COMPSCI 366 S1 C 2006
Foundations of Artificial Intelligence

—Resolution—

Hans W. Guesgen
Computer Science Department

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

The Main Features of Resolution

- Produces proofs by refutation.
- Operates on wff's in clause form.
- Uses Unification to compare literals.

Conjunctive Normal Form

All Romans know each other but they know people who aren't Romans also.

$[\forall x : Roman(x) \rightarrow ([\forall y : Roman(y) \rightarrow knows(x,y)] \wedge$
$\neg[\forall y : knows(x,y) \rightarrow Roman(y)])]$

Aim:

- Flat structure
- Quantifiers separated

$[\neg Roman(x) \vee \neg Roman(y) \vee knows(x,y)] \wedge$
$[\neg Roman(x) \vee knows(x, f(x,y))] \wedge$
$[\neg Roman(x) \vee \neg Roman(f(x,y))]$

Clause Form

- For resolution to work, go one step further.
- Reduce a set of wff's to a set of clauses.
- A set of clauses is a wff in conjunctive normal form without $\wedge$.

Algorithm: Convert to Clause Form

1. Eliminate $\rightarrow$, using the fact that $a \rightarrow b$ is equivalent to $\neg a \vee b$.

$$[\forall x : \neg Roman(x) \vee ([\forall y : \neg Roman(y) \vee knows(x, y)] \wedge$$
$$\neg[\forall y : \neg knows(x, y) \vee Roman(y)])]$$

Algorithm: Convert to Clause Form (cont'd)

2. Reduce the scope of each $\neg$ to a single term, using:

   - $\neg(\neg p) = p$
   - DeMorgan's laws
     $\neg(a \wedge b) = \neg a \vee \neg b$
     $\neg(a \vee b) = \neg a \wedge \neg b$
   - Correspondences between quantifiers
     $\neg\forall x : P(x) = \exists x : \neg P(x)$
     $\neg\exists x : P(x) = \forall x : \neg P(x)$

$$[\forall x : \neg Roman(x) \vee ([\forall y : \neg Roman(y) \vee knows(x,y)] \wedge$$
$$[\exists y : knows(x,y) \wedge \neg Roman(y)])]$$

Algorithm: Convert to Clause Form (cont'd)

3. Standardize variables so that each quantifier binds a unique variable.

$$\forall x : \neg Roman(x) \vee (\forall y : \neg Roman(y) \vee knows(x, y) \wedge \\ \exists w : knows(x, w) \wedge \neg Roman(w))$$

Algorithm: Convert to Clause Form (cont'd)

4. Move all quantifiers to the left of the formula without changing their relative order.

$\forall x : \forall y : \exists w :$
$\neg Roman(x) \vee (\neg Roman(y) \vee knows(x, y) \wedge$
$knows(x, w) \wedge \neg Roman(w))$

Algorithm: Convert to Clause Form (cont'd)

5. Eliminate existential quantifiers by replacing them with Skolem functions.

$\forall x : \forall y :$
$\neg Roman(x) \vee (\neg Roman(y) \vee knows(x, y) \wedge$
$knows(x, f(x, y)) \wedge \neg Roman(f(x, y)))$

Algorithm: Convert to Clause Form (cont'd)

6. Drop the quantifiers.

$$\neg Roman(x) \lor (\neg Roman(y) \lor knows(x,y) \land$$
$$knows(x, f(x,y)) \land \neg Roman(f(x,y)))$$

Algorithm: Convert to Clause Form (cont'd)

7. Convert the wff into a conjunctive normal form.

   - Distributive laws
     $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$
     $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$

$$[\neg Roman(x) \vee \neg Roman(y) \vee knows(x, y)] \wedge$$
$$[\neg Roman(x) \vee knows(x, f(x, y))] \wedge$$
$$[\neg Roman(x) \vee \neg Roman(f(x, y))]$$

Algorithm: Convert to Clause Form (cont'd)

8. Create a separate clause corresponding to each conjunct.

$$\neg Roman(x) \vee \neg Roman(y) \vee knows(x, y)$$
$$\neg Roman(x) \vee knows(x, f(x, y))$$
$$\neg Roman(x) \vee \neg Roman(f(x, y)))$$

Algorithm: Convert to Clause Form (cont'd)

9. Rename the variables so that no two clauses make reference to the same variable

$\neg Roman(x) \vee \neg Roman(y) \vee knows(x, y)$
$\neg Roman(z) \vee knows(z, f(z, w))$
$\neg Roman(v) \vee \neg Roman(f(v, u)))$

The Basis of Resolution

- Resolution is a iterative procedure.
- At each step, two clauses (called parent clauses) are resolved, yielding a new clause.
- No strong control strategies.

Example

Eliminating literals:

$$\frac{\begin{array}{l} winter \vee summer \\ \neg winter \vee cold \end{array}}{summer \vee cold}$$

Contradiction:

$$\frac{\begin{array}{l} winter \\ \neg winter \end{array}}{\square}$$

Algorithm: Propositional Resolution

Given a set of axioms F, prove proposition P.

1. Convert all propositions of F to clause form.
2. Negate P and convert the result to clause form. Add it to the set of clauses.
3. Repeat until either a contradiction is found or no progress can be made:
   (a) Select two clauses. Call these the parent clauses.
   (b) Resolve them together. The resolvent will be the disjunction of all of the literals of both of the parent clauses with the following exception: If there are any pairs of literals L and $\neg L$ such that one of the parent clauses contains L and the other contains $\neg L$, then select one such pair and eliminate both L and $\neg L$ from the resolvent.
   (c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

Example

Assume we want to prove R.

