1

Knowledge Based Systems

(CM0377)

Lecture 5

(Last modified 8th February 2001)

2

Resolution (again!)

- Suppose we have two clauses:
  $A; A_{1,1}; \ldots; A_{1,k}$:-$B_{1,1}, \ldots, B_{1,l}$
  $A_{2,1}; \ldots; A_{2,k}$:-$B_{2,1}, \ldots, B_{2,m}, A$
- We *select* a literal to resolve on, which appears in the head of one clause and body of the other - in this case, w.l.o.g., *A*
- The resolvent is
  $A_{1,1}; \ldots; A_{1,k}; A_{2,1}; \ldots; A_{2,k}$:-
  $B_{1,1}, \ldots, B_{1,l}, B_{2,2}, \ldots, B_{2,m}$
- except that if a literal occurs in the head of *both* clauses, it occurs only once in the resolvent; similarly if it occurs in the body

3

Simple example

- Resolve:
  `get_wet; umbrella:-rain`
  `:-umbrella`
- on `umbrella` to get:
  `get_wet:-rain`
- (In other words, if it rains either you have an umbrella or you get wet; you don't have an umbrella, so the consequence is that if it rains you get wet!)

4

Another simple example

- Resolve:

```
get_wet:-rain
rain:-get_wet
```

  on `rain` to get:

```
get_wet:-get_wet
```

- (If it rains you get wet; if you get wet it rains. The resolvent is a *tautology,* in that it is true in every interpretation)
- Could also resolve on `get_wet` to obtain:

```
rain:-rain
```

5

Resolution - another example

- We saw that resolvent of:

```
happy:-boy, has_football.
has_football; impoverished:-boy,
  good_footballer.
```

} P

- Is:

```
happy; impoverished:-boy,
  good_footballer.
```

} C

- We write $P \vdash C$. But is it true that $P \models C$?

6

Is it a logical consequence?

- It's a logical consequence if all models of *P* are also models of *C*. Models of *P* are:
  - `{happy, has_football}`
  - `{happy, impoverished}`
  - `{boy}`
- We need to draw a kind of truth table ...

7

happy	boy	has_football	impoverished	good_footballer	model of P	model of C
f	f	f	f	f	t	t
f	f	f	f	t	t	t
f	f	f	t	f	t	t
f	f	f	t	t	t	t
f	f	t	f	f	t	t
f	f	t	f	t	t	t
f	f	t	t	f	t	t
f	f	t	t	t	t	t
f	t	f	f	f	f	f
f	t	f	f	t	f	f
f	t	f	t	f	f	t
f	t	f	t	t	f	t
f	t	t	f	f	f	t
f	t	t	f	t	f	f
f	t	t	t	f	f	t
f	t	t	t	t	f	t
f	t	t	t	t	f	t
t	f	f	f	f	t	t
t	f	f	f	t	t	t
t	f	f	t	f	t	t
t	f	f	t	t	t	t
t	f	t	f	f	t	t
t	f	t	f	t	t	t
t	f	t	t	f	t	t
t	f	t	t	t	t	t
t	t	f	f	f	f	t
t	t	f	f	t	f	t
t	t	f	t	f	t	t
t	t	f	t	t	t	t
t	t	t	f	f	t	t
t	t	t	f	t	t	t
t	t	t	t	f	t	t
t	t	t	t	t	t	t

8

happy	boy	has_football	impoverished	good_footballer	model of P	model of C
f	t	f	f	t	f	f
f	t	t	f	f	f	t
f	t	t	f	t	f	f
f	t	t	t	f	f	t
f	t	t	t	t	f	t
t	t	f	f	t	f	t

9

Our conclusions so far

- In this particular case, it happens to be true that $P \models C$
- Thus far we have a technique that derives new clauses that we would then have to check to see if they were logical consequences.
- Wouldn't it be nice if $P \models C$ if, and only if, $P \vdash C$?
- (Note that some models of C are *not* models of P. C is a logical consequence of P if it is true whenever P is true. It might be true at other times too.)

10

The two questions ...

- Is resolution *sound?*
  (Does it produce *only* clauses that are logical consequences?)
- Is resolution *complete?*
  (Does it produce *all* clauses that are logical consequences, given sufficient time?)

