CAP 5602
Summer, 2011

Lesson 2: Satisfying Goals

The topics

1. matching
2. goals
3: multiple goals

1. Matching

Before we go on to describe **matching** let us define **substitutions**. A substitution is a finite set of pairs $s = \{V_1 = t_1, \ldots, V_n = t_n\}$, where $V_1, \ldots, V_n\}$ are distinct variables and $t_1, \ldots, t_n$ are data objects. We say that s **instantiates** V_1 to t_1, V_2 to t_2, ..., V_n to t_n.

We can apply a substitution $s = \{V_1 = t_1, \ldots, V_n = t_n\}$ to a a term t by simultaneously replacing every occurrence of V_1 in t by t_1, every occurrence of V_2 by t_2 , and so on. So, a substitution changes only the variables that occur to the left of the = sign in the body of the substitution.

For example, let us apply $s =$ { X = a, Y = Z} to t = f(X,Y,g(U,X,b)). We replace every X in t by a and every Y by Z and get f(a,Z,g(U,a,b)).

Now, let us describe matching. Let t_1 and t_2 be two data objects. We say that t_1 and t_2 **match** if there is a substitution that makes them identical.

For example, t_1 =address(X,'Elms Street', 'Treetown') and t_2 = address(23, A_Street, A_Town) match because the substitution { X=23, A_Street = 'Elms Street', A_Town = 'Treetown' } makes both objects identical to the string address(23, 'Elms Street', 'Treetown').

On the other hand, t_1=parent(X,Y) and t_2 =father(Y) do not match because whatever substitution we choose, the functor of t_1 remains parent and the functor of t_2 is father.

Now, let's take a closer look at matching. Take t_1 =address(My_Number, My_Street, 'Treetown') and t_2 =address(23, A_Street, A_Town). The substitution $s =$ { My_Number = 23, My_Street = A_Street = 'Elms Street', A_Town = 'Treetown'} makes the two terms identical to address(23, 'Elms Street', 'Treetown').

But so does the substitution m = { My_Number = 23, My_Street = A_Street, A_Town = 'Treetown'} that produces the common string address(23, A_Street , 'Tree Town'). The substitution m is **more general** than s because s is an instance of m, obtained by assigning 'Elms Street' to the variable A_Street.

It can be shown, see Pelin for example, that whenever the data objects t_1 and t_2 match, there is a **most general substitution**, such that every match is an instance of that substitution.

Prolog first relabels the variables of t_1 and t_2 such that they have no common variables and then matches using the most general substitution.

Now, let us spell out the matching rules for t_1 and t_2.

1. If both t_1 and t_2 are constants (atoms or numbers) they match iff they are identical.

The exception here is that the single quotes around an atom do not count, so the atoms 'tree' and tree match.

2. If t_1 is a variable, then t_1 and t_2 match and t_1 is instantiated to t_2. Conversely, if t_2 is a variable, t_1 and t_2 match and t_2 is instantiated to t_1.

3. If t_1 and t_2 are structures they match only if they have the same functor, the same arity (number of arguments) and the components match pairwise. The resulting substitution is given by the matching of the components.

Examples of Matching

1. The objects $point(X, 3)$ and $point(5, Z)$ match because they have the same functor *point*, the same arity (2), the first arguments X and 5 are matched by $X = 5$, and the second arguments, 3 and Z are matched by $Z = 3$. So, the substitution $X = 5, Z = 3$ is a match for $point(X, 3)$ and $point(5, Z)$.

2. Using the same reasoning we get that the substitution $U = 5, Y = 6$ is a match for the objects $point(5, Y)$ and $point(U, 6)$.

3. The objects $point(0, 2)$ and $point(V, 2)$ match because they have the same functor, the same arity, and the components are matched by the substitution $V = 0$.

4. The terms $t_1 = triangle(point(X, 3), point(5, Y), point(0, 2))$ and $t_2 = triangle(point(5, Z), point(U, 6), point(V, 2))$ match because they have the same main functor *triangle*, the same arity 3, and the components are pairwise unifiable. The substitution $X = 5, Z = 3, U = 5, Y = 6, V = 0$ is the match.

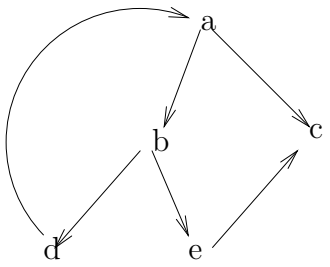

Figure 1: The graph for prog1

More Examples of Matching

1. The terms $point(X, X)$ and $point(3, 5)$ do not match. The terms have the same functor *point* and the same arity 2, and the first components, X and 3 match via the substitution $X = 3$. However the second components, X and 5, do not match, because X is no longer an uninstantiated variable. It has the value 3 and 3 does not match 5.