Given axioms	Clause form
P	P
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$
	$\neg T \vee Q$
T	T

$$\begin{tikzcd}
\neg P \vee \neg Q \vee R \arrow[dr] & & \neg R \arrow[dl] & \\
& \neg P \vee \neg Q \arrow[dr] & & P \arrow[dl] \\
\neg T \vee Q \arrow[dr] & & \neg Q \arrow[dl] & \\
& \neg T \arrow[dr] & & T \arrow[dl] \\
& & \square &
\end{tikzcd}$$

Resolution in Predicate Logic

In predicate logic, the situation is more complicated, since we must consider all possible ways of substituting values for the variables.

The theoretical basis of the resolution procedure in predicate logic is Herbrand's theorem:

- To show that a set of clauses S is unsatisfiable, it is necessary to consider only interpretations over a particular set, called the Herbrand universe of S.
- A set of clauses S is unsatisfiable if and only if a finite subset of ground instances (in which all bound variables have had a value substituted for them) of S is unsatisfiable.

To compare two literals with variables, apply unification.

Unification is a matching procedure that compares two literals and discovers whether there exists a set of substitutions that makes them identical.

Algorithm: Unify

Given two literals $L1$ and $L2$.

1. If $L1$ or $L2$ are variables or constants, then:
   (a) If $L1$ and $L2$ are identical, then return NIL.
   (b) Else if $L1$ is a variable, then if $L1$ occurs in $L2$ then return $\{$FAIL$\}$, else return $(L2/L1)$.
   (c) Else if $L2$ is a variable, then if $L2$ occurs in $L1$ then return $\{$FAIL$\}$, else return $(L1/L2)$.
   (d) Else return $\{$FAIL$\}$
2. If the initial predicate symbols in $L1$ and $L2$ are not identical, then return $\{$FAIL$\}$.
3. If $L1$ and $L2$ have a different number of arguments, then return $\{$FAIL$\}$.

Algorithm: Unify (cont'd)

```
4. Set $SUBST$ to NIL. (At the end of this procedure, $SUBST$ will contain all the
   substitutions used to unify $L1$ and $L2$.)
5. For $i \leftarrow 1$ to number of arguments in $L1$:
   (a) Call $\mathrm{Unify}$ with the $i$th argument of $L1$ and the $i$th argument of $L2$, putting result
       in $S$.
   (b) If $S$ contains FAIL then return $\{$FAIL$\}$.
   (c) if $S$ is not equal to NIL then:
        i. Apply $S$ to the remainder of both $L1$ and $L2$.
       ii. $SUBST := \mathrm{APPEND}(S, SUBST)$.
6. Return $SUBST$.
```

Unify Example

1. $\neg Roman(x_2) \vee loyalto(x_2, Caesar) \vee hate(x_2, Caesar)$

2. $hate(Marcus, Caesar)$

Algorithm: Resolution

Given a set of statements F, prove statement P.

1. Convert all statements of F to clause form.
2. Negate P and convert the result to clause form. Add it to the set of clauses.
3. Repeat until either a contradiction is found, no progress can be made, or a predetermined amount of effort has been expended:
   (a) Select two parent clauses and resolve them together.
   (b) The resolvent will be the disjunction of all the literals (with sustitutions) of both parent clauses with the exception of one pair of complimentary literals:
      - One of the parent clauses contains a literal $T1$ and the other $\neg T2$.
      - $T1$ and $T2$ are unifiable.

   (c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

Example

Axioms in clause form:

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\neg Pompeian(x_1) \vee Roman(x_1)$
4. $ruler(Caesar)$
5. $\neg Roman(x_2) \vee loyalto(x_2, Caesar) \vee hate(x_2, Caesar)$
6. $loyalto(x_3, f_1(x_3))$
7. $\neg man(x_4) \vee \neg ruler(y_1) \vee \neg tryassassinate(x_4, y_1) \vee \neg loyalto(x_4, y_1)$
8. $tryassassinate(Marcus, Caesar)$

Prove:

9. $hate(Marcus, Caesar)$

Example (cont'd)

- Assumption: $\neg hate(Marcus, Caesar)$
- Assumption and 5 result in 10: $\neg Roman(Marcus) \vee loyalto(Marcus, Caesar)$
- 2 and 3 resolve in 11: $Roman(Marcus)$
- 10 and 11 resolve in 12: $loyalto(Marcus, Caesar)$
- 8 and 7 resolve in 13: $\neg man(Marcus) \vee \neg ruler(Caesar) \vee \neg loyalto(Marcus, Caesar)$
- 12 and 13 resolve in 14: $\neg man(Marcus) \vee \neg ruler(Caesar)$
- 14 and 4 resolve in 15: $\neg man(Marcus)$
- 15 and 1 resolve in contradiction: $\square$
- Result: $hate(Marcus, Caesar)$

Some Remarks on Resolution

- Resolution will find a contradiction if one exists.
- However, it may take a very long time.
- There are strategies for speeding up the process:
  - Remove all tautologies $(P(x) \vee Q(x), \neg P(x) \vee \neg Q(x))$.
  - Remove subsumed clauses $(P(a) \vee Q(y)$ is subsumed by $P(a))$.
  - Resolve pairs of clauses that contain complementary literals.
  - Whenever possible, resolve either with one of the clauses that is part of $\neg P$ or with a clause generated by a resolution with such a clause (set-of-support strategy).
  - Only make resolvents that use the most recent resolvent as one of their parents (linear format).
  - Whenever possible, resolve with clauses that have a single literal (unit-preference strategy).