Arithmetic in Prolog

TK2ICM: Logic Programming (2nd Term 2017-2018)

M. Arzaki

Computing Laboratory, School of Computing
Telkom University

SoC Tel-U

February 2018

Acknowledgements

This slide is compiled using the materials in the following sources:
Books:

1. P. Blackburn, J. Bos, K. Striegnitz, *Learn Prolog Now!* (Chapter 1-6, 10,11), London: College Publications (available online), 2006. **[LPN]**
2. M. Bramer, *Logic Programming with Prolog* (Chapter 1-9), 2nd Edition, Springer, 2013. **[LPwP]**
3. I. Bratko, *Prolog Programming for Artificial Intelligence* (Chapter 1-3, 5,6,8,9), Pearson Education, 2001. (advanced reference). **[PPAI]**
4. K. H. Rosen, *Discrete Mathematics and Its Applications* (Chapter1), 7th Edition, 2012.
5. M. Ben-Ari, *Mathematical Logic for Computer Science* (Logic Programming Sections), 2nd Edition, 2000.

Lecture slides and lecture notes:

1. *Prolog Programming* by Kristina Striegnitz.
2. *Learn Prolog Now!* by Patrick Blackburn, Johan Bos, and Kristina Striegnitz.
3. *Logic Programming* at Fasilkom UI by A. A. Krisnadhi and A. Saptawijaya.
4. *Computational Logic Part 2: Logic Programming* at Fasilkom UI by L. Y. Stefanus.
5. *Logic Programming* at ILLC, University of Amsterdam by U. Endriss.
6. *Functional Programming* at Fasilkom UI by A. Azurat.
7. *Bahasa Prolog* at FPMIPA UPI by Munir.
8. Other available sources online.

Some of the pictures are taken from the above resources. This slide is intended for academic purpose at FIF Telkom University. You are prohibited for using these slides for professional purpose outside Telkom University, unless with the author authorization. If you have any suggestions/comments/questions related with the material on this slide, send an email to `<pleasedontspam>@telkomuniversity.ac.id`.

Contents

Contents

Basic Arithmetic in Prolog

- Prolog provides a number of basic arithmetic tools for integers and real numbers.
- The general arithmetic predicates all handle expressions.
- An expression is either a simple number or a function. The arguments of a function are expressions.
- Note that '=' does not cause any arithmetic operation.
- Arithmetic computation is achieved using the built-in predicate `is/2`, which is predefined as an infix operator and thus is written between its two arguments.

Arithmetic Notations in Prolog

Arithmetic examples	Prolog Notation
$6+2=8$	`8 is 6+2.`
$6*2=12$	`12 is 6*2.`
$6-2=4$	`4 is 6-2.`
$6-8=-2$	`-2 is 6-8.`
$6\div 2=3$	`3 is 6/2.`
$7\div 2=3$	`3 is 7/2.`
1 is the remainder when 7 is divided by 2	`1 is mod(7,2).`

Arithmetic Using Variables

In Prolog, we can work out the answers to arithmetic questions by using variables.

Exercise

Try to pose the following queries in your Prolog interpreter:

1. `?- X is -7+2.`
2. `?- X is -7*2.`
3. `?- X is -7/2.`
4. `?- X is mod(-7,2).`
5. `?- X is div(-7,2).`
6. `?- X is -7//2.`
   `% a//b gives the integer part of a divided by b.`
7. `?- X is 2^(-7).`
8. `?- X is 2**7.`
9. `?- X is 2+7*3.`
10. `?- X is (2+7)*3.`

Defining Predicates with Arithmetic

- We can use arithmetic operations when we define predicates.
- Let's define a predicate `add3andtimes2/2`
- `add3andtimes2` has two arguments, both are numbers.
- `add3andtimes2` takes its first argument, adds three to it, doubles the result, and returns the number obtained as the second argument.
- We can define `add3andtimes2` as follows:

Defining Predicates with Arithmetic

- We can use arithmetic operations when we define predicates.
- Let's define a predicate `add3andtimes2/2`
- `add3andtimes2` has two arguments, both are numbers.
- `add3andtimes2` takes its first argument, adds three to it, doubles the result, and returns the number obtained as the second argument.
- We can define `add3andtimes2` as follows:
  ```
  add3andtimes2(X,Y):- Y is (X+3)*2.
  ```

Contents

Important Remarks

- It is important to know that `+`, `-`, `*`, and `/` do not carry out any arithmetic.
- Expressions such as `3+2`, `4-7`, `5/5` are ordinary Prolog terms:
  - Functor: `+`, `-`, `/`.
  - Arity: 2.
  - Arguments: integers.

