$. Following are the steps:

1. To find the composition $\lambda\mu$, A is is substituted for v, and v is then substituted for x. Thus, $\lambda\mu = \{A/x, f(B)/w\}$.
2. When result of $\lambda\mu$ is substituted in θ, we get composition $\theta\lambda\mu = \{A/y, f(B)/z\}$. □

Example 5.3 *Find out the composition of* $\lambda = \{g(x, y)/z\}$, *and* $\theta = \{A/x, B/y,\ C/w, D/z\}$.

Solution. By composition,

$$\begin{aligned}\lambda\theta &= \{g(x, y)/z\} \circ \{A/x, B/y\} \\ &= \{g(A, B)/z, A/x, B/y, c/w\}\end{aligned}$$

The $\{D/z\}$ has not been included in the resultant substitution set, because otherwise, there will be two terms for the variable z, one $g(A, B)$ and other D. □

One of the important property of substitution is that, if E is any string, and $\sigma = \theta\lambda$, then $E\sigma = E\theta\lambda$. It is straight forward to verify that $\epsilon\theta = \theta\epsilon = \theta$ for any substitution θ. Also, composition enjoys the associative property $(\theta\lambda)\mu = \theta\lambda\mu)$, so we may omit the parentheses in writing multiple compositions of substitutions. The substitutions are not in general commutative; i.e., it is generally not the case that $\theta\lambda = \lambda\theta$.

The point of the composition operation on substitution is that, when E is any string, and $\sigma = \theta\lambda$, the string $E\sigma$ is just the string $E\theta\lambda$, i.e., the instance of $E\theta$ by λ. The composition also have distributive property.

5.2.1 Most General Unifiers

The most important way of avoiding needless search in a first-order derivation is to keep the search as general as possible. Consider, for example two clauses

$$c_1 = p(g(x), f(x), z),$$

and

$$c_2 = \neg p(y, f(w), a).$$

These two literals are unified by the substitution,

$$\theta_1 = \{b/x, g(b)/y, a/z, b/w\},$$

and also by,

$$\theta_2 = \{f(z)/x, g(f(z))/y, a/z, f(z)/w\}.$$

We may very well be able to derive the empty clause using c_1, c_2 with substitution of θ_1; but if we cannot, we will need to consider other substitutions like θ_2.

The trouble is that both of these substitutions are overly specific. We can see that any unifier must give w the same value as x, and to y the same as $g(x)$, but we do not need to commit yet to a value for x. The substitution,

$$\theta_3 = \{g(x)/y, a/z, x/w\}$$

unifies the two literals without making an arbitrary choice that might preclude a path to the empty clause. It is a most general unifier.

More precisely, a most general unifier (mgu)(i. e., simplest one) θ of literals ρ_1 and ρ_2 is a unifier that has the property that for any other unifier $\theta^{'}$, there is a further substitution θ^* such that $\theta^{'} = \theta\theta^*$. So starting with θ you can always get to any other unifier by applying additional substitutions. For example, given θ_3, we can get to θ_1 by further applying b/x, and to θ_2 by applying $f(z)/x$. Note that an mgu need not be unique, in that $\theta_4 = \{g(w)/y, a/z, w/x\}$ is also an *mgu* for c_1 and c_2.

The key fact about mgus is that we can limits the resolution rule to mgus without loss of completeness. This helps immensely in the search since it dramatically reduce the number of resolvents that can be inferred from these two input clauses.

Example 5.4 *Given a unifier, obtain a more general unifier.*

Suppose you have two expressions $p(x)$ an $p(y)$. One way to unify these is to substitute any constant expression for x and y: $S = \{fred/x, fred/y\}$. But this is not the most general unifier, because if we substitute any variable for x and y, we get a more general unifier: $G = \{z/x, z/y\}$. The first unifier is a valid unifier, but it would lessen the generality of inferences that we might want to make.

Let, $E = \{p(x), p(y)\}$
Let, $S = \{fred/x, fred/y\}$
Let, $G = \{z/x, z/y\}$
Now, let $S^{'} = \{fred/z\}$
Then $ES = \{p(fred), p(fred)\}$
and $GS^{'} = \{fred/x, fred/y\}$
and therefore $EGS^{'} = \{p(fred), p(fred)\} = ES$. $\square$

So, given a unifier, you can always create a more general unifier. When both of these unifiers are composed and instantiate the original expression E, you get the same instance as it was obtained with the earlier unifier.