11

Soundness

- Consider the following two clauses taken from a program *P,* to which we're going to apply resolution:

```
A; A1,1; ...; A1,k:-B1,1, ..., B1,l
A2,1; ...; A2,k:-B2,1, ..., B2,m, A
```

- If an interpretation is a model for the program to which they belong, then it must be a model for both these clauses

12

Soundness (ctd.)

```
A; A1,1; ...; A1,k:-B1,1, ..., B1,l          (i)
A2,1; ...; A2,k:-B2,1, ..., B2,m, A          (ii)
```

has resolvent (ignoring elimination of duplicates):

```
A1,1; ...; A1,k; A2,1; ...; A2,k
  :-B1,1, ..., B1,l, B2,1, ..., B2,m
```

- If, in a model *M* of the program, *A* is true, then at least one of $B_{2,1},...,B_{2,m}$ must be false, or at least one of $A_{2,1},...A_{2,k}$ must be true, else it's not a model of clause (ii). So *M* is a model of the resolvent.
- Similarly if, in model *M* of the program, *A* is false, at least one of $A_{1,1},...A_{1,l}$ must be true, or at least one of $B_{1,1},...,B_{1,l}$ must be false, so *M* is a model of the resolvent.

13

Soundness (ctd.)

- In other words, we have shown that ...
- ... Every model of the clauses being resolved is also a model of the resolvent
- Therefore if $P \vdash C$ it is also true that $P \models C$
- Also, as each individual resolution step is sound, it follows that a *sequence* of them is too. So we can have a proof of more than one step.

14

Completeness

- Proving completeness will tend to be more difficult than proving soundness, and will prove particularly difficult for resolution because ...
- ... resolution isn't complete!
- It's easy to prove it isn't, by a counter-example.

15

Counter-example

- Consider the program:

  `a:-`

  and the tautology

  `a:-a`
- clearly you can't derive this clause from the above program.

16

But ...

- Resolution is *refutation complete,* i.e. we can use it to show that some particular clause *C* is a logical consequence of a program *P,* if it is.
- The idea is proof-by-refutation. Informally, we assume the opposite of what we are trying to prove, then show that this leads to a contradiction.

17

Refutation

- So suppose we want to prove that *C* is a consequence of *P,* and *C* is of the form:
  A_1; ...; A_n :- B_1, ..., B_m
- then *M* is a model of *C* if and only if it is *not* a model of:
  B_1 :-
  ...
  B_m :-
  :- A_1
  ...
  :- A_n
- (Because any model of *C* assigns at least one of the A's to true or one of the B's to false; similarly any model of the new set of clauses must assign all the B's to true and all the A's to false, so not a model of the old clause)

18

Proof by refutation

- To prove *C* is a logical consequence of a program *P,* 'negate' *C* and prove that *P* and the negated form of *C* are mutually inconsistent, i.e. have no common model.
- Note that the empty clause is a logical consequence of an inconsistent set of clauses ('every model' means none at all!)
- Conversely if □ is a logical consequence of a set of clauses, they must be inconsistent

19

Resolution is refutation complete

- If a set of clauses is inconsistent, it is always possible to derive the empty clause by resolution.
- (Proof outside scope of this course: basic idea is to define a measure 'excess literals', equal to the number of clauses subtracted from the number of literal occurrences; prove the assertion is true for *no* excess literals and prove that if it's true for *n* excess literals then it's true for *n+1,* i.e. inductively.)

20

Resolution example

- Suppose we have:

  dolphin:-

  read:-intelligent

  intelligent:-dolphin

  and want to prove:

  read:-
- negate the hypothesis:

  :-read

21

Resolution example (ctd.)

- And resolve:

  dolphin:- (i)

  read:-intelligent (ii)

  intelligent:-dolphin (iii)

  :-read (iv)
- Resolve (iv) and (ii):

  :-intelligent (v)
- Resolve (v) and (iii):

  :-dolphin (vi)
- Resolve (vi) and (i):

  □

22

Resolution can't always prove consistency

- If you try to prove that a program is inconsistent when it is, in fact, consistent, resolution can go on for ever. e.g. resolving

  :-a

  a:-a
- gives

  :-a
- ad nauseum
- Semi-decidable: derives proof tree in finite time if one exists, but there is no way to ensure that an attempt to derive a non-existent tree will halt and return 'no'. So we can write infinite Prolog loops!