Logic Programming

Backtracking and Cut

Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University of Linz, Austria
kutsia@risc.jku.at

Contents

Finitely Many Alternatives

Simplest way: Several facts match against the question.

Example

```
father(mary,george).
father(john,george).
father(sue,harry).
father(george,edward).

?- father(X,Y).
X=mary, Y=george ;
X=john, Y=george ;
X=sue, Y=harry ;
X=george, Y=edward
```

The answers are generated in the order in which the facts are given.

Repeating the Same Answer

Old answers do not influence newer ones: same answer can be returned several times.

Example

```
father(mary,george).
father(john,george).
father(sue,harry).
father(george,edward).

?- father(_,X).
X=george ;
X=george ;
X=harry ;
X=edward
```

`george` returned twice because George is the father of both Mary and John.

Embedding Does Not Matter

Backtracking happens in the same way if the alternatives are embedded more deeply.

Example

```
father(mary,george).
father(john,george).
father(sue,harry).
father(george,edward).
child(X,Y):-father(Y,X).

?- child(X,Y).
X=george, Y=mary ;
X=george, Y=john ;
X=harry, Y=sue ;
X=edward, Y=george
```

Mixing facts and Rules

If facts and rules are mixed, the alternatives follow again in the order in which things are presented.

Example

```
person(adam).                  ?- person(X).
person(X):-mother(X,Y).        X=adam ;
person(eve).                   X=cain ;
mother(cain,eve).              X=abel ;
mother(abel,eve).              X=jabal ;
mother(jabal,adah).            X=tubalcain ;
mother(tubalcain,zillah).      X=eve
```

Multiple Goals with Multiple Solutions

More interesting case: Two goals, each with several solutions.

Example

```
pair(X,Y):-            ?- pair(X,Y).
  boy(X),girl(Y).      X=john, Y=griselda ;
boy(johm).             X=john, Y=ermintrude ;
boy(marmaduke).        X=john, Y=brunhilde ;
boy(bertram).          X=marmaduke, Y=griselda ;
boy(charles).          X=marmaduke, Y=ermintrude ;
girl(griselda).        X=marmaduke, Y=brunhilde ;
girl(ermitrude).       X=bertram, Y=griselda ;
girl(brunhilda).       ...
```

12 solutions.

Infinite Number of Possibilities

Sometimes one might want to generate an infinite number of possibilities.
It might not be known in advance how many of them needed.

Example

```
is_integer(0).
is_integer(X):-is_integer(Y), X is Y + 1.

?- is_integer(X).
X=0 ;
X=1 ;
X=2 ;
...
```

How does it work?

Member and Multiple Solutions

Most rules give rise to alternative solutions if they are used for goals that contain many uninstantiated variables.

Example

```
member(X, [X|_]).
member(X,[_|Y]):-member(X,Y).

?- member(a,X).
X=[a|_G314] ;
X=[_G313, a|_G317] ;
X=[_G313, _G316, a|_G320] ;
...
```

There is a way to tell PROLOG to discard choices: The "cut".

The "Cut"

Cut (written "!") tells the system which previous choices need not to be considered again when it backtracks.
Advantages:

- The program will run faster. No time wasting on attempts to re-satisfy certain goals.
- The program will occupy less memory. Less backtracking points to be remembered.

Example of Cut

Reference library:

- Determine which facilities are available.
- If one has an overdue book can only use the *basic facilities*.
- Otherwise can use the *general facilities*.

Reference Library

Example

```
facility(Pers,Fac):-
    book_overdue(Pers,Book),!,
    basic_facility(Fac).
facility(Pers,Fac):-general_facility(Fac).

basic_facility(reference).
basic_facility(enquiries).

additional_facility(borrowing).
additional_facility(inter_library_loan).

general_facility(X):-basic_facility(X).
general_facility(X):-additional_facility(X).
```

Reference Library

Example

```
book_overdue('C. Watzer', book10089).
book_overdue('A. Jones', book29907).
...
client('C. Watzer').
client('A. Jones').
...

?- client(X), facility(X,Y).
```

How does it proceed?

Reference Library

The effect of cut:

- If a client has an overdue book, then only allow her/him the basic facilities.
- Don't bother going through all the clients overdue books.
- Don't remember any other rule about facilities.

The Effect of Cut

In general, when a cut is encountered as a goal

- The system becomes committed to all choices made since the parent goal was invoked.
- All other alternatives are discarded.
- An attempt to re-satisfy any goal between the parent goal and the cut goal will fail.

Common Uses of Cut

Three main cases:

1. To tell the system that it found the right rule for a particular goal. *Confirming the choice of a rule.*
2. To tell the system to fail a particular goal without trying for alternative solutions. *Cut-fail combination.*
3. To tell the system to terminate the generation of alternative solutions by backtracking. *Terminate a "generate-and-test".*

Confirming the Choice of a Rule

Typical situation:

- We wish to associate several clauses with the same predicate.
- One clause is appropriate if the arguments are of one form, another is appropriate if the arguments have another form.
- Often (but not always) these alternatives can be made disjoint by providing just the argument patterns (e.g., empty list in one clause, and a nonempty list in another.)
- If we cannot specify an exhaustive set of patterns, we may give rules for some specific argument types and gave a "catchall" rule at the end for everything else.

