Logic Programming

Lecture 9: Constraint logic programming

Outline for today

• Infix operators/declarations
• Logic programming with constraints
  • Finite domain constraints
  • Real/rational constraints
• Course review outline

Infix operators

• Syntax of Prolog has many **built-in** infix operators
  
  + - * / = is =..  

• You can also define your own prefix, infix, or postfix operators

• Syntax and meaning are defined independently

Defining your own operators

• :- op(Prec, Fixity, Op).

• Prec is precedence - higher is weaker binding

• Fixity is
  
  • xfx, xfy, yfx - infix (non, right, left assoc)
  • fx, fy - prefix
  • xf, yf - postfix

• x, y indicate associativity (x needs explicit parentheses)

• Op can be an atom or list of atoms
Looking under the hood (bonnet?)

- Standard Prolog ops declared as:
  
  ```prolog
  :- op(1200, xfx, [ :-, --> ]).
  :- op(1100, xfy, [ ; ]).
  :- op(1000, xfy, [ ',' ]).
  :- op( 700, xfx, [ =, is, ...]).
  :- op( 500, yfx, [ +, - ]).
  :- op( 500,  fx, [ +, - ]).
  ...
  ```

Remember

- Prolog supports arithmetic, but it's not very "logical"

  ```prolog
  ?- 2+2 = 4.
  no
  ?- X is 1+2.
  X = 3
  ?- 1+2 is X.
  Instantiation error...
  ```

More problems with this

- Using `is/2` for arithmetic, sometimes we have to commit to ground values too early
- Leads to higher branching factor
- Also imposes order of evaluation on programs that use arithmetic
- making programs less readable or reusable

Example

```prolog
between(Low,_,Low).
between(Low,High,N) :-
    Low < High,
    Next is Low + 1,
    between(Next, High, N).
?- between(1,1000,N), N > 999.
?- N > 999, between(1,1000,N).
```
**Constraint Programming**

- Why can't we just say things like
  \[- X + 1 = 5 * Y, \ Y = 1.\]
- and have the system "solve for $X$"?
  \[X = 4\]
- **Constraint Programming** is a well-studied framework that lets us do this
  - (Example: Linear Programming)

**Constraint Logic Programming**

- Constraint Programming is powerful and declarative
- But it can be a pain to use
  - Have to put problem in a specific syntactic form
  - Wouldn't it be nicer to specify constraint problems using Prolog?
  - That's **Constraint Logic Programming**

**Basic idea**

- Expand the program "state" to include special predicates called **constraints**
  - Program can generate constraints at any time
  - Note: Equations $t = u$ are a form of constraint.
- **Reduce** new constraint goals to normal form
  - e.g. unification for $=$
- **Backtrack** if collection of all constraints becomes inconsistent
- **Enumerate** solutions on request

**Finite domain constraints**

- $N$ in $i..j$
  - says that $N$ has one of finitely many values $i..j$
- $t \neq u$
  - equality constraint
- $t \ngeq u$, $t \nkg u$, etc.
  - inequality constraint
- These predicates **constrain** but don't **generate or require values**
between revisited

?- \( N \in 1..100, N \#> 99. \)
\( N \) in \( 99..100 \)
?- \( N \in 1..100, N \#> 99, \)
   \( \text{indomain}(N) \).
\( N = 100. \)

indomain/1

Generates solutions to constraints

?- \( X \in 1..5, Y \#= 2\times X+1, \text{indomain}(Y) \).
\( X = 1, Y = 3 \) ? ;
\( X = 2, Y = 5 \) ? ;
\( X = 3, Y = 7 \) ? ;
\( X = 4, Y = 9 \) ? ;
\( X = 5, Y = 11 \) ? ;

labeling/2

- First argument a list of options ([ ] for now)
- Second argument a list of constrained variables
- Enumerates all solutions, using options to control search.

