A cut prunes or “cuts out” an unexplored part of a Prolog search tree.

Cuts can make a computation more efficient by eliminating futile search and backtracking.

Cuts are controversial because they are impure.

A cut is written as “!”.

When a rule

\[ B : - C_1, \ldots, C_{j-1}, !, C_{j+1}, \ldots, C_k \]

is applied, the cut tells control to backtrack past \( C_{j-1}, \ldots, C_1 \), and \( B \) without considering any more rules for them.

Example with CUTS

age(leah, 48).
age(natalie, 30).
age(octavia, 34).
age(darrell, 59).
age(michael, 8).
age(sue, 15).
age(sylvia, 81).
age(loren, 29).
age(lura, 87).
age(ron, 60).
blond(leah).
blond(natalie).
blond(octavia).
brunette(darrell).
brunette(michael).
redhair(sylvia).
redhair(loren).
redhair(sue).
grayhair(lura).
grayhair(ron).

cast(X) :- age(X, A), satisfactory(X, A).
satisfactory(X, A) :- between(0, 10, A), !, blond(X).
satisfactory(X, A) :- between(11, 20, A), !, redhair(X).
satisfactory(X, A) :- between(20, 50, A), !, brunette(X).
satisfactory(X, A) :- between(50, 90, A), !, grayhair(X).

This eliminates some needless search.
Cut + Fail achieve Negation

not( X ) :- X, !, fail
not( _ ) .

Fail is a system predicate that fails. _ is a wild-card variable.

The first rule attempts to satisfy X. If X fails, then the second rule succeeds, because _ unifies with any term.

If X succeeds, then the fail predicate forces failure, and the cut prevents consideration of the second rule.

Note that if not( X ) succeeds, it merely means that X is not provable according to the database.

X may or may not be actually false.

Another Cut/Fail Combination Example

allow(elephant) :- !, fail .
allow(Animal) :- size(Animal, lessthan50), license(Animal).
allow(Animal) :- lives(Animal, cage) .

meaning

If an animal is not an elephant and either weights less than 50 pounds and has a license or lives in a cage, it is allowed.

Elephants, even small ones that live in cages, are not allowed.
Gathering Answers into Bags or Sets

The predicates bagof and setof are used to gather instances of objects.

We specify a goal, a variable in the goal, and a bag or set name.

For each success of the goal, the constant that matched this variable is gathered into the bag or set.

Example

```prolog
parent( jan, bet ).
parent( jan, cat ).
parent( joe, ann ).
parent( joe, cat ).

?- bagof( Child, parent( jan, Child ), B ).
   B = [ bet, cat ]

?- bagof( Child, Who^ ( parent ( Who, Child ) ), B ).
   B = [ bet, cat, ann, cat ]

?- setof( Child, P^( parent ( P, Child ) ), S ).
   S = [ ann, bet, cat ]
```

Dynamic Knowledge Assertion/Retraction

Prolog provides built in functions to work with Horn Clauses.

You can

1) Construct a structure representing a clause
2) add a clause to the database
3) remove a clause from the database

* All Prolog structures have the form functor ( arguments )

Facts are already in this form. To convert a rule to this form

```
P(X₁, ..., Xₙ) :- Q₁(X₁, ..., Xₙ), Q₂(···), ···Qₙ(···)
```

converts to

```
' :- '(P(X₁, .., Xₙ)), ' (Q₁(···), Q₂(···), ···Qₙ(···))
```

example: ` :- '( cat(X), ' (animal(X), furry(X)) )`
Some Utilities for Dynamic Knowledge

read/write
read( T ) reads a term T from the input stream.
write( T ) writes a term T to the output stream.

listing
listing( A ) writes out all clauses with atom A as their predicate to the output stream.

functor
functor( T, F, N ) succeeds if T is a structure with functor F and arity N. (If T is a variable, it constructs such a structure.)

arg
arg( Num, T, Argument ) puts Argument into structure T as argument number Num.

assert
assert{ q}( C ) adds clause C to the database at the beginning.

retract
retract( C ) removes the first clause that matches C from the database.

Example
new_fact :- read( A1), read( A2 ), read( A3 ),
functor( C, A3, 2 ),
arg( 1, C, A1 ),
arg( 2, C, A2 ),
assert( C ).

This rule reads 3 terms; uses functor to set up a structure named C with A3 as its predicate, and room for 2 arguments; uses arg to make A1 and A2 the arguments; and asserts it.
|?- new_fact.
|: bob.
|: mike.
|: father.
yes

|?- new_fact.
|: mauro.
|: nick.
|: father.
yes

|?- listing( father ).
   father( bob, mike ).
   father( mauro, nick ).
yes

P.58

Call

A call event occurs when Prolog starts trying to satisfy a goal.
You can also invoke call dynamically, like assert.

Example

check_fact :- read( B1 ), read( B2 ), read( B3 ),
functor( D, B3, 2 ),
arg( 1, D, B1 ),
arg(2, D, B2 ),
call( D ).

|?- check_fact.
|: bob.
|: mike.
|: father.
yes

|?- check_fact.
|: mauro.
|: mike.
|: father.
no

P.59
The Univ Operator = ..

This is the easiest and clearest way to construct dynamic assertions and calls.

-- The predicate \( f(a, b, c) \) corresponds to the list \([ f, a, b, c ]\).

-- The operator =.. converts back and forth between the two representations.

?- \( f(a, b, c) =.. X \).
\( X = [ f, a, b, c ] \)
yes

?- \( X =.. [ w, x, y, z ] \).
\( X = w(x, y, z) \).
yes

Using =.. To Construct Dynamic Calls

mother(linda, sylvia).
father(linda, aaron).

answer_questions :-
    write('mother or father?'),
    read(X),
    write('of whom?'),
    read(Y),
    Q =.. [X, Y, Who],
    call(Q),
    write(Who),
    nl.

1  ?- answer_question.
mother or father? mother.
of whom? linda.
sylvia
Yes

2  ?- answer_question.
mother or father? father.
of whom? linda.
aaron
Yes
Using =.. To Construct Dynamic Asserts

fact :- F =.. [dog, sierra],
       assert(F),
       write(ok),
       nl.

rule :- R =.. [' :- ', animal(X), dog(X)],
        assert(R),
        write(ok),
        nl.

comprule :- C =.. [' , ', dog(X), waggingtail(X)],
            S =.. [' :- ', friendly(X), C],
            assert(S),
            write(ok),
            nl.

?- fact. ok yes
?- rule. ok yes
?- comprule. ok yes
?- consult(user).
|: waggingtail(sierra).

2 ?- dog(Who).
Who = sierra

8 ?- friendly(Who).
Who = sierra

Clause provides another way of selecting Horn clauses.

Clause(X, Y) succeeds if it can match X and Y to the head and body of an existing clause in the database.
X must be instantiated enough so that the main predicate is known. Only works for dynamically asserted clauses!!

Example

list1(X) :- clause(X, Y),
           output_clause(X, Y),
           write('·'), nl, fail.

list1(X).
output_clause(X, true) :- !, write(X).
output_clause(X, Y) :- write(X :- Y).

Note that for facts, the tail is true.
Ex. assert(q(a, b)).

list1(q(V1, V2)).
Parsing Simple English Sentences

article(a). article(the). adjective(giant).
preposition(on). preposition(from).
verb(rose). verb(sat). verb(was).

sentence( X ) :- np( X, R ), vp( R, [ ] ).
np( [X,Y[Z], Z ) :- article( X ), noun( Y ).
vp( [X[Y], R ) :- verb( X ), pp( Y, R ).
pp( [X[Y], Z ) :- preposition( X ), np( Y, Z ).

|?- sentence( [the, cat, sat, on, the, mat] ).
|?- sentence( [the, rocket, was, on, the, pad] ).
|?- sentence( [the, mat, was, on, the, cat] ).
|?- sentence( [the, rocket, rose] ).
|?- sentence( [the, giant, cat, rose, from, the, mat] ).