2. The terms a and $a(b, X)$ do not match because the first is an atom and the second a structure.

2. Goals

Let us consider the program prog1.pl shown below that describes the graph from Figure 1.

```
% the arc clauses
arc(a,b).          % fact 1
arc(a,c).          % fact 2
arc(b,d).          % fact 3
arc(b,e).          % fact 4
arc(d,a).          % fact 5
arc(e,c).          % fact 6
```

The program has 6 clauses. For debugging purposes, we enclosed the clause number as a comment.

When we run prolog, the interpreter will first prompt us with the ?− sign. We must enter a command. Most likely, the first command will be to compile one or more files. We can do this with the built-in predicate consult that compiles the clauses of the file prog1, or whatever file we may choose.

?− consult(prog1).

In SWI prolog we can click on the File icon and then select Consult. Then we can navigate the directory to select the desired file.

In either case, the interpreter will compile the file and translate the clauses that have no syntax errors. This way we can run a program even if it has syntx errors!

In our case prolog will print,

and prompt us for the next command.

The translated clauses form the **database**. Let us say that we want to find out if the fact arc(b,d) is in the database.

We type arc(b,d) followed by a period after the ?− prompt and press return.

?- arc(b,d).

The computer will set arc(b,d) as its **goal** and try to satisfy it by matching the goal with the clauses in the database, starting with clause 1. Since all clauses are facts, we apply the matching rules listed in the first section. The goal cannot be matched with fact 1 because the first argument of the fact, a, is different from the first argument, b, of the goal. Then, prolog tries to match the goal with the second fact. Again, the first argument of the fact is not identical to the first argument of the goal, so the matching fails. The interpreter tries to match the goal with the third clause and this time it succeeds. It prints,

true

and waits for our input. If we press the return key, the interpreter will answer with ?− prompt. Some prolog interpreters use yes and no instead of true and false. Now let us enter the command

?-arc(a,e).

The computer will try to match the goal arc(a,e) with the facts 1,2,3,4,5, and 6. Each time the matching will fail. When there are no more clauses, the interpreter will write

false.

and prompt us for the next command.

Now let us use a goal that uses variables. For example, we may want to know the target of the arcs that start in the vertex a.

?-arc(a,X).

Again, the interpreter tries to match the goal arc(a,X) with the clauses in the database, starting with clause 1. The substitution {X=b} matches arc(a,b) and arc(a,X) and the computer writes

X = b

This time we type a semicolumn and then press the return key. This tells the interpreter to continue matching the goal with the remaining clauses. So, the interpreter tries to match arc(a,X) with fact 2. The substitution {X} = c is such a match. The SWI compiler will print

X = c.

and prompt us with ?− sign. The period at the end of the substitution means that there are no more matches.

3. Multiple Goals

Now let us compile the program prog2 shown below.

```
likes(john,mary).          % fact 1
likes(bill, susan).        % fact 2
likes(susan,bill).         % fact 3
likes(mark,ann).           % fact 4
likes(mary,bill).          % fact 5
likes(ann,mark).           % fact 6
```

Now, let us ask the query below.

?- likes(X,Y),likes(Y,X).

This query has 2 goals, linked by the comma.

First prolog tries to satisfy the first goal, starting with clause 1. If the first goal is not satisfied, prolog writes false and finishes. If the first goal is satisfied, then prolog tries to satisfy the second goal starting with clause 1. If it succeeds, it displays the substitution. If not, it repeats the loop: try to resatisfy the first goal starting from the clause that follows the last match. If unsuccessful, write false and exit. If successful, try to satisfy the second goal. If this is also satisfied, display the matching substitution, else go back to the loop.

Let us see how this works for our database. First, goal 1, likes(X,Y), is unified with clause 1. We get X = john, Y = mary. Then we try to satisfy goal 2, likes(mary,john), starting with clause 1. None of the 6 facts can be matched with goal 2, so prolog backtracks and tries to resatisfy goal 1. So, goal 1, likes(X,Y), is matched with fact 2. We get X = bill, Y= susan. Prolog tries to satisfy goal 2, likes(susan,bill) starting with fact 1. The goal is matched with fact 3 and prolog writes

X = bill, Y= susan

If we type ; and then press return, prolog continues. It tries to unify the second goal, likes(susan, bill), with facts 4, 5, and 6 and fails. So, it backtracks and resatisfies goal 1 with fact 3. We get X = susan, Y = bill. Goal 2 is now likes(bill, susan) and is satisfied by fact 2. So, prolog writes

X= susan, Y = bill

If we again type ; followed by the return, prolog continues. It tries to satisfy the second goal, likes(bill, susan), with facts 3,4,5,6 and fails. So, it resatisfies goal 1 with fact 4. We get X = mark, Y = ann. Goal 2 becomes likes(ann,mark) and it is satisfied by fact 6. So, prolog writes

X = mark, Y = ann

We can end the prolog session with the built-in predicate halt.

?-halt.