?- \( X \in 0..3, Y \in 0..3, \)
   \( X \#< Y, \text{labeling}([], [X,Y]) \).

minimize/2, maximize/2

- Given a goal \( G \), find min or max value of constrained var \( Y \) after running \( G \)

?- \( X \in 1..100, \)
   \( Y \#= (X - 50)*X, \)
   \( \text{minimize}(\text{indomain}(Y), Y) \).
\( X = 25, Y = -625 \)
Distinctness

- We also have **inequality** constraints:
  - \( X \neq Y \)
    - says \( X \) and \( Y \) have to be different (both may be nonground)
- and **distinctness** constraints:
  - all_different([\( X_1, \ldots, X_n \)])
  - forces all elements of list to be different

A cryptarithmetic puzzle

SEND
+ MORE
------
MONEY

Goal:
Find distinct numbers S,E,N,D,M,O,R,Y between 0 and 9 such that the numbers formed by SEND and MORE add up to MONEY

Traditional solution

```prolog
solve_money([S,E,N,D], [M,O,R,E], [M,O,N,E,Y]) :-
  between(0,9,S), ..., between(0,9,Y),
  distinct([S,E,N,D,M,O,R,Y]),
  add_carry([0,S,E,N,D], [0,M,O,R,E], [M,O,N,E,Y], 0).
```

Traditional solution

```prolog
add_carry([],[],[],0).
add_carry([A|As],[B|Bs],[C|Cs],Carry) :-
  add_carry(As,Bs,Cs,NextCarry),
  C is (A + B + NextCarry) mod 10,
  Carry is (A + B + NextCarry) / 10.

distinct([]).
distinct([X|Xs]) :- \+(member(X,Xs)),
                   distinct(Xs).
```
CLP(FD) solution

solve_money2( [S, E, N, D],
       [M, O, R, E],
       [M, O, N, E, Y]) :-
  S in 0..9, ..., Y in 0..9,
  all_different([S, E, N, D, M, O, R, Y]),
  add_carry2([0, S, E, N, D],
             [0, M, O, R, E],
             [M, O, N, E, Y], 0),
  labeling([], [S, E, N, D, M, O, R, Y]).

add_carry2([], [], [], 0).
add_carry2([A|As], [B|Bs], [C|Cs], Carry) :-
  add_carry2(As, Bs, Cs, NextCarry),
  C #= (A + B + NextCarry) mod 10,
  Carry #= (A + B + NextCarry) / 10.

Note: Almost the same except for use of constraints.

Other constraint domains

• Real numbers: CLP(R)

?- { 2*X+Y =< 16, X+2*Y =< 11,
     X+3*Y =< 15, Z = 30*X+50*Y },
    maximize(Z).
  X = 7.0, Y = 2.0, Z = 310.0

• Rational numbers: CLP(Q)

Using CLP

• Provided as SICSTUS libraries
  • [library(clpfd)].
  • [library(clpr)].
  • [library(clpq)].
Note: Weird SICSTUS-ism

?- X is 3/2.  % exact division
X = 1.5
?- X is 3//2.  % integer division
X = 1
?- X #= 3/2.  % FD-constraint integer division
X = 1
?- X #= 3//2.   % error!
Domain error....

Review

• Material covered in LPN, ch. 1-6:
  • Terms, variables, unification (+/- occurs check)
  • Arithmetic expressions/evaluation
  • Recursion, avoiding nontermination
  • Programming with lists and terms
  • Expect ability to solve problems similar to those in tutorial programming exercises (or textbook exercises)

Review

• Material covered in LPN, ch. 7-11:
  • Definite clause grammars
  • Difference lists
  • Nonlogical features ("is", cut, negation, assert/retract)
  • Collecting solutions (findall, bagof, setof)
  • Term manipulation (var, =.., functor, arg, call)
  • Expect ability to explain concepts & use in simple Prolog programs

Review

• Advanced topics (Bratko ch. 11-12, 14, 23)
  • Search techniques (DFS, BFS)
  • Symbolic programming & meta-programming
  • Constraint logic programming
  • Expect understanding of basic ideas
  • not ability to write large programs from scratch under time pressure
Some exam info

- Programming exam: 2 hours
- DICE machine with SICSTUS Prolog available
- (Documentation won't be, but exam will not rely on memorizing obscure details)
- Sample exams on course web page
- Exams from >1 year ago are on ITO web page; questions similar but different format.

Learning more

- There is a lot more to logic programming
- Online: comp.lang.prolog
- Association for Logic Programming
- Main journal: Theory and Practice of Logic Programming (CUP) - main journal before 2001 was Journal of Logic Programming
- Main conferences:
  - International Conference on Logic Programming (ICLP) - main annual conference.
  - Principles and Practice of Declarative Programming (PPDP) - covers LP and other “declarative” paradigms
- Honors/MSc projects? Let me know