1

Knowledge Based Systems

(CM0377)
Lecture 6

(last modified 20th February 2002)

2

Relational clausal logic

- Propositional clausal logic based on *propositions* (true or false)
- How to express this example?
  - Compaq is obliged to all its customers
  - Fred is one of Compaq's customers
  - Therefore Compaq is obliged to Fred
- Need to be able to talk about *individuals, sets* of individuals and *relations* between individuals - Relational Clausal Logic

3

Example

```
obliged_to(compaq, C):-
    customer_of(C, compaq).
customer_of(fred, compaq).

obliged_to(compaq, fred).
```

4

What we're aiming for ...

- Relational clausal logic
  is a step on the way to
- Full clausal logic
  upon which Prolog is based
- Specifically, Prolog is restricted to *definite clause logic*

5

At each stage, need to understand

- Syntax
- Semantics (giving meaning, among other things, to logical consequence)
- Proof theory (how does our chosen inference rule - resolution - apply?)
- Meta theory (is our inference rule sound and (refutation) complete?)

6

Syntax of relational clausal logic

- Constants
  - Single 'words' starting with lower case character, or arbitrary 'strings in single quotes'
- Variables
  - Arbitrary individuals, denoted by single 'words' starting with upper-case letter
- Terms
  - Constants, variables
- Ground terms
  - Terms without variables
- Predicates
  - Relations between individuals (same syntax as constants)
- Atoms
  - Predicates followed by terms inside brackets, sep. by commas
- Arguments
  - Terms between the brackets
- Arity
  - Number of arguments
- Ground atom
  - Atom without variables
- (See previous example for illustrations of these)

7

Semantics

- *Herbrand universe* of a program *P* is now the *set of ground terms* occurring in it
- *Herbrand base* of *P* is now the set of *ground atoms* that can be constructed using predicates in *P* and terms from the H. universe - representing all the things that could be true or false about relationships between individuals in the H. universe.

8

Example

```
obliged_to(compaq, C):-
      customer_of(C, compaq).
customer_of(fred, compaq).
```

- Herbrand universe is {`fred, compaq`}
- Herbrand base is:

```
{obliged_to(compaq, compaq),
 obliged_to(compaq, fred),
 obliged_to(fred, compaq),
 obliged_to(fred, fred),
 customer_of(compaq, compaq),
 customer_of(compaq, fred),
 customer_of(fred, compaq),
 customer_of(fred, fred)}
```

9

Semantics (ctd)

- A *Herbrand interpretation* is an assignment of elements of the H. base to **true** and **false**, as before; can just write the **true** ones, e.g.

```
{obliged_to(compaq, fred),
 customer_of(compaq, compaq)}
```

10

Semantics (ctd.)

- A *substitution* is a mapping of variables to terms. E.g. consider the clause:

```
obliged_to(compaq, C):-
    customer_of(C, compaq).
```

- Applying the substitution {`C -> fred`} gives:

```
obliged_to(compaq, fred):-
    customer_of(fred, compaq).
```

- Applying substitution {`C -> Y`} gives:

```
obliged_to(compaq, Y):-
    customer_of(Y, compaq).
```

- First example is a *ground instance* of the clause; the substitution was a *grounding substitution*.

11

Interpretations & models

- Reasoning with ground clauses identical to reasoning with propositional clauses (treating each term like a proposition).
- An interpretation is defined to be a model for a *non-ground* clause if it is a model for every ground instance.
- So to show

  M = {`obliged_to(compaq, fred), customer_of(fred, compaq)`}

  is a model of the clause from our program:

  `obliged_to(compaq, C):- customer_of(C, compaq)`

  need to construct all ground instances of the clause over the program's Herbrand universe i.e.

  {`obliged_to(compaq, fred):-customer_of(fred,compaq), obliged_to(compaq, compaq):-customer_of(compaq, compaq)`}

12

Logical consequence

- Remember that a clause *C* is a logical consequence of a program *P* if every model of *P* is also a model of *C*.
- This definition still applies with our extended definition of Herbrand bases, interpretations and models for relational clausal logic

13

Example

- Suppose our program *P* is:

```
obliged_to(compaq, C):-          (i)
  customer_of(C, compaq).
customer_of(fred, compaq).       (ii)
```

- and we want to prove that the following clause *C* is a logical consequence:

```
obliged_to(compaq, fred).
```

- Herbrand universe is {`compaq, fred`}

14

Example (ctd.)

