

Tracing in Prolog

Diane Horton

An example

Suppose we have this database of prolog facts and rules:

```

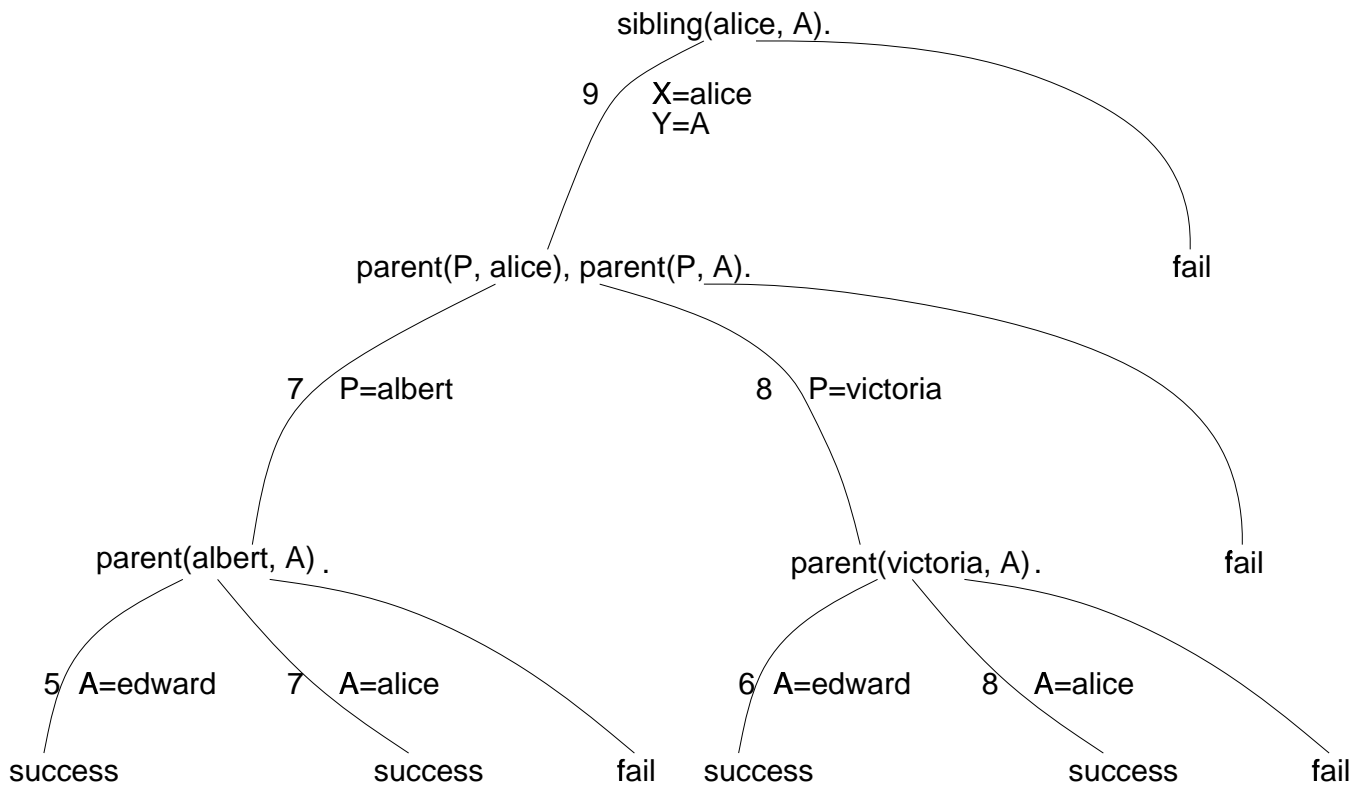
1  male(albert).
2  female(alice).
3  male(edward).
4  female(victoria).
5  parent(albert,edward).
6  parent(victoria,edward).
7  parent(albert,alice).
8  parent(victoria,alice).
9  sibling(X,Y):- parent(P,X), parent(P,Y).

```

(The facts and rules are numbered here for convenient reference. Such numbers aren't allowed by Prolog.) If we ask this query:

```
| ?- sibling(alice,A).
```

and use “;” to get all possible answers, here is a trace of what happens:



Working through the example

Let's work through the trace one step at a time. My description is in great and gory detail. Once you get the hang of doing these traces, you'll be able to whip them up without this much tedious thought.

