The process of deduction

Overview

What is the Prolog interpreter doing during the process of deduction (replying to a query or searching to achieve a goal)?

1. A Prolog program is consulted and the facts and rules in the program are loaded into the knowledge database.
2. When a query or goal is given, the Prolog interpreter seaches the knowledge database from top to bottom to find any facts or rules match with the query or goal.
3. If a match is found, the search for answers to the query succeeds or the goal is true. The values of any variables found are given.
4. If the user asks for further solutions (e.g by entering a semicolon in some Prolog interpreter), the Prolog interpreter continues searching. It keeps track of the facts and the rules which have been searched. When it can find no more solutions, it says `no` and the search ends.

A detailed description of Step 2

The Prolog interpreter starts with the goal/query and works backward by identifying the facts and the rules (in the knowledge database) from which the goal can be derived. This can be done by the expansion of a goal by applying a rule and the reduction of a goal by applying a fact. You will see the process of deduction in the example below.

Unification

Unification is the derivation of a new rule from a given rule through the binding of variables.

In order to match a goal or a query, any variable encountered is substituted with the value of an appropriate constant (called binding).

Backtracking

Sometimes there are more than one fact/rule which can be applied to a goal/query. In this case, the fact/rule which appears first in the knowledge database will be applied first. The other applicable facts or rules will be applied later such that all possible solutions are found. This process is known as backtracking.

Examples

We will use the following program and some queries to explain the process of deduction.

Note: In each step of the process of deduction, there may be more than one subgoals to be processed. In this case, only the first subgoal is considered. If there is a rule that can be applied to it, it is replaced by those subgoals found in the applied rule. If it satisfies a fact, it is removed. If no rules or facts can be applied, we have to backtrack.

Program 4: A program about some computers, installed software and some users

```
spec(comp1, pc, 32).                                   /* Fact  1 */
spec(comp2, mac, 128).                                 /* Fact  2 */
spec(comp3, pc, 64).                                   /* Fact  3 */
runs(pc, movie_edit, 96).                              /* Fact  4 */
runs(pc, vb, 16).                                      /* Fact  5 */
runs(pc, cpp, 28).                                     /* Fact  6 */
runs(mac, vb, 24).                                     /* Fact  7 */
runs(mac, prolog, 128).                                /* Fact  8 */
access(judy, comp1).                                   /* Fact  9 */
access(peter, comp3).                                  /* Fact 10 */
access(david, comp1).                                  /* Fact 11 */
access(david, comp2).                                  /* Fact 12 */

can_use(P, SW) :- access(P, Comp), can_run(Comp, SW).  /* Rule  1 */

can_run(Comp, SW) :- spec(Comp, CompType, MemAvail),
                     runs(CompType, SW, MemNeeded),
                     MemAvail >= MemNeeded.            /* Rule   2 */
```

Can Judy use VB?

```
?- can_use(judy, vb).
```

Level	Results of deduction	Rules or facts to be applied	Bindings of variables
Original goal	`can_use(judy, vb).` *In order that the original goal (`can_use(judy, vb)`) is true, according to Rule 1, the conditions in the rule must be true (i.e. `access(judy, Comp)` and `can_run(Comp, vb)` for some `Comp`). Therefore the goal can be reduced as shown below. In order to apply Rule 1 to the goal, a unification is done by binding (i.e. assigning) the value `judy` to the variable `P` and binding the value `vb` to the variable `SW`.*	Rule 1	`P` = `judy` `SW` = `vb`
1	`access(judy, Comp), can_run(Comp, vb).` *After unification, `access(judy, comp1)` is found to be true (from Fact 9). Therefore, whether the original goal is true or not depends on only whether `can_run(comp1, vb)` is true or not.*	Fact 9	`Comp` = `comp1`
2	`can_run(comp1, vb).`	Rule 2	-
3	`spec(comp1, CompType, MemAvail),` `runs(CompType, vb, MemNeeded),`	Fact 1	`CompType` = `pc`

	MemAvail >= MemNeeded.		MemAvail = 32
4	runs(pc, vb, MemNeeded), 32 >= MemNeeded.	Fact 5	MemNeeded = 16
5	32 >=16. *The above is true, and so we can deduce that subgoals at level 4 is true, and hence those at levels 3, 2, 1 and the original goal.*	Satisfied! **Output:** yes	
Overall output	yes		

Can David use Prolog?