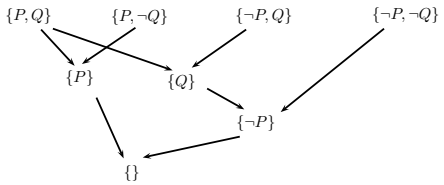

Figure 5.1: DAG for Theorem Proving.

5.3 Resolution Principle

Recall that a literal, L, is either a propositional letter, P, or the negation, $\neg P$, of a propositional letter. A clause is a finite set of literals, $\{L_1, \ldots, L_k\}$, interpreted as the disjunction $L_1 \vee \cdots \vee L_k$ (when $k = 0$, this is the empty clause denoted []). A set of clauses, $\Gamma = \{C_1, \ldots, C_n\}$, is interpreted as the conjunction $C_1 \wedge \cdots \wedge C_n$. For short, we write $\Gamma = C_1, \ldots, C_n$. The resolution method is a procedure for checking whether a set of clauses, Γ, is *unsatisfiable*. The resolution methods consist in building a certain kind of labeled *DAG* (Directed Acyclic Graph) whose leaves are labeled with clauses in Γ and whose interior nodes are labeled according to the resolution rule.

Given two clauses $C = A \cup \{P\}$ and $C' = B \cup \{\neg P\}$ (where P is a propositional letter, $P \notin A$ and $\neg P \notin B$), the resolvent of C and C' is the clause $R = A \cup B$ obtained by canceling out P and $\neg P$. A resolution DAG for Γ is a DAG whose leaves are labeled with clauses from Γ and such that every interior node n has exactly two predecessors, n_1 and n_2 so that n is labeled with the resolvent of the clauses labeling n_1 and n_2. A resolution refutation for Γ is a resolution DAG with a single root whose label is the empty clause.

Example 5.5 *Resolution refutation.*

A resolution refutation for the set of clauses

$$\Gamma = \{\{P, Q\}, \{P, \neg Q\}, \{\neg P, Q\}, \{\neg P, \neg Q\}\}.$$

is shown in figure 5.1.

□

5.3.1 Theorem Proving Formalism

It is a syntactic inference procedure, when applied to clauses, determines, if the satisfied set is unsatisfiable. Proof is similar to proof by *contradiction* and deduce [] (i.e., null). If for example, we have set of clauses (axioms) $C_1, C_2, \ldots, C_n$, and we want to deduce D, i.e., D is logical consequence of of $C_1, C_2, \ldots, C_n$. For this we add $\neg D$ to the set $\{C_1, C_2, \ldots, C_n\}$, then we show that set is unsatisfiable by deducing contradiction.

The process of deduction using resolution is given in algorithm 1. Given two clauses C_1, C_2 with no variables in common, and if l_1 is a literal in C_1 and its complement literal l_2 is a literal in C_2, then l_1, l_2 can be dropped and disjunction C is obtained from the remaining part of C_1, C_2. The C is called resolvent of C_1, C_2.

Let $C_1 = \neg P \vee Q$, and $C_2 = \neg Q \vee R$, then following can be deduced through resolution,

$$\frac{P \Rightarrow Q, Q \Rightarrow R}{P \Rightarrow R} \tag{5.1}$$

equivalently,

$$\frac{(\neg P \vee Q), (\neg Q \vee R)}{\therefore (\neg P \vee R)} \tag{5.2}$$

It can be easily verified that $(\neg P \vee Q) \wedge (\neg Q \vee R) \models (\neg P \vee R)$, hence $(\neg P \vee Q) \wedge (\neg Q \vee R) \Rightarrow (\neg P \vee R)$ is a valid statement. Thus, $\neg P \vee R$ is inference or the resolvent. Arriving to a proof by above is called proof by *refutation.*

Resolution says that if there are axioms of the form $\neg P \vee Q$ and there is another axiom of the form $\neg Q \vee R$, then $\neg P \vee R$ logically follows; called the *resolvent.* Let us see why it is so? When $\neg P \vee Q$ is True, then either $\neg P$ is True or Q is True. For other expression, when $\neg Q \vee R$ is True, then either $\neg Q$ is True or R is True. Then we can say that $\neg P \vee R$ is certainly True. This can be generalized to two expressions, when we have any number of expressions, but two must be of opposite signs.