Confirming the Choice of a Rule

Example of the case when an exhaustive set of patterns can not be specified:

Example

```
sum_to(1,1).

sum_to(N,Res):-
  N1 is N - 1,
  sum_to(N1,Res1),
  Res is Res1 + N.

?- sum_to(5,X).
X=15 ;
```

It loops.

Confirming the Choice of a Rule

What happened?

- `sum_to(1,1)` and `sum_to(N,Res)` are not disjoint alternatives.
- `sum_to(1,1)` matches both `sum_to(1,1)` and `sum_to(N,Res)`.
- But if a goal matches `sum_to(1,1)`, there is no reason why it should try the second alternative, `sum_to(N,Res)`.
- Cut the second alternative.

Confirming the Choice of a Rule

Example

```
sum_to(1,1):-!.

sum_to(N,Res):-
  N1 is N - 1,
  sum_to(N1,Res1),
  Res is Res1 + N.

?- sum_to(5,X).
X=15 ;

No
```

More Usual Situation

- In the previous example we could specify a pattern for the boundary case `sum_to(1,1).`
- Usually, it is hard to specify pattern if we want to provide extra conditions that decide on the appropriate rule.
- The previous example still loops on goals `sum_to(N,Res)` where `N =< 1.`
- We can put this condition in the boundary case telling PROLOG to stop for such goals.
- But then the pattern can not be specified.

Cut with Extra Conditions

Example

```
sum_to(N,1):-N =< 1, !.

sum_to(N,Res):-
  N1 is N - 1,
  sum_to(N1,Res1),
  Res is Res1 + N.
```

Cut and Not

General principle:

- When cut is used to confirm the choice of a rule, it can be replaced with `not`.
- `not(X)` succeeds when `X`, seen as a PROLOG goal, fails.
- Replacing cut with `not` is often considered a good programming style.
- However, it can make the program less efficient.
- Trade-off between readability and efficiency.

Cut and Not

Example (With Cut)

```
sum_to(1,1):-!.

sum_to(N,Res):-
  N1 is N - 1,
  sum_to(N1,Res1),
  Res is Res1 + N.
```

Example (With Not)

```
sum_to(1,1).
sum_to(N,Res):-
  not(N=1),  % N\ = 1,
  N1 is N - 1,
  sum_to(N1,Res1),
  Res is Res1 + N.
```

Cut and Not

Example (With Cut)

```
sum_to(N,1):-N =< 1,
  !.
sum_to(N,Res):-
  N1 is N - 1,
  sum_to(N1,Res1),
  Res is Res1 + N.
```

Example (With Not)

```
sum_to(N,1):-N =< 1.
sum_to(N,Res):-
  not(N=<1),  %N > 1,
  N1 is N - 1,
  sum_to(N1,Res1),
  Res is Res1 + N.
```

Double Work

`not` might force PROLOG to try the same goal twice:

Example

```
A:-B,C.
A:-not(B),D.
```

B may be tried twice after backtracking.

The "Cut-fail" Combination

`fail.`

- Built-in predicate.
- No arguments.
- Always fails as a goal and causes backtracking.

The "Cut-fail" Combination

`fail` after `cut`:

- The normal backtracking behavior will be altered by the effect of cut.
- Quite useful combination in practice.

The Average Taxpayer

Write a program to determine an average taxpayer.
Two cases:

- Foreigners are not average taxpayers.
- If a person is not a foreigner, apply the general criterion (whatever it is) to find out whether he or she is an average taxpayer.

The Average Taxpayer

Example

```
average_taxpayer(X) :-
     foreigner(X), !, fail.
average_taxpayer(X) :-
      satisfies_general_criterion(X).
```

What would happen had we omitted the cut?

The Average Taxpayer

Wrong version, without cut:

Example (Wrong)

```
average_taxpayer(X) :-
     foreigner(X), fail.
average_taxpayer(X) :-
      satisfies_general_criterion(X).
```

If there is a foreigner `widslewip` who satisfies the general criterion, the program will incorrectly answer yes on the goal

```
?- average_taxpayer(widslewip).
```

The Average Taxpayer

We can use cut-fail combination to define `satisfies_general_criterion.`

Two cases:

- A person whose spouse earns more than a certain amount (e.g. Euro 3000) does not satisfy the criterion of being an average taxpayer.
- If this is not the case, then a person satisfies the criterion if his income is within a certain interval (e.g. more that Euro 2000 and less than Euro 3000).

The Average Taxpayer

Clauses for `satisfies_general_criterion`.