Exercise

Try to pose the following queries in your Prolog interpreter:

1. `?- X = -7+2.`
2. `?- -7+2 = X.`

Remark

To force Prolog to actually evaluate arithmetic expressions, we have to use the predicate `is/2`.

- This is an instruction for Prolog to carry out calculations.
- Because this is not an ordinary Prolog predicate, there are some restrictions.

Limitation of `is/2` Predicate

Exercise

Try to pose the following queries in your Prolog interpreter:

1. `?- X is -7+2.`
2. `?- -7+2 is X.`
3. `?- X is 3+3+3.`
4. `?- X is +(+(3,3),3).`
5. `?- X is 3+3*3.`
6. `?- X is *(+(3,3),3).`

Remark

Limitation of `is/2` Predicate

Exercise

Try to pose the following queries in your Prolog interpreter:

1. `?- X is -7+2.`
2. `?- -7+2 is X.`
3. `?- X is 3+3+3.`
4. `?- X is +(+(3,3),3).`
5. `?- X is 3+3*3.`
6. `?- X is *(+(3,3),3).`

Remark

1. We are free to use variables on the **right hand side** of the `is/2` predicate.
2. But when Prolog actually carries out the evaluation, the variables must be instantiated with a variable-free Prolog term.
3. This Prolog term must be an arithmetic expression.

Arithmetic Functors

3+2, 4/2, 4-5 are just ordinary Prolog terms in a user-friendly notation:

- 3+2 can be expressed as `+(3,2).`
- 4/2 can be expressed as `/(4,2).`
- 4-5 can be expressed as `-(4,5).`

Contents

between/3, succ/2, and plus/3

- `between(+Low, +High, ?Value)`
  `Low` and `High` are integers. `High` $\leq$ `Low`. If `Value` is an integer, then `between(+Low, +High, ?Value)` is true if `Low` $\leq$ `Value` $\leq$ `High`. When `Value` is a variable, it is successively bound to all integers between `Low` and `High`. The `+Low` and `+High` notations mean **both arguments must be instantiated**.
- `succ(?Int1, ?Int2)`
  The predicate succeeds if `Int2` is equal to `Int1 + 1` (i.e., `Int2` is the successor of `Int1`). **At least one of the arguments must be instantiated to an integer**.
- `plus(?Int1, ?Int2, ?Int3)`
  The predicate succeeds if `Int3` is equal to `Int1+Int2`. **At least two of the three arguments must be instantiated to integers**.

Exercise

Try to pose the following queries in your Prolog interpreter:

1. `?- between(10,20,X).`
2. `?- succ(2018,X).`
3. `?- succ(X,2018).`
4. `?- plus(2018,X,2020).`
5. `?- plus(X,2017,2020).`

Other Arithmetic Predicates (Relational Operators)

- `Expr1 > Expr2`
  The predicate succeeds when expression `Expr1` evaluates to a larger number than `Expr2`.
- `Expr1 < Expr2`
  The predicate succeeds when expression `Expr1` evaluates to a smaller number than `Expr2`.
- `Expr1 =< Expr2`
  The predicate succeeds when expression `Expr1` evaluates to a number that is smaller than or equal to `Expr2`.
- `Expr1 >= Expr2`
  The predicate succeeds when expression `Expr1` evaluates to a number that is larger than or equal to `Expr2`.

Remark

We must avoid using => and <= in Prolog as a relational operators.

Arithmetic Equality and Inequality

- `Expr1 =\= Expr2`
  The predicate succeeds when expression `Expr1` evaluates to a number not equal to `Expr2`.
- `Expr1 =:= Expr2`
  The predicate succeeds when expression `Expr1` evaluates to a number equal to `Expr2`.

Arithmetic Functions

Arithmetic functions are terms which are evaluated by the arithmetic predicates, such as `is/2`.