- Herbrand base is:

```
{obliged_to(compaq, compaq),
 obliged_to(compaq, fred),
 obliged_to(fred, compaq),
 obliged_to(fred, fred),
 customer_of(compaq, compaq),
 customer_of(compaq, fred),
 customer_of(fred, compaq),
 customer_of(fred, fred)}
```

- Clause (ii) is already ground; set of ground instances of clause (i) is:

```
{obliged_to(compaq, fred):-customer_of(fred,compaq),
obliged_to(compaq, compaq):-customer_of(compaq, compaq)}
```

15

Example (ctd.)

- The models of clause (ii) are all those including `customer_of(fred, compaq)`
- The models of clause (i) are all those in which it is true simultaneously that:
  (`obliged_to(compaq, fred)` is true or `customer_of(fred, compaq)` is false)
  and:
  (`obliged_to(compaq, compaq)` is true or `customer_of(compaq, compaq)` is false)
- So models of *P* are all those in which `customer_of(fred, compaq)` is true, `obliged_to(compaq, fred)` is true, and
  (`obliged_to(compaq, compaq)` is true or `customer_of(compaq, compaq)` is false)
- We could enumerate these models, but note that we have deduced that every model of *P* includes `obliged_to(compaq, fred)`. But this means it's a model of *C*.

16

Proof theory

- Option 1
  - note that reasoning with ground clauses is identical to reasoning with propositional clauses, so apply all possible ground substitutions to every clause and then try to get the empty clause by resolution. Lots of clauses to choose from, though!!
- Option 2
  - Somehow derive the required substitutions to lead to a refutation from the clauses themselves

17

Unification

- Consider:

```
obliged_to(compaq, C):-  customer_of(C, compaq).
customer_of(fred, Manuf):-
   owns(fred, Obj), makes(Manuf, Obj).
```

- Can derive new clauses by applying a substitution {`C -> fred, Manuf -> compaq`} :

```
obliged_to(compaq, fred):- customer_of(fred, compaq).
customer_of(fred, compaq):-
   owns(fred, Obj), makes(compaq, Obj).
```

- Then resolving on `customer_of(fred, compaq)` to get a new clause:

```
obliged_to(compaq, fred):-
   owns(fred, Obj), makes(compaq, Obj).
```

18

Most general unifiers

- Consider:

```
obliged_to|compaq, C|:-     customer_of|C, compaq|.
customer_of|Pers, Manuf|:-
   owns|Pers, Obj|, makes|Manuf, Obj|.
```

- Can again derive

```
obliged_to|compaq, fred|:-  owns|fred, Obj|, makes|compaq, Obj|
```

by resolving on `customer_of` using the substitution {`C -> fred, Manuf -> compaq, Pers -> fred`}

- But could instead derive:

```
obliged_to|compaq, C|:-  owns|C, Obj|, makes|compaq, Obj|
```

by applying substitution {`Manuf -> compaq, Pers -> C`}

- The second resolvent is *more general* than the first; the second substitution (unifier) is more general than the first.
- A *most general unifier* is one that would *not* be a unifier if any substitutions were removed.
- *mgus* are unique, apart from variable namings.

19

A proof is ...

- ... showing that the set of clauses is inconsistent under the substitutions needed for unification of the literals. E.g.

```
obliged_to|compaq, C|:- customer_of|C, compaq|.
customer_of|Pers, Manuf|:-
   owns|Pers, Obj|, makes|Manuf, Obj|.
makes|compaq, pressario|.
owns|fred, pressario|.
```

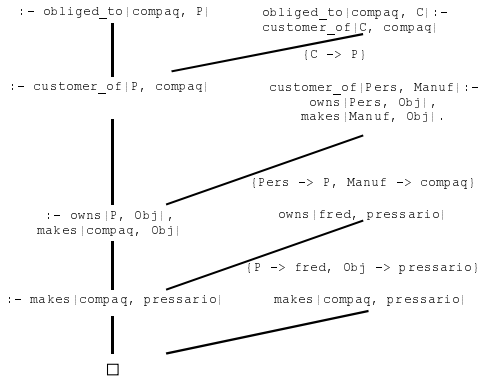
- to find out if Compaq is obliged to anyone, add the negation of this, i.e.

```
:-obliged_to|compaq, P|
```

- and try to refute it. The proof tree is a successful refutation under substitution {`P -> fred`} . This is an *answer* to the query.

20

Proof tree



21

Meta-theory

- It can be shown that relational resolution is:
  - sound
  - *not* complete
  - *refutation* complete
- Note that the Herbrand universe is always finite, therefore models are finite and there is a finite number of different models.
- So could answer 'is *C* logical conseq. of *P*' by enumeration, and the procedure would terminate. So relational clausal logic is decidable.