Example

```
satisfies_general_criterion(X) :-
      spouse(X,Y),
      gross_income(Y,Inc),
      Inc > 3000,
      !, fail.
satisfies_general_criterion(X) :-
      gross_income(X,Inc),
      Inc < 3000,
      Inc > 2000.
```

The Average Taxpayer

We can use cut-fail combination to define `gross_income`.

Two cases:

- A person who gets a pension less than certain amount (e.g. Euro 500), is considered to have no gross income.
- Otherwise, person's gross income is determined as the sum of his/her gross salary and investment income.

The Average Taxpayer

Clauses for `gross_income`.

Example

```
gross_income(X,Y) :-
      receives_pension(X,P),
      P < 500,
      !, fail.
gross_income(X,Y) :-
      gross_salary(X,Z),
      investment_income(X,W),
      Y is Z+W.
```

`not` with Cut and Fail

`not` can be defined in terms of cut and fail.

Example

```
not(P) :- call(P), !, fail.
not(P).
```

Replacing Cut with `not`

- Cut can be replaced with `not` in cut-fail combination.
- Unlike the first use of cut, this replacement does not affect efficiency.
- However, more reorganization of the program is required.

Example

```
average_taxpayer(X) :-
  not(foreigner(X)),
  not(spouse(X,Y),gross_income(Y,Inc),Inc>3000),
  ⋮
```

Terminating a "Generate-and-Test"

"Generate-and-Test":

- One of the simplest AI search techniques.
- Generate: Generate all possible solutions to a problem.
- Test: Test each to see whether they are a solution.
- A possible solution is generated and then tested.
- If the test succeeds a solution is found.
- otherwise, backtrack to next possible solution.

Tic-Tac-Toe

Tic-Tac-Toe game: Get three in a raw, column, or diagonal:

X		O
O	O	
X	X	X

X	X	O
O	X	
O	X	

O		O
X	O	
X	X	O

Representation:

1	2	3
4	5	6
7	8	9

Tic-Tac-Toe

We will show a part of the program to play Tic-Tac-Toe.

Used predicates:

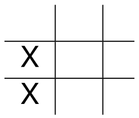
- `var`: built-in predicate. `var(T)` succeeds if `T` is a free variable.
- `arg`: built-in predicate. `arg(N,T,A)` succeeds if `A` is `N`th argument of the term `T`.
- `aline`: defined predicate. Generator of possible lines. For instance, `aline([1,5,9])` is the following line:

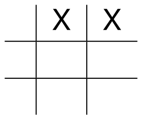
X		
	X	
		X

Part of the Program for Tic-Tac-Toe

The opponent (playing with crosses) is threatening to claim a line:

```
threatening([X,Y,Z],B,X) :-
  empty(X,B), cross(Y,B), cross(Z,B).
```





Part of the Program

Example

```
forced_move(Board,Sq) :-
    aline(Squares),
    threatening(Squares,Board,Sq),
    !.
aline([1,2,3]).
aline([4,5,6]).
aline([7,8,9]).
aline([1,4,7]).
aline([2,5,8]).
aline([3,6,9]).
aline([1,5,9]).
aline([3,5,7]).
```

Part of the Program

Example (Cont.)

```
threatening([X,Y,Z],B,X) :-
  empty(X,B), cross(Y,B), cross(Z,B).
threatening([X,Y,Z],B,X) :-
  empty(Y,B), cross(X,B), cross(Z,B).
threatening([X,Y,Z],B,X) :-
  empty(Z,B), cross(X,B), cross(Y,B).
```

`forced_move`

`forced_move` implements "generate-and-test":

- Moves Generated by `alines`: All possible ways that cross can win.
- Moves Tested by `threatening`: If cross can win in the next move.
- If no forced moves are found, then the predicate fails and some other predicate would decide what move to make.

Cut

Suppose embedded in a larger program:

- If `forced_move` successfully finds a move then `Sq` becomes instantiated to the move.
- If, later, a failure occurs (after this instantiation) `forced_move` would retry.
- Cut can prevent PROLOG to search further (which would be futile) and not waste time.
- When we look for forced moves it is only the first solution that is important.

Problems with the Cut

Cut changes behavior of programs:

- Introducing cuts may give a correct behavior when goals are of one form.
- There is no guarantee that anything sensible will happen if goals of another form start appearing.

Problems with the Cut

Example

```
number_of_parents(adam,0) :- !.
number_of_parents(eve,0) :- !.
number_of_parents(_,2).

?- number_of_parents(eve,X).
X = 0 ;
No
?- number_of_parents(john,X).
X = 2 ;
No
?- number_of_parents(eve,2).
Yes
```

Problems with the Cut

Example

Improved Version

```
number_of_parents(adam,N) :- !, N=0.
number_of_parents(eve,N) :- !, N=0.
number_of_parents(_,2).

?- number_of_parents(eve,2).
No
```

However, it will still not work properly if we give goals such as

```
?- number_of_parents(X,Y).
```