# Programming in Logic: Prolog

Cuts & Negation

Readings: 5.3 & 5.4

# *min/3*

- **/* min(+X,+Y,?Z)**
  **where Z is X if X =< Y else Z is Y */**

  *min(X,Y,X) :- X =< Y.*

  *min(X,Y,Y) :- Y < X.*

- Note: the two clauses are mutually exclusive.

# Guards & Guarded Clauses

- *min(X,Y,X) :- X =< Y.*
  *min(X,Y,Y) :- Y < X.*
- *X =< Y* and *Y < X* are mutually exclusive and decide which clause succeeds.
- Goals which determine which clause are to be committed to, we will call *guards*.
- A clause containing guards we will call a *guarded clause*.

# Guarded Clauses & Green Cuts

- *min(X,Y,X) :- X =< Y, !.*
  *min(X,Y,Y) :- Y < X.*

- Green cuts are put in after the guard goals.

- The cut above is a green cut.

# Cuts - Red for Danger

- Consider the following code for *min/3*:

  *min(X,Y,X) :- X=<Y, !.*

  *min(X,Y,Y).*

- Since they're mutually exclusive, we can remove the guard from the 2nd clause, right?
- Is this still a green cut? Why?
- It's red because the set of answers change if you remove the cut. Give example!

# Red Cuts

- Just because a cut is red doesn't mean it is wrong.
- For some relations, need red cuts.
- The problem with red cuts is that you can no longer just rely on understanding "declarative" meaning of program, you have to understand the procedural meaning as well!!!
- This is much trickier!

# Red for Danger

- **/* min(+X,+Y,?Z)**
  **where Z isX if X =< Y else Z is Y */**

  *min(X,Y,X) :- X=<Y, !.*

  *min(X,Y,Y).*
- Does this code capture its dfn of *min/3*?
- How might it go wrong?
- Consider the query *min(2,5,5)*. What does Prolog reply?

# Eureka!

- Need to break implicit binding sharing between 1st & 3rd args, make explicit after test satisfied.
- *min(X,Y,Z) :- X=< Y, !, X = Z.*
  *min(X,Y,Y).*
- This one works.
- Red cuts should avoided unless there's some overriding reason why they should be used.

# Good Use of a Red Cut

- **/* ifThenElse(+P,+Q,+R)**
  **if P then Q else R */**

*ifThenElse(P,Q,R) :- P, Q.*

*ifThenElse(P,Q,R) :- not(P), R.*

- Because of defn of *not/1* these clauses are mutually exclusive. P and *not(P)* are the guards for these clauses. Put green cuts in.

# Conditions & Definitions

- *ifThenElse(P,Q,R) :- P, !, Q.*
  *ifThenElse(P,Q,R) :- not(P), R.*
- If *P* does not have uninstantiated variables (as modes indicate) this defn works. **However,** if *P* has uninstantiated variables & want *P* to succeed as many ways as possible, then this defn fails to capture that & no green cuts are possible.

# Red Cut to the Good

- *ifThenElse(P,Q,R) :- P, !, Q.*
  *ifThenElse(P,Q,R) :- not(P), R.*
- Computing *P* can be arbitrarily expensive, doing it twice, doubly so. Only need to do it once.
- *ifThenElse(P,Q,R) :- P, !, Q.*
  *ifThenElse(P,Q,R) :- R.*
- This red cut avoids recomputing *P*.

# Determinate - One Success

- *ifThenElse/3* shows that simply mutually exclusive clauses is not enough.
- There must be only one way for the guard goals to succeed.
- Otherwise, cuts placed behind guard goals might not be green because they change the meaning (i.e., answers) of the code.

# Negation

- Sometimes, want to know whether something is false.
- One approach is to have facts for falsity, e.g., *not(on(a,b))*. is a fact stating that *a* is *not on b*.
- Another is to have a rule that states when something is false.

# Defining Negation

- Negation as failure:

  *not(P) :- P, !, fail.*

  *not(_).*

- If *P* succeeds then *not(P)* fails, and vice versa.
- This is not true negation, why?
- Can true negation be implemented? Why or why not?

# Default Rules

- Often the non-final clauses are special cases, and the last clause is the default.
- *not/1* and *ifThenElse/3* are examples of this.