1

Knowledge Based Systems

(CM0377)

Lecture 13

(Last modified 2nd May 2002)

2

Default reasoning

- Consider:
  - Tweety is a bird
  - Normally birds fly
  - Therefore Tweety flies
- Could represent it thus:

```
bird(tweety).
flies(X):-bird(X), \+ abnormal(X).
```

- If this is the entire program, then we can derive `flies(tweety).`

3

Non-monotonicity

- If we add that Tweety is an ostrich and that ostriches are non-fliers, we have:

```
bird(tweety).
flies(X):-bird(X), \+ abnormal(X).
ostrich(tweety).
abnormal(X):-ostrich(X).
```

- Now we can't derive `flies(tweety)` any more. So we have *added* to a 'theory' and *removed* certain elements from the deductive closure of the theory.

4

Cancellation

- The *default rule* 'normally birds fly' is *cancelled* by a more *specific* rule (about ostriches).
- Problem with use of *not:*
  - Relying on *procedural* behaviour of *not.* To find out what the program does we have to run it; it doesn't have declarative semantics.

5

Alternative: distinct kinds of rule

- Distinguish between ordinary rules without exceptions and *default* rules, which could have exceptions. e.g.

```
default((flies(X):-bird(X))).
rule((\+ flies(X):-penguin(X))).
rule((bird(X):-penguin(X))).
rule((penguin(tweety):-true)).
rule((bird(opus):-true)).
```

6

Some goals

```
| ?- conclude(flies(X)).

X = opus ? ;

no
| ?- conclude(\+ flies(X)).

X = tweety ? ;

no
| ?-
```

A meta-interpreter

7

Conclude if you can prove it using non-default rules, or if there's a default rule that isn't provably false:

```
conclude(true):-!.
conclude((A, B)):-!, conclude(A), conclude(B).
conclude(A):-prove(A).
conclude(A):-
      default((A:-B)), conclude(B), \+ contradiction(A).

prove(true):-!.
prove((A, B)):-!, prove(A), prove(B).
prove(A):-rule((A:-B)), prove(B).

contradiction(\+ A):-!, prove(A).
contradiction(A):-prove(\+ A).
```

Another example

8

```
default((\+ flies(X):-mammal(X))).
default((flies(X):-bat(X))).
default((\+ flies(X):-dead(X))).
rule((mammal(X):-bat(X))).
rule((bat(dracula):-true)).
rule((dead(dracula):-true)).
```

- This says mammals typically don't fly; bats typically fly; ... anything that's a bat is definitely a mammal; dracula is definitely a bat; ...
- But can prove both that he does fly (because he's a bat and typically they fly) and that he doesn't (2 explanations: he's a mammal and mammals typically don't fly; he's dead and dead things typically don't fly)
- So our naive approach still isn't quite satisfactory. More sophisticated techniques exist.

Abduction

9

- We won't cover abduction & induction in any detail, but ...
- Given a *Theory,* Abduction tries to find an *Explanation* for some *Observation,* i.e. find an *Explanation* such that:

  $Theory \cup Explanation \models Observation$

10

Abduction: Example

- E.g. if *Theory* is:

```
likes(andrew,S):-student_of(S,andrew).
likes(X,Y):-friend(Y,X).
```

- And we have observation:

```
likes(andrew,marion)
```

- Then possible *Explanations* include:

```
{friend(marion,andrew)},
{student_of(andrew,marion)}
```

11

Induction

- Given a *Theory*, induction tries to find a *Hypothesis* that explains a set of *Examples,* i.e. find *Hypothesis* such that:

  $Theory \cup Hypothesis \models Examples$

- Main difference between this and abduction is that *Hypothesis* is allowed to be a set of clauses, rather than just a set of ground facts.

12

Induction: Example

- Given the following *Theory:*

```
bird(tweety).          bird(polly).
has_feathers(tweety). has_beak(polly).
```

- and the following *Examples*:

```
flies(tweety).         flies(polly).
```

- we might find the *Hypothesis:*

```
flies(X):-bird(X).
```

- (because everything that can fly in this particular example is a bird).

13

Finding explanations and hypotheses

- Finding *explanations* (for abductive reasoning) and *hypotheses* (for inductive reasoning) is obviously not necessarily straightforward.
- You'll be relieved to hear that although you should understand the general idea of abduction and induction, details of how to actually do these two things computationally is outside the scope of the present course.

14

Further reading

- See Flach pp. 147-154, if interested, for:
  - further background explanation
  - a version of the interpreter given here, but which builds an *explanation* of its conclusions in a list
  - a way of dealing with the problem of conflicting conclusions from defaults (attaching names to defaults)