Logic Programming

Lecture 4:
Nonlogical features:
arithmetic, I/O, cut

Quick note
• Several predicates discussed so far (or today) are built-in (sometimes with different names):
  • append/3
  • mem (member/2)
  • len (length/2)
• It is good to know how to define them from scratch, if necessary
• LPN text "predicate index" lists all built-in predicates you are expected to know (and more!)

Nonlogical features
• So far we've worked (mostly) in pure Prolog
  • Solid logical basis
  • Elegant solutions to symbolic problems
• But, many practical things become inconvenient
  • Arithmetic
  • I/O
• And standard proof search not always efficient
  • Can we control proof search better?

Outline for today
• Nonlogical features
  • Expression evaluation
  • I/O
  • "Cut" (pruning proof search)
  • Negation-as-failure
  • more in 2 weeks
Expression evaluation

- Prolog has built-in syntax for arithmetic expressions
- But it is uninterpreted

?- 2 + 2 = 4.
no.

?- X = 2+2.
X = 2+2
Expression evaluation

- Prolog has built-in syntax for arithmetic expressions
- But it is uninterpreted

?- 2 + 2 = 4.
no.
?- X = 2+2.
X = 2+2
?- display(2+2).
+(2,2)

The evaluation predicate "is"

- Prolog provides a built-in predicate "is"

?- X is 2+2.
X = 4
?- X is 6*7.
X = 42

Machine arithmetic with "is"

- addition (+), subtraction (-)

?- X is 2+(3-1).
X=4

- multiplication (*), division (/), "mod"

?- X is 42 mod 5, Y is 42 / 5.
X = 2,
Y = 8.4
**Warning**

- Unlike "=","is" is **asymmetric**
- **only** has mode (?,+)
  
  `- 2+(3-1) is X.  
  ! Instantiation error...
- requires RHS to be a **ground expression**
  
  `- X is foo(bar).
  ! Domain error...

**Lists and arithmetic**

- Length of a list
  
  `len([],0).`
  
  `len([_|L], N) :- 
      len(L,M), N is M+1.`
- **Only works in mode (+,?).**
- **Built-in length/2**
  
  • (works in both directions)

**Building a list of length n**

- Similar to list length...
  
  `build([],0).`
  `build([_|L], N) :- 
      M is N-1, build(L,M).`
- **Only works in mode (?,?,+).**
  
  • (But see built-in length(?L,?N))

**Lists and arithmetic**

- Summing elements of a list
  
  `sumall([],0).`
  
  `sumall([X|L],S) :- 
      sumall(L,M), S is M+X.`
- **What mode can this have?**
Arithmetic comparisons

- Binary relations also built-in as goals:
  - less than (<), greater than (>)
  - less/equal (=<), greater/equal (>=)
  - equality (=:=), inequality (=/=)
- All have mode (+,+)
  - both arguments must be ground!

Example

- "Maximum" predicate
  
  max(X,Y,Y) :- X =< Y.
  max(X,Y,X) :- X > Y.
- Works in mode (+X,+Y,?M).

Basic Input/Output

- read(?X) reads in a term (followed by ".")
- write(+X) prints out its argument as a term.
- nl/0 prints a newline
- Simple expression calculator:
  
  calc :- read(X),
        Y is X,
        write(X = Y), nl,
        calc.

Backtracking through I/O

- Short answer: can backtrack, but **can't undo I/O**
  
  ?- write(foo),fail; write(bar).
  fooobar
- Any **bindings** will be **undone**
  
  ?- read(X), fail; X = 1.
       |: foo.
       X = 1
Cut

• Sometimes we **know** we've made the right choice
  • No backtracking needed
• In Pure Prolog we can't take advantage of this
• Introducing "cut" (!), the **proof-search pruning** operator

---

Example

• The "member of a list" predicate
  
  member(X,[X|L]).

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

  • If X is ever found in L, it is pointless to backtrack and keep looking for solutions

---

Example

• The "member of a list" predicate
  
  member(X,[X|L]) :- !.

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

  • If X is ever found in L, it is pointless to backtrack and keep looking for solutions
  • Insert a cut **in first rule** to cut off search

---

How it works

• Remember choice points (places where we could have tried a different rule or branch).
  • When we encounter a cut, "prune" all pending alternatives **since cut was introduced**
How it works

**Without cut**

R1: member(X,[X|_]).
R2: member(X,[_|L]) :- member(X,L)

------

James Cheney  Logic Programming  October 2, 2014

---

R1 (X=1)
R2 (X = 1, L = [2,1])

---

James Cheney  Logic Programming  October 2, 2014

---

R1: member(X,[X|_]).
R2: member(X,[_|L]) :- member(X,L)

---

James Cheney  Logic Programming  October 2, 2014

---

R1: member(X,[X|_]).
R2: member(X,[_|L]) :- member(X,L)

---

James Cheney  Logic Programming  October 2, 2014

---
How it works

**Without cut**

R1: member(X,[X|_]).
R2: member(X,[_|L]) :- member(X,L)

---

**With cut**

R1: member(X,[X|_]) :- !.
R2: member(X,[_|L]) :- member(X,L)

---

James Cheney Logic Programming October 2, 2014
How it works

With cut

R1: member(X,[X|_]) :- !.
R2: member(X,[_|L]) :- member(X,L)

Another example

p(X,Y) :- q(X), r(X,Y).
p(X,X) :- a(X).
q(X) :- a(X).
r(X,Y) :- b(X), c(Y).
a(1), a(3).
b(1), b(2).
c(2), c(3).
Another example

\[
p(X,Y) :- q(X), \neg, r(X,Y).
\]
\[
p(X,X) :- a(X).
\]
\[
q(X) :- a(X).
\]
\[
r(X,Y) :- b(X), c(Y).
\]
\[\text{a(1). a(3).}
\]
\[
\text{b(1). b(2).}
\]
\[
\text{c(2). c(3).}
\]
Max

\[ \text{max}(X,Y,Y) :- X \leq Y. \]
\[ \text{max}(X,Y,X) :- X > Y. \]

- Pointless to try to backtrack
  - if the first goal succeeds, then the second won't!

Max using cut

\[ \text{max}(X,Y,Y) :- X \leq Y, !. \]
\[ \text{max}(X,Y,X) :- X > Y. \]

- Pointless to try to backtrack
  - if the first goal succeeds, then the second won't!

But what about...

- Isn't it silly to test \( X < Y \) in second rule?
  \[ \text{max}(X,Y,Y) :- X \leq Y, !. \]
  \[ \text{max}(X,Y,X). \]
- Maybe (slightly) more efficient to skip it
  - But damages transparency
  - \( \text{max}(1,2,1) \) and \( \text{max}(1,2,2) \) both succeed!
  - Rule order matters!

Safe use of cut

- Cut can make program more efficient
  - by avoiding pointless backtracking
- But as shown with "max", cuts can change meaning of program (not just efficiency)
  - "Green" cut - preserves meaning of program
  - "Red" cut - doesn't.
Next time

- More about cut & negation
- Further reading:
  - LPN, ch. 5 & 10