- We have given prolog a single thing to make true (if it can): `sibling(alice, A)`. This is the root of the tree.
- Prolog tries to prove this is true by starting at the beginning of its database of facts and rules. Item 1 is a fact, `male(albert)`. This doesn't match the goal `sibling(alice, A)`. Neither do any of facts 2 through 8. But item 9 is a rule, and its head, `sibling(X,Y)` does match `sibling(alice, A)` as long as `X=alice` and `Y=A`. So we draw a branch from the root and label it with 9 to show that rule 9 was our match, and we also label it with `X=alice` and `Y=A` to show that these bindings make the match possible.
- Now rule 9 doesn't tell us that `sibling(alice, A)` is actually true; it just tells us that it is true if we can prove all the goals on the right hand side of the rule: `parent(P,X)`, `parent(P,Y)`. So these goals remain to be proven. And by now we have bound `X=alice` and `Y=A`, so in fact what we have to prove is `parent(P,alice)`, `parent(P,A)`. So we draw that as the node along the branch we just added.
- Now Prolog tries to satisfy these two remaining goals, starting with the first of them, `parent(P,alice)`. As always, it starts at the beginning of its database of facts and rules. The first thing to match the goal `parent(P,alice)` is fact 7, `parent(albert,alice)`. It matches if `P=albert`, so we draw a branch labelled 7 and `P=albert`,
- We still have the second goal left to prove: `parent(P,A)`. By now we have bound `P` to `albert`, so in fact what we have to prove is `parent(albert, A)`. So we draw that as the child node along the branch we just added.
- Now Prolog tries to satisfy this one remaining goal, starting, as always, at the beginning of its database. The first item to match is fact 5: `parent(albert, edward)`. It matches as long as `A=edward`, and it leaves us with no goals to prove. So we draw a branch labelled 5 and `A=edward`, and label the node on this branch "success".
- Prolog has now successfully proven our initial goal `sibling(alice, A)`. The entire path from the root to our success leaf represents the steps taken and the bindings made along the way. In effect, we have proven that alice is a sibling of edward, by virtue of their common parent albert.
- We made a number of bindings along the way. The bindings to `X`, `Y`, and `P` refer to variables that are local to individual rules (rule 9), and so they would not be reported to the user. But the binding `A=edward` refers to a variable in the initial query, so it would be reported.
- If the user were to type a carriage return, that would be the end of things. But suppose we were now to type a ";", asking Prolog to find another way to prove our goal. Prolog would back up to the parent of the node we left off at. That parent node is `parent(albert, A)`. Prolog then tries to find *another* way to satisfy that goal. If it can, it will have found another way to make the initial goal true.

- Prolog has already tried using items 1 through 5 in the database. (You can see this is the case by looking at the “5” label on the branch to the success node.) So it tries again starting with item 6. It finds a successful match with item 7, as long as **A=alice**. This leaves us with no goals to prove. So we draw a branch labelled 7 and **A=alice**, and label the node on this branch “success”.
- Prolog has now successfully proven our initial goal a second way. It has proven that alice is the sibling of alice, by virtue of their common parent albert. Now this is a silly conclusion, but that’s not Prolog’s fault. The sibling rule never says that in order for X and Y to be siblings they must be different people! We’ll learn later how to avoid this sort of thing.
- If we type “;” again, Prolog will back up once more to the `parent(albert, A)` node, and try to find another way to satisfy it, starting from item 8 in the database. Nothing else in the database matches, so we draw a new branch and label it “fail”. But there are other things we can still try; Prolog backs up further in the tree to the “`parent(P,alice), parent(P,A)`” node and tries to find alternative ways to satisfy it, starting with the first goal, `parent(P,alice)`.
- The last way that it found to satisfy this used fact 7 (see the branch label that indicates this), so Prolog starts trying again with fact 8. Fact 8 works, if **P=victoria**. So we have a new branch, and a new subgoal, `parent(victoria, A)`.
- Prolog now tries to prove this new subgoal. Since it’s a new node, it begins at the top of the database and finds the first match is fact 6: `parent(victoria, A)` is true as long as **A=edward**. This leaves us with no goals, so we have a new branch to a success node. We have proven that alice is the sibling of edward, by virtue of their common parent victoria. This conclusion is redundant with our very first success. We’ll learn later how to avoid this sort of redundancy.
- If we type “;” again, Prolog will back up to the `parent(victoria, A)` node and try to find another way to satisfy it, starting with item 7 in the database (since we’ve tried items 1 through 6 already). Item 8 matches, leading to another success. This time we have proven that alice is the sibling of alice by virtue of their common parent victoria.
- Another “;” will cause Prolog to back up to the `parent(victoria, A)` node again, and try to find a way to satisfy it starting with item 9 in the database. It’s the last item and it doesn’t match, so we add a fail node. But we keep backing up, this time up a level to the “`parent(P,alice), parent(P,A)`”.
- Prolog tries to satisfy the node’s first goal, starting with rule 9 (having already tried rules 1 through 8). But it fails to find a match. So it backs up even further, now to the very root of the tree. It has used items 1 through 9 of the database trying to satisfy `sibling(alice, A)`, and there is nothing after that to try. So we add another fail node. Since we have failed at the root, there is nowhere further to back up. We have found every possible way to prove `sibling(alice, A)`, and since there are no more, Prolog reports to the user “fail”.