`?- can_use(david, prolog).`

This goal demonstrates backtracking.

Level	**Results of deduction**	**Rules or facts to be applied**	**Bindings of variables**

Original goal	can_use(david, prolog).	Rule 1	P = david
			SW = prolog

1	access(david, Comp), can_run(Comp, prolog). *Both Fact 11 and Fact 12 are applicable. Since Fact 11 comes before Fact 12, Fact 11 is tried first.*	Fact 11	Comp = comp1
2	can_run(comp1, prolog).	Rule 2	-

3	spec(comp1, CompType, MemAvail),
	runs(CompType, prolog, MemNeeded),
	MemAvail >= MemNeeded.
		Fact 1	CompType = pc
			MemAvail = 32

4	runs(pc, prolog, MemNeeded), 32 >= MemNeeded. *No rules or facts can further be applied. Backtracking is done here.*	Failed! Backtracks.	
3	spec(comp1, CompType, MemAvail), runs(CompType, prolog, MemNeeded), MemAvail >= MemNeeded. *There is no applicable rules or facts that were not applied to these subgoals before. So backtracking has to be done.*	Failed! Backtracks.	
2	can_run(comp1, prolog).	Failed! Backtracks.	
1	access(david, Comp), can_run(Comp, prolog). *Fact 11 has been used. Now we try Fact 12.*	Fact 12	Comp = comp2
2	can_run(comp2, prolog).	Rule 2	-

3	spec(comp2, CompType, MemAvail),
	runs(CompType, prolog, MemNeeded),
	MemAvail >= MemNeeded.
		Fact 2	CompType = mac
			MemAvail = 128

4	`runs(mac, prolog, MemNeeded),`
	`128 >= MemNeeded.`
		Fact 8	`MemNeeded` = `128`

5	`128 >= 128.`	Satisfied! **Output:** `yes`	
Overall output	`yes`		

What software can Judy use?

`?- can_use(judy, X).`

This query demonstrates how backtracking is done to find all answers.

Level	**Results of deduction**	**Rules or facts to be applied**	**Bindings of variables**
Original query	`can_use(judy, X).`	Rule 1	`P` = `judy`
1	`access(judy, Comp), can_run(Comp, X).`	Fact 11	`Comp` = `comp1`
2	`can_run(comp1, X).`	Rule 2	-

3	`spec(comp1, CompType, MemAvail),`
	`runs(CompType, X, MemNeeded),`
	`MemAvail >= MemNeeded.`
		Fact 1	`CompType` = `pc`
			`MemAvail` = `32`

4	`runs(pc, X, MemNeeded),`
	`32 >= MemNeeded.`
		Fact 4	`X` = `movie_edit`
			`MemNeeded` = `96`

5	`32 >= 96.`	Failed! Backtracks.	

4	`runs(pc, X, MemNeeded),`
	`32 >= MemNeeded.`
		Fact 5	`X` = `vb`
			`MemNeeded` = `16`

5	`32 >= 16.` *`X = vb` is a solution to the original query.* *Backtracking is still peroformed because there may be other solutions.*	Satisfied! **Output:** **`X = vb`** Then backtracks.	

4	`runs(pc, X, MemNeeded),`
	`32 >= MemNeeded.`
		Fact 6	`X` = `cpp`
			`MemNeeded` = `28`

5	`32 >= 28.`	Satisfied! **Output:** **`X = cpp`** Then backtracks.	

4	`runs(pc, X, MemNeeded),`
	`32 >= MemNeeded.`
		Failed! Backtracks.	

3	`spec(comp1, CompType, MemAvail),`
	`runs(CompType, X, MemNeeded),`
	`MemAvail >= MemNeeded.`
		Failed! Backtracks.	

2	`can_run(comp1, X).`	Failed! Backtracks.	
1	`access(judy, Comp), can_run(Comp, X).`	Failed! Backtracks.	

Original query	`can_use(judy, X).`	Failed! Since the top level is reached, all answers are found.	

Overall output	`X = vb`
	`X = cpp`		

Who can use MovieEdit?

