Programming in Logic: Prolog

Operators & Arithmetic
Readings: 3.3 & 3.4
Quick Quiz

• In the following code, on which lines do which generators appear?

/* p(?,?,?) */

1. p(X,Y,Z) :-
   2. member(X, [1,2,3]),
   3. member(Y, [1,2,3]),
   4. member(Z, [1,2,3]).

• Where could we put not(X=1), not(Y=2), and not(Z=3)?
Structure Presentation

• Presentation has to do with how a structure appears to the user, representation with how it is stored internally.

• For most structure types
  presentation = representation,
  – i.e., functor(arg, …),
  – e.g., in the movie puzzle, movie info item:
    \( m(\text{doggone}, \text{dougDrew}, \text{comedy}) \)
Structure Presentation cont’d

• For some structure types, \( \text{presentation} \neq \text{representation} \), e.g., lists are presented as \([elt, \ldots]\), even though internally they are represented as \((elt, (\ldots, [\ ]) \ldots)\).

• This presentation of lists is built into Prolog.
Operators

• For structure types of arity 1 or 2, Prolog has a facility to specify that the functor be presented as a unary/binary operator.

• A Prolog operator need not be a relation, it can be only a data structure.
Example: Boolean Expressions

• To represent boolean expressions, use binary and and or structures and unary not structures.
• For example: the structure \( \text{and}(\text{or}(A,B),\text{not}(C)) \) represents the boolean expression \( A \lor B \land \neg C \).
• We could tell Prolog to treat them syntactically as binary/unary operators:
  \( A \ or \ B \ and \ not \ C \)
Example cont’d

• If we did this, they would still be represented internally as functors with arguments, but they would present as operators.

• We can specify “operators” at load time:

```prolog
:- op(500, yfx, or).
:- op(400, yfx, and).
:- op(200, fy, not).
```
Operator Specification

• Op spec: $op(+Precedence, +Type, +Functor)$
  – Precedence describes “binding” force of operator.
  – Type describes:
    • Arity: unary or binary.
    • Location of arguments:
      – If unary then whether it is prefix or postfix.
      – If binary then it is infix.
    • Associativity: right, left, or not at all.
  – Functor is the atom that “names” the structure.
Precedence Numbers

• Given $4 + 5 \times 6$, like to know if this represents $(4 + 5) \times 6$ or $4 + (5 \times 6)$, i.e., whether * or + binds their arguments more tightly.

• The operator that has highest precedence number is principal functor of internal representation. + has precedence 500 & * 400, thus $4 + 5 \times 6$ is $+(4, *(5,6))$, i.e., $4 + (5 \times 6)$
Associativity

• Given an expression where adjacent operators have same precedence, e.g., 3 - 4 - 5, is that 3 - (4 - 5) {i.e.,2} or (3 - 4) - 5 {i.e.,-6}?  
• Normally, we would assume the latter, i.e., “-” associates to the left first.  
• So the left side is the associative side for “-”.
Operator Type Specification

• Possible types are: \( xfx, xfy, yfx, fx, fy, xf, yf \).
• \( f \) represents where the functor goes.

• \( y \) represents the associative side of the operator.
• Can’t have two associative sides, why?
• \( x \) represents the non-associative side of the op.
Type Examples

• $xfx$ specifies a binary infix operator that is non-associative, e.g., “$<$” (in most programming languages $3 < 4 < 5$ would be illegal, why?)

• $yf$ specifies an associative unary prefix operator, e.g., “-” (--3 would be the same value in most programming languages as 3).
Quick Quiz

Given the following load-time directives:

```prolog
:- op(700, yfx, garp).
:- op(350, yfx, gulp).
:- op(175, yf, goop).
```

Would the following Prolog expression be legal?

```
goop goop pip gulp pup garp pap
```

If so, what would its internal structure look like?
Arithmetic

• There are some predefined operators to describe arithmetic expressions: +,-,*,/,**,//, and mod.
• These operators do not represent relations (i.e., by itself, \(1 + 2\) is simply the data structure \(+\(1,2\)\).
• Given the top-level goal \(5 = 4 + 1\), Prolog would answer no, why?
Arithmetic Expressions

- $5 = 4 + 1$ is structure $=(5, +(4, 1))$ where "=" is the match relation, and $5$ does not match $+(4,1)$.

- There is a special operator that is a relation that evaluates arithmetic expressions: $\text{is}(\text{Result}, +(\text{ArithmeticExpr}))$

- $\text{ArithmeticExpr}$ must be fully instantiated!

- Like $\text{not}/1$, $\text{is}/2$ is not part of pure Prolog.
is(?Result, +ArithmeticExpr)

• *is/2* evaluates the *ArithmeticExpr* data structure and matches the result against *Result*.

• \(X \text{ is } 4 + 5 \times 6\), assuming \(X\) is initially unbound, would return \(X = 34\) ?.
is(?Result, +ArithmeticExpr)

- 34 \textit{is} \ 4 + 5 \times 6 \ \textit{would return yes}.

- 36 \textit{is} \ 4 + 5 \times 6 \ \textit{would return no}.
is(\(?\text{Result, +ArithmeticExpr}\) )

- 34 is $4 + 5 \times X$, assuming $X$ is initially unbound, would generate a run-time error.

