CSE571 SCRIBE NOTES

Andrew Davis

September 22, 2003

We began with the problem from the previous class.

Normally you say:

$anc(X, Y) \leftarrow par(X, Y).$
$anc(X, Y) \leftarrow par(X, Z), anc(Z, Y).$

$par(a, b).$
$par(b, c).$
$par(d, e).$

Suppose we have

$par(a, b).$
$par(b, c).$
$\neg par(a, c).$
$\neg par(b, c).$

We want to be able to define $\neg anc(X, Y) \leftarrow$ _____.

So we could define

$$m\_par(X, Y) \leftarrow \textbf{not } par(X, Y), \textbf{not } \neg par(X, Y).$$

But we don't want to exclude those whom we know to be parents. so it is better to say

$$m\_par(X, Y) \leftarrow \textbf{not } \neg par(X, Y).$$

So then we can define

$$m\_anc(X, Y) \leftarrow m\_par(X, Y).$$
$$m\_anc(X, Y) \leftarrow m\_par(X, Z), m\_anc(Z, Y).$$

So then someone is not an ancestor if it's not possible for them to be maybe an ancestor.

$$\neg anc(X, Y) \leftarrow \textbf{not } m\_anc(X, Y).$$

An issue was raised that this modified (translated) program was incomplete, because it was possible to have such things as

$par(a, b).$
$m\_par(b, a).$

when we know intuitively that if b is a's parent then a cannot be b's parent.

However, the translated program need not fit with out intuitive notions of what to expect from parents and ancestors. What is necessary is that it be mathematically consistent with the definition of parents and ancestors in the original program.

For example, the original program would allow

$par(a, b).$
$par(b, a).$

So it is not necessary that this be prohibited in the translated program.

Some More Examples...

Basic rules:
Birds *normally* fly.
Tweety is a bird.
Sam is a bird.

$fly(X) \leftarrow bird(X), \textbf{not}\ ab(X).$
$bird(tweety).$
$bird(sam).$

Now we learn something more:
Penguins are birds.
Penguins are exceptional with respect to flying.
Sam is a penguin.

$bird(X) \leftarrow penguin(X).$
$ab(X) \leftarrow penguin(X).$
$penguin(sam).$

(Note that after saying $penguin(sam)$ we may remove the rule $bird(sam)$ from the first set of rules.)

Now we can conclude that *tweety* flies, but *sam* doesn't.

Suppose now we don't want to infer by default, but instead we want special rules to tell when something doesn't fly. We could add:

$\neg fly(X) \leftarrow \textbf{not } fly(X).$
$\neg bird(X) \leftarrow \textbf{not } bird(X).$
$\neg ab(X) \leftarrow \textbf{not } ab(X).$
$\neg penguin(X) \leftarrow \textbf{not } penguin(X).$

Now we want to add:
Wounded birds are birds.
Wounded birds are *weakly exceptional* to flying.
John is a wounded bird.

$bird(X) \leftarrow wounded\_bird(X).$
$wounded\_bird(john).$

We could write

$$ab(X) \leftarrow wounded\_bird(X).$$

Then we would need to remove the rule

$$\neg fly(X) \leftarrow \textbf{not } fly(X).$$

and then add

$$\neg fly(X) \leftarrow \textbf{not } fly(X), \textbf{not } wounded\_bird(X).$$

wounded_bird is not a normal exception. It is a special kind of exception. Exceptions block the positive. Weak exceptions block both the positive and the negative.

Suppose that we have

$bird(tweety)$.
$\neg bird(sylvester)$.

This behaves correctly, because we can conclude $\neg fly(sylvester)$.

But suppose that we also have et? We would conclude $\neg fly(et)$, because et is not a bird.

Computer languages are different from natural languages. With natural languages, you can say whatever you want and then change what you said completely in the next paragraph. With computer languages you don't get that degree of flexibility.

If we replace

$$\neg fly(X) \leftarrow \textbf{not } fly(X), \textbf{not } wounded\_bird(X).$$

with

$$\neg fly(X) \leftarrow \neg bird(X).$$

then we still don't know about et. So how do we define $\neg bird$? When information is incomplete, certain things must be done. When we have a bunch of predicates, there are two types:
– Derived Predicates – e.g. $anc, fly, bird, ab$
– Base Predicates – e.g. $par, penguin, wounded\_bird$

Life is simpler when you just have base predicates. Derived predicates make things more confusing. If facts are about base predicates only, then it is straightforward. If we have observations about derived predicates, then more advanced techniques are necessary.

Derived predicates are written in terms of base predicates and other derived predicates.

Starting over back at the beginning of the problem, we can express the rule that "normally birds fly" with

$$fly(X) \leftarrow bird(X), \mathbf{not}\ \neg fly(X).$$

So a bird will fly unless we have contrary evidence that it doesn't fly.

Also note that fly is derived and $bird$ is base.

Going back to an earlier instance of the program

$fly(X) \leftarrow bird(X), \textbf{not } ab(X).$
$bird(tweety).$
$bird(et).$

$bird(X) \leftarrow penguin(X).$
$ab(X) \leftarrow penguin(X).$
$penguin(sam).$
$\neg penguin(tweety).$

We don't know about *et*. It may or may not be a penguin.

Using a strategy similar to the one discussed at the beginning of class – where we used m_ to indicate that *maybe* X was a *par* of Y – we can add the rule

$ab(X) \leftarrow \textbf{not } \neg penguin(X).$

We want to be conservative in what we assume flies.

In general, when writing rules, start with normative statements such as the ones that begin "normally..."

Then proceed to weak exceptions.

2 other issues
– most programs are written with closure assumption in mind.
– when you write rules, what kinds of observations will you have to deal with?

The important issue to keep in mind is: How do you assimilate observations with existing knowledge?