KNOWLEDGE REPRESENTATION, REASONING AND DECLARATIVE PROBLEM SOLVING

By Jicheng Zhao
based on the notes of others

Class 8: More Modules for Declarative Programming with Answer sets

Close world assumption and open world assumption

- Problem: Consider the following program:

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

and a set of facts:

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

Now we want to define $\neg anc(X,Y)$ in terms of $\neg par(U,V)$.

Now, the negative literals are allowed. The facts are of the form:

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

- Solution:

– The basis is to define the $m\_par(X,Y)$ to denote that X may be a parent of Y.

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

But we don't want to exclude those whom we know to be parents.

– With the $m\_par(U,V)$, we can define $m\_anc(X,Y)$ to denote that X may be a ancestor of Y:

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

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

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

- Discussion:

  – Question: Is this translation correct? For example, given the facts

$$par(a,b).$$

Since we do not know $\neg par(b, a)$, we may draw a conclusion that $m\_(b, a)$. when we know intuitively that if b is a's parent then a cannot be b's parent.

– Answer: However, the translated program need not fit with out intuitive notions of what to expect from parents and ancestors. For example, the original program would allow

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

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

Given a program Π of defining predicate p, we say that the program $Tr(\Pi)$ for the definition of $\neg p$ is correct if the following condition holds.

The program Π is based on the set of atoms D. $Tr(\Pi)$ is based on the $D' = ext(D)$. $D' = ext(D)$ as long as $D' \supseteq D$ and D' is consistent. D' is a complete extension $c\_ext(D)$ of D if $D' = ext(D)$ and all atoms are either true or false in D'.

The translation $Tr(\Pi)$ is correct if

$$Tr(\Pi) \cup D = \bigcap_{D'=c\_ext(D)} (\Pi \cup D')$$

– How can we handle it if the constraint

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

is added into the initial program?

Weak Exception and Missing information

1. "Birds *normally* fly, Tweety is a bird, Sam is a bird" can be presented as a program:

$$\begin{aligned} fly(X) &\leftarrow bird(X), \mathbf{not}\ ab(X). \\ bird(tweety). & \\ bird(sam). & \end{aligned}$$

2. Now we know something more: "Penguins are birds, penguins are exceptional with respect to flying, and Sam is a penguin." These can be encoded as the rules as follows:

$$\begin{aligned} bird(X) &\leftarrow penguin(X). \\ ab(X) &\leftarrow penguin(X). \\ penguin(sam). & \end{aligned}$$

   Now we can conclude that *tweety* flies, but *sam* does not.

3. 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:

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

4. Now we want to add: "Wounded birds are birds. Wounded birds are *weakly exceptional* to flying. John is a wounded bird.". wounded birds are weakly exceptional, that means that they may or may not fly.

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

To present the "weakly exceptional", we could write

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

Then we would need to replace the rule

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

by

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

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

$$\neg bird(sylvester).$$

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

5. Now suppose that we also have et. We do not know whether et is a bird or not. In the previous program, we would conclude $\neg fly(et)$, because $bird(et)$ is not true.

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.

By replacing

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

with

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

the problem of et can be solved. After this replacement, since we do not have $\neg bird(et)$, we cannot conclude $\neg fly(et)$. However, we cannot conclude $\neg fly(sam)$ in this case.

We can add a rule

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

Again, we do not know whether et is a penguin or not, but the program will conclude that $penguin(et)$.

When information is incomplete, certain things must be done. When we have a bunch of predicates, there are two types:

5.1. Derived Predicates – e.g. $anc, fly, bird, ab$.

5.2. 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.

Using a strategy similar to the one discussed at the beginning of class – where we used $m\_par(X, Y)$ to indicate that *maybe* X was a *par* of Y.

The program we have for this problem can be:

$$bird(tweety).\;\; penguin(sam).\;\; wounded\_bird(john). \tag{1}$$

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

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

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

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

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

$$fly(X) \leftarrow bird(X), not\ \neg fly(X), not\ wounded\_bird(X). \tag{2}$$

6. two other issues:

   6.1. Most programs are written with closure assumption in mind.

   6.2. 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?