5.3.2 Proof by Resolution

To prove a theorem, one obvious strategy is to search forward from the axioms, using sound rules of inference. Hence, we try to prove a theorem by refutation. It requires to show that negation of a theorem cannot be True. The steps for a proof by resolution are:

1. Assume that negation of the theorem is True.
2. Try to show that axioms and assumed negation of theorem, together are True, which cannot be True.
3. Conclude that above leads to contradiction.
4. Conclude that theorem is True because its negation cannot be True.

To apply the resolution rule,

1. Find two sentences that contain the same literal, once in its positive form and once in its negative form, like,
$$CNF : summer \vee winter, \neg winter \vee cold,$$
2. use the *resolution* rule to eliminate the complement literals from both sentences to get,
$$CNF : summer \vee cold.$$

Algorithm 1 Algorithm-Resolve(Input: α(the theorem to be proved), β(the set of axioms))

```
β = β ∪ {¬α}
Γ = clausal form of β
while there is a resolvable pair of clauses C_i, C_j ∈ Γ do
    C = resolve(C_i, C_j)
    if C = NIL then
        return "Theorem α is true"
    end if
    Γ = Γ ∪ {C}
end while
Report that theorem is False
```

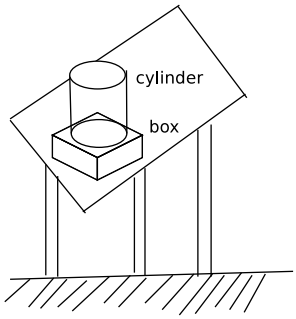

Figure 5.2: Objects on Table.

Example 5.6 *Theorem proving using resolution-refutation.*

Following axioms are about the observed block relationship shown in figure 5.2.

$$on(cylinder, box)$$

$$on(box, table)$$

These are already in the clausal form, hence need not to be converted to the clausal form.

It is required to show that object *cylinder* is above table, i.e., $above(cylinder, table)$, given the the following rules:

- $\forall x \forall y[on(x, y) \rightarrow above(x, y)]$, and
- $\forall x \forall y \forall z[above(x, y) \wedge above(y, z) \rightarrow above(x, z)]$.

After we have gone through the procedure for conversion to clausal form, the above axioms get transformed to:

- $\neg on(u, v) \vee above(u, v)$
- $\neg above(x, y) \vee \neg above(y, z) \vee above(x, z)$

The expression to be proved is "above(cylinder, table)"; its negation is $\neg above(cylinder, table)$. Let us list all the clauses systematically.

(1) $\neg on(u, v) \vee above(u, v)$

(2) $\neg above(x, y) \vee \neg above(y, z) \vee above(x, z)$

(3) $on(cylinder, box)$

(4) $on(box, table)$

(5) $\neg above(cylinder, table)$

Now, we manually run the algorithm 1 on the clauses (1)-(5), as well as those which would created new, to unify them according to unification algorithm **??**, until we reach to a null resolvent.

We resolve clauses (2) and (5) and bind x to '*cylinder*'and z to '*table*'. Applying the resolution, we get resolvent (6). Unifier for this is $\{cylinder/x, table/z\}$.

(2) $\neg above(cylinder, y) \vee \neg above(y, table) \vee above(cylinder, table)$

(5) $\neg above(cylinder, table)$

(6) $\neg above(cylinder, y) \vee \neg above(y, table)$

Now, we resolve (1) with (6), instantiating u with y and instantiating (called binding) v with '*table*', we get (7). Unifier for this is $\{y/u, table/v\}$.

(1) $\neg on(y, table) \vee above(y, table)$

(6) $\neg above(cylinder, y) \vee \neg above(y, table)$

(7) $\neg on(y, table) \vee \neg above(cylinder, y)$

We use (1) again with (7) with u instantiated to $cylinder$ and v replaced y. Unifier for this is $\{cylinder/u, y/v\}$.

(1) $\neg on(cylinder, y) \vee above(cylinder, y)$

(7) $\neg on(y, table) \vee \neg above(cylinder, y)$

(8) $\neg on(cylinder, y) \vee \neg on(y, table)$

Next use (3) and (8), instantiating y to box, with unifier $\{box/y\}$. a set with exactly one element

(3) $on(cylinder, box)$

(8) $\neg on(cylinder, box) \vee \neg on(box, table)$

(9) $\neg on(box, table)$

Finally, the (4) and (9) are resolved to get NIL:

(4) $on(box, table)$

(9) $\neg on(box, table)$

(10) NIL

Since we have arrived at the contradiction, so that negation of the theorem: $\neg above(cylinder, table)$ must be False. Hence the theorem $above(cylinder, table)$ must be True. □