`?- can_use(X, movie_edit).`

Backtracking has to be performed before concluding that there is no answer to a query.

Level	**Results of deduction**	**Rules or facts to be applied**	**Bindings of variables**
Original query	`can_use(X, movie_edit).`	Rule 1	`SW` = `movie_edit`

1	`access(X, Comp), can_run(Comp, movie_edit).`	Fact 9	`X` = `judy`
			`Comp` = `comp1`

2	`can_run(comp1, movie_edit).`	Rule 2	-

3	`spec(comp1, CompType, MemAvail),`
	`runs(CompType, movie_edit, MemNeeded),`
	`MemAvail >= MemNeeded.`
		Fact 1	`CompType` = `pc`
			`MemAvail` = `32`

4	`runs(pc, movie_edit, MemNeeded),`
	`32 >= MemNeeded.`
		Fact 4	`MemNeeded` = `96`

5	`32 >= 96.`	Failed! Backtracks.	

4	`runs(pc, movie_edit, MemNeeded),`
	`32 >=MemNeeded.`
		Failed! Backtracks.	

3	`spec(comp1, CompType, MemAvail),`
	`runs(CompType, movie_edit, MemNeeded),`
	`MemAvail >= MemNeeded.`
		Failed! Backtracks.	

2	`can_run(comp1, movie_edit).`	Failed! Backtracks.	

1	`access(X, Comp), can_run(Comp, movie_edit).`	Fact 10	`X` = `peter`
			`Comp` = `comp3`

2	`can_run(comp3, movie_edit).`	Rule 2	-

3	`spec(comp3, CompType, MemAvail),`
	`runs(CompType, movie_edit, MemNeeded),`
	`MemAvail >= MemNeeded.`
		Fact 3	`CompType` = `pc`
			`MemAvail` = `64`

4	`runs(pc, movie_edit, MemNeeded),`
	`64 >= MemNeeded.`
		Fact 4	`MemNeeded` = `96`

5	`64 >= 96.`	Failed! Backtracks.	

4	`runs(pc, movie_edit, MemNeeded),`
	`64 >=MemNeeded.`
		Failed! Backtracks.	

3	spec(comp3, CompType, MemAvail), runs(CompType, movie_edit, MemNeeded), MemAvail >= MemNeeded.	Failed! Backtracks.	
2	can_run(comp3, movie_edit).	Failed! Backtracks.	
1	access(X, Comp), can_run(Comp, movie_edit).	Fact 11	X = david Comp = comp1
2	can_run(comp1, movie_edit).	Rule 2	-
3	spec(comp1, CompType, MemAvail), runs(CompType, movie_edit, MemNeeded), MemAvail >= MemNeeded.	Fact 1	CompType = pc MemAvail = 32
4	runs(pc, movie_edit, MemNeeded), 32 >= MemNeeded.	Fact 4	MemNeeded = 96
5	32 >= 96.	Failed! Backtracks.	
4	runs(pc, movie_edit, MemNeeded), 32 >=MemNeeded.	Failed! Backtracks.	
3	spec(comp1, CompType, MemAvail), runs(CompType, movie_edit, MemNeeded), MemAvail >= MemNeeded.	Failed! Backtracks.	
2	can_run(comp1, movie_edit).	Failed! Backtracks.	
1	access(X, Comp), can_run(Comp, movie_edit).	Fact 12	X = david Comp = comp2
2	can_run(comp2, movie_edit).	Rule 2	-
3	spec(comp2, CompType, MemAvail), runs(CompType, movie_edit, MemNeeded), MemAvail >= MemNeeded.	Fact 2	CompType = mac MemAvail = 128
4	runs(mac, movie_edit, MemNeeded), 128 >= MemNeeded.	Failed! Backtracks.	
3	spec(comp2, CompType, MemAvail), runs(CompType, movie_edit, MemNeeded), MemAvail >= MemNeeded.	Failed! Backtracks.	
2	can_run(comp2, movie_edit).	Failed! Backtracks.	
1	access(X, Comp), can_run(Comp, movie_edit).	Failed! Backtracks.	
Original query	can_use(X, movie_edit).	Failed. The top level is reached and no answers have been found. **Output:** **no**	
Overall output	**no**		