- `-Expr` denotes minus.
- `Expr1 + Expr2` denotes addition.
- `Expr1 - Expr2` denotes subtraction.
- `Expr1 * Expr2` denotes multiplication.
- `Expr1 / Expr2` denotes division.
- `Expr1 // Expr2` denotes integer divison, `Expr1 // Expr2` gives the integer part of `Expr1` divided by `Expr2`.
- `Expr1 ^ Expr2` denotes power operation. This notation is also equivalent to `Expr1 ** Expr2`.
- `IntExpr1 mod IntExpr2` denotes mod operation, both arguments `IntExpr1` and `IntExpr2` must be integers and `IntExpr2` cannot be zero.
- `IntExpr1 div IntExpr2` denotes div operation, both arguments `IntExpr1` and `IntExpr2` must be integers and `IntExpr2` cannot be zero.
- `IntExpr1 rem IntExpr2` gives the remainder the division.

- `exp(Expr)` gives $e^{\texttt{Expr}}$ where $e = 2.7182....$
- `abs(Expr)` gives the absolute value of `Expr`.
- `sin(Expr)` gives the sine of `Expr` (`Expr` is measured in radian).
- `cos(Expr)` gives the cosine of `Expr` (`Expr` is measured in radian).
- `asin(Expr)` gives the $\arcsin(\texttt{Expr})$, i.e., the angle (measured in radian).
- `acos(Expr)` gives the $\arccos(\texttt{Expr})$, i.e., the angle (measured in radian).
- `max(Expr1,Expr2)` gives the maximum value between `Expr1` and `Expr2`.
- `round(Expr)` returns the nearest integer of `Expr`.
- `sqrt(Expr)` returns the square root of `Expr`.
- `log(Expr)` returns $\ln(\texttt{Expr})$.
- `log10(Expr)` returns $\log_{10}(\texttt{Expr})$.
- `pi` returns the mathematical constant $\pi = 3.1415....$
- `e` returns the mathematical constant $e = 2.7182....$

Contents

Elementary Exercise

Exercise

Construct a predicate `double(X,Y)` that returns true whenever `X` is equal to `2Y`. The predicate succeeds whenever one of its argument is instantiated. For example:

1. `double(3,6)` returns **true** (there might be some side effects though).
2. `double(3,X)` returns `X = 6` (there might be some side effects though).
3. `double(X,6)` returns `X = 3` (there might be some side effects though).

Exercise

Construct a predicate `average(X,Y,Z)` that returns true whenever `Z` is the average of `X` and `Y`. The predicate succeeds whenever two of its argument are instantiated. For example:

1. `average(3,5,4)` returns **true** (there might be some side effects though).
2. `average(3,5,X)` returns `X = 4`.(there might be some side effects though).
3. `average(3,X,4)` returns `X = 5`.(there might be some side effects though).
4. `average(X,5,4)` returns `X = 3`.(there might be some side effects though).

Euclidean Distance

Exercise

Given two points $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$, the Euclidean distance between these points, denoted by d, is defined as

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Construct a predicate `distance(point(A,B),point(C,D),Distance)` that is true whenever `Distance` is the euclidean distance between $point\,(A, B)$ and $point\,(C, D)$, both of `point(A,B)` and `point(C,D)` must be instantiated. For example, we have:

- `distance(point(0,0),point(3,4),Distance)` returns `Distance = 5`.
- `distance(point(1,2),point(2,1),Distance)` returns `Distance = 1.414....`

Strange Squares

Exercise

Observe the following phenomenon:

- $45^2 = 2025$
- If we split 2025 into two parts, we have 20 and 25.
- If we add $20 + 25$, we get 45.

Find other **four digit pefect squares** that exhibit the same peculiarity! Construct a predicate `strange_square(M,N)` that is true whenever `M` is a "strange square" and `N` is equal to $\texttt{M}^2$. For example:

- `strange_square(45,2025)` returns **true**.
- `strange_square(45,N)` returns `N = 2025`.
- `strange_square(M,2025)` returns `M = 45`.
- `strange_square(25,N)` returns **false**.

Arithmetic Series

Exercise

Construct a predicate `sumseries(M,N)` that returns true whenever N is equal to the sum $1 + 2 + 3 + \cdots + M$. For instance:

1. `sumseries(1,1)` returns **true**
2. `sumseries(10,N)` returns `N = 55` because $1 + 2 + 3 + \cdots + 10 = 55$.

You may define the predicate `sumseries/2` recursively. Notice that `sumseries(1,1)` is the base case. You have to define the recursive predicate `sumseries(M,N)` for the case $M \neq 1$. You may complete the following rule: `sumseries(M,N):- M > 1,...`.
Ignoring the side effect, you must obtain:

1. `sumseries(10,N)` returns `N = 55`
2. `sumseries(100,N)` returns `N = 5050`.

(Note: the first argument, i.e., `M`, must be instantiated).