Operators in Prolog

TK2ICM: Logic Programming (2nd Term 2018-2019)

M. Arzaki

Computing Laboratory, School of Computing
Telkom University

SoC Tel-U

February 2019

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

1. Operators in Prolog
2. Predefined Operators in Prolog
3. Defining New Operators in Prolog
4. Exercise
5. Defining Logical Operators

Contents

1. Operators in Prolog
2. Predefined Operators in Prolog
3. Defining New Operators in Prolog
4. Exercise
5. Defining Logical Operators

Operators in Prolog

- Operators are defined to improve the readability of source-code.
- For example, without operators, to write `a*b+c*d`, one would have to write: `+(*(a,b),*(c,d))`.
- A number of operators have been predefined.
- In Prolog, all operators, except for the comma (`,`) can be redefined by the user.
- The notation for arithmetic operators was an example.
- Internally, Prolog will use `is(11,+(2,*(3,3)))`, but Prolog allows us to write `11 is 2+(3*3)` instead.

Increasing Readability with Operators in Prolog

- Operators in Prolog can be used to enhance the readability of the source code.
- Up to now, the notation used for predicates in this book is the standard one of a functor followed by a number of arguments in parentheses, e.g., `likes(john,mary)`.
- As an alternative, any user-defined predicate with two arguments (a *binary predicate*) can be converted to an *infix operator*.
- This enables the functor (predicate name) to be written between the two arguments with no parentheses, e.g., `john likes mary`.

Contents

1. Operators in Prolog
2. Predefined Operators in Prolog
3. Defining New Operators in Prolog
4. Exercise
5. Defining Logical Operators

Predefined Operators in Prolog

Precedence	Type	Examples
1200	$x\ f\ x$	`- ->`, `:-`
1200	$f\ x$	`:-`, `?-`
1150	$f\ x$	`dynamic`, `discontiguous`, `initialization`, `module_transparent`, `multifile`, `thread_local`, `volatile`
1100	$x\ f\ y$	`;`, |
1050	$x\ f\ y$	`->`, `op*->`
1000	$x\ f\ y$	`,`
954	$x\ f\ y$	`\`
900	$f\ y$	`\+`
900	$f\ y$	`~`

Precedence	Type	Examples
700	$x\ f\ x$	<, =, =..., =@=, =:=, =<, ==, =\=, >, >=, @<, @=<, @>, @>=,\=, \==, is
600	$x\ f\ y$	:
500	$y\ f\ x$	+, -, /\, \/, xor
500	$f\ x$	+, -, ?, \
400	$x\ f\ x$	*, /, //, rdiv, <<, >>, mod, rem
200	$x\ f\ x$	**
200	$x\ f\ y$	^

Contents

1 Operators in Prolog

2 Predefined Operators in Prolog

3 Defining New Operators in Prolog

4 Exercise

5 Defining Logical Operators

Defining New Operators in Prolog

The command `op(+Precedence, +Type, :Name)` declares `Name` to be an operator of type `Type` with precedence `Precedence`.

- `Name` can also be a list of name, in which case all elements of the list are declared to be identical operators (we'll study list later, mostly after the midterm).
- `Precedence` is an integer between 0 and 1200 (inclusive). Precedence 0 removes the declaration. The higher the value, the lower the binding power of the operator.
- Type is one of the following:
  - $x\ f$ (postfix operator)
  - $y\ f$ (postfix operator)
  - $x\ f\ x$ (infix operator)
  - $x\ f\ y$ (infix operator)
  - $y\ f\ x$ (infix operator)
  - $f\ y$ (prefix operator)
  - $f\ x$ (prefix operator)
- The f indicates the functor, while x and y indicate the position of arguments.

- 'y' should be interpreted as: "on this position a term with **precedence lower or equal to the precedence of the functor** should occur".
- For 'x' the **precedence of the argument must be strictly lower** than the precedence of the functor.
- The precedence of a term is 0, unless its principal functor is an operator, in which case the precedence is the precedence of this operator.
- A term enclosed in brackets (...) has precedence 0.

Examples of Operator Definition

- Programmer can define new operators by inserting into the program special kinds of clauses, called directives, starting with ":-".
- Note that operator definition do not specify any operation or action.

The predicate suka

```
:- op(650, xfx, suka).
alia suka burger.
alia suka mie.
amin suka burger.
amin suka sate.
bambang suka bakso.
bambang suka mie.
caca suka sate.
caca suka rendang.
```

- We can ask the following query:

Examples of Operator Definition

- Programmer can define new operators by inserting into the program special kinds of clauses, called directives, starting with ":-".
- Note that operator definition do not specify any operation or action.

The predicate suka

```
:- op(650, xfx, suka).
alia suka burger.
alia suka mie.
amin suka burger.
amin suka sate.
bambang suka bakso.
bambang suka mie.
caca suka sate.
caca suka rendang.
```

- We can ask the following query:
  - `?- alia suka Apa`.

Examples of Operator Definition

- Programmer can define new operators by inserting into the program special kinds of clauses, called directives, starting with ":-".
- Note that operator definition do not specify any operation or action.

The predicate suka

```
:- op(650, xfx, suka).
alia suka burger.
alia suka mie.
amin suka burger.
amin suka sate.
bambang suka bakso.
bambang suka mie.
caca suka sate.
caca suka rendang.
```

- We can ask the following query:
  - `?- alia suka Apa.`
  - `?- Siapa suka sate.`

We can extend the aforementioned knowledge base with the following fact.

```
:- op(650,xf,pedas).
mie pedas.
rendang pedas.

:- op(650,xf,gurih).
burger gurih.
bakso gurih.

:- op(650,xf,manis).
sate manis.
```

We can ask the following query:

- `?- alia suka Apa, Apa pedas.`
- `?- Siapa suka Apa, