Prolog Cut and Negation

Lesson Overview (Agenda)

In the following lesson, we will explore:

1. Prolog cut operator
2. Prolog Negation

Cut operator

- The cut operator in Prolog is the exclamation point: !
- Cut is non-logical, so in this course we avoid using it, but…
- We do need to know what it means
- When cut (!) appears as a goal in the body of a predicate
    - it is always true,
    - it discards choice points (see scope of cut slide below)
- Intuitively cut means "if the proof process gets to a cut in a predicate body, then commit to all choices made so far while working on that predicate"

Example of cut (we would not use it). Student Grades

Numeric Grade	Letter Grade
90	A+
85	A
80	A-
77	B+
73	B
70	B-
67	C+
63	C
60	C-
57	D+
53	D
50	D-
0	F

Prolog: Convert Number Grade to Letter

Code	Numeric Grade	Letter Grade
convert(X,'A+'):-X>=90,!.	90	A+
convert(X,'A'):-X>=85,!.	85	A
convert(X,'A-'):-X>=80,!.	80	A-
convert(X,'B+'):-X>=77,!.	77	B+
convert(X,'B'):-X>=73,!.	73	B
convert(X,'B-'):-X>=70,!.	70	B-
convert(X,'C+'):-X>=67,!.	67	C+
convert(X,'C'):-X>=63,!.	63	C
convert(X,'C-'):-X>=60,!.	60	C-
convert(X,'D+'):-X>=57,!.	57	D+
convert(X,'D'):-X>=53,!.	53	D
convert(X,'D-'):-X>=50,!.	50	D-
convert(X,'F').	0	F

Seems to work? (no, not good)

```
?- convert(85,Grade).
Grade = 'A'.


?- convert(85,'C-').
true.
```

- Cut can result in wrong answers because it is non-logical
- We stay as purely logical as we can, so we avoid cut
- Bad code can be inefficient compared to good code, so we need to make sure we write good code, but we don't use cut to increase efficiency in this course

THE SCOPE OF CUT

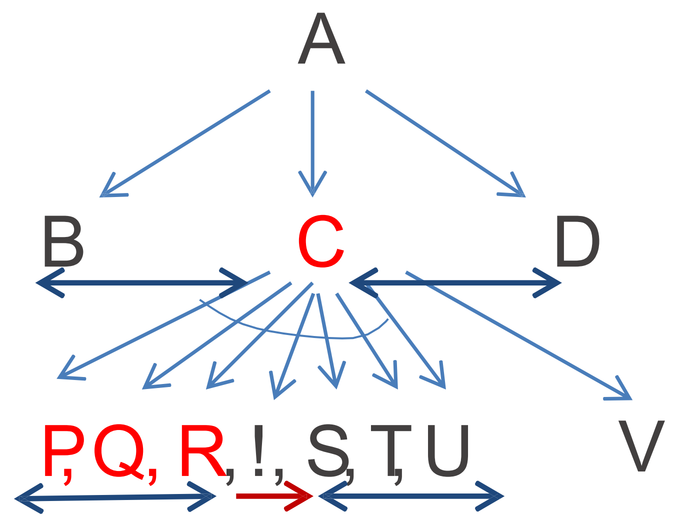
C :- P, Q, R, !, S, T, U.

C :- V.

A :- B, C, D.

?- A.

- This cut discards choice points in R, Q, P, C
- The two rules for C are a choice point when trying to prove C, for example.
- The cut does not discard choice points in B or A because those choice points are out of the scope of the cut: scope shown in RED



The cut is not "visible" from A (cut is nested too deep from point of view of A)

Negation

- In Prolog, negation is defined as:

```
not( P) :-
    P, !, fail      % if P is true, then commit to fail
    ;               % this line makes a choice point that would be discarded by the cut
    true.           % P must be false because the cut wasn't reached
                    % so not(P) is true
```

- This is called *negation as failure*
- **not** can be written as a prefix operator: **\+ P**

Negation Example

```
likes(john, X) :-
    music(X),
    \+ heavy_metal(X).
```

- John likes all music except heavy_metal
- This is more readable than the formulation with cut + fail

Negation as Failure

- Not exactly the same as negation in logic (mathematics)

- Negation as failure makes the “closed world assumption”
- Standard abbreviation: CWA = Closed World Assumption
- The CWA is: Everything that Prolog cannot derive from the program is assumed to be false
- SWI Prolog notation for **not P** is:

```
\+ P
```

Closed World Assumption

- What does yes/no mean under CWA? Consider this single line program:

  round(sun).

- How should Prolog's answers be understood in the following?

```
?-  round(sun).
    true              % true, round(sun) logically follows from program
?-  round(earth).
    false             % false means: I don't know, can't be derived from program
?-  \+ round(earth).
    true              % It follows from the program, but only under CWA
```

Problems with Negation

- Negation as failure is defined through non-logical cut, so we can expect some difficulties. Consider this example:

```
% person(X) means that X is a person
person(jack).
person(judy).
person(jeff).
% male(X) means that X is male
male(jack).
male(jeff).
% female(X) means that X is female
female(X):-
   \+ male(X).
```

Unexpected results due to negation

```
?- male(jack).
true.
?- female(judy).
true.
?- male(X).
X = jack ;
X = jeff.
?- female(X).
false.          % nobody is female
?- female(judy).
true.           % judy is female but nobody is female?
?-
```

Negation is non-logical

- Negation gives incorrect answers when the negated term involves unbound variables
- A term with no unbound variables is called a "gound term"
- Order matters with negation: delay negation as much as possible to increase chances all variables will be bound

```
?- \+ X = a.
false.

?- \+ X = a, X = b.
false.

?- X = b, \+ X =a.
X = b.

?-
```

When does order matter due to unbound variables?

- Each of these is a problem if we change the order:

1. Recursion and infinite loops

```
ancestor(X,Z):-
   parent(X,Y),
   ancestor(Y,Z).       % recursion after Y is bound by parent(X,Y)
```

1. Arithmetic

```
X = 4, Y is X * 3.      % arithmetic after X is bound by X = 4
```

1. Negation

```
X = b, \+ X = a.        % negation after X is bound by X = b
```

Conclusion

In this lesson, you learned about Prolog negation, and the cut operator.

In the next lesson, you will learn to use knowledge representation when developing Prolog programs.