- 34 is $4 + 5 \times X$, assuming $X$ is bound to 6, would return yes (or $X = 6$ ?)
Doing Arithmetic

• Assume we want to define the \( \text{length}(\textit{List}, \textit{Length}) \) relation where \( \text{Length} \) is the number of top-level elements in \( \textit{List} \).

• What will the base case be?

• What will the inductive case be?
Doing Arithmetic cont’d

• /* length(?List,?Length) */
  
  length([], 0).
  length([_|Tail], N) :-
  length(Tail, N1),
  N is N1 + 1.

• **Note**: cannot switch order of goals in last clause, otherwise \( N1 \) would not be instantiated!
Arithmetic Comparisons

• In addition to is/2, arithmetic comparison ops cause arithmetic expressions to be evaluated.

• These are: >, <, >=, =<, =:=, =/=.

• They are non-associative infix operators: $xfx$

• They cause the arithmetic expressions on both sides to be evaluated.

• Their argument modes are (+Left, +Right)
Example: $gcd(+X,+Y,?D)$

- Greatest Common Divisor:
  
  $gcd(X,X,X)$.

$gcd(X,Y,D)$ :-
  
  $X < Y,$
  
  $Y1 \text{ is } Y - X,$
  
  $gcd(X,Y1,D).$

$gcd(X,Y,D) :-$

  $Y < X,$

  $gcd(Y,X,D).$
Example: Complex Numbers

• How do we use operators to make certain types of expressions easier to use?
• We’ll use operators to make using complex numbers easier.
• We’ll just be implementing complex adds, and real multiplication of a complex number.
What do we want to do?

• Make it easy to write complex numbers.

• Make it easy to express complex addition.

• Make it easy to express real multiplication of complex numbers.
Writing Complex Numbers

• A complex number has two parts: a real part and an imaginary part.

• So, we’d like to be able to write a complex number as \textit{realPart} \, ? \, \textit{imaginaryPart}.
  Where “?” represents some special symbol that indicates that we’re dealing with a complex number.
Writing Complex Numbers cont’d

• Since we want to be able to indicate normal arithmetic operations, we can’t use any symbol that already means something else in arithmetic.

• We’ll try “&”.

• $X&Y$ is our representation of a complex number, $X$ and $Y$ must be reals. If either represents a calculation, no subparts can be imaginary.
Examples

• $5 + 6i$ will be $5 & 6$

• $(6 + 6 \times 7 / 8) & (1 - 2 \times 5)$ is a valid complex number $(6 + 4i)$.

• $(0 & 1 \times 0 & 1) & 3$ is not valid because $(0 & 1 \times 0 & 1)$ has imaginary parts, even though the product is real.
Complex Constructor Operator

• `:- op(100, xfx, &).`

• The 100 precedence number means it will be the innermost functor.

• The type says it’s a non-associative binary operator.
Complex Addition

• We could use “+” to represent complex addition, but, we will use “&+” instead.
• Both operands to &+ must be complex.
• 1&2 &+ 5&2 is legal and represents
  \[ 1 + 2i + 5 + 2i \]
• 1 &+ 5&2 is illegal since 1 is not presented as a complex number.
Complex Addition

• Introduce the complex addition operator:
  – :- op(500, yfx, &+).
Multiply Real times Complex

• We allow:
  – 4 * (4&6)
  – (4&6) * 4

• We do not allow:
  – (4&6)*(1&2)
Multiply Real times Complex cont’d

• Since we’re using the same symbol for this type of multiplication as for real multiplication, the operator symbol has already been declared.

• Why can we do this?

• Could we have done this for complex addition?
:- op(100, xfx, &). % complex number constructor

:- op(500, yfx, &+). % complex add

:- op(700, xfx, cis). % complex arithmetic evaluation

/* real(?Complex, ?RealPart) */
real(R&_, R).

/* imag(?Complex, ?ImaginaryPart) */
imag(_&I, I).
/* cis(?Result, +ComplexExpression)  
--------------------------------  
ComplexExpression is evaluated and the result of  
that evaluation is matched against Result.  
*/

W&X cis Y&Z :- X is Z, W is Y.  % Simplifying complex number  
W&X cis Y&+Z :-  % Complex add  
    Y1 cis Y, Z1 cis Z),  
    real(Y1, RY1), real(Z1, RZ1), W is RY1 + RZ1,  
    imag(Y1, IY1), imag(Z1, IZ1), X is IY1 + IZ1.

V&W cis X * Y&Z :-  % multiply Real times Complex  
    V is X * Y, W is X * Z.
$cis$ versus $is$

• Could we use $is$ instead of introducing $cis$?

• Why or why not?
Origin of Relations

• Prolog keeps track of the file where a relation was loaded from.

• What happens when you load two files in Prolog & both contain defns of name/arity?
  – The latter’s clauses clobber the former’s?
  – The latter’s clauses are added to the former’s?
  – The latter’s are ignored?
Relation Collision

- If reload same file then relation is redefined.

- Else if relation is system relation, then warning generated and latter’s clauses are ignored.

- Otherwise, SICStus asks the user what to do.
Summary

• Difference between representation and presentation.
• Operator specifications affect presentation.
• Op specs describe the precedence, type, and functor to be associated with the op.
• Ops can be used with relations or with data structures.
Summary cont’d

- Arithmetic expressions are just data structures.
- \textit{is}/2 and the relational operators evaluate arithmetic expressions and require all variables in expressions to be bound.
- Relations are associated with the file where their defns loaded from, this allows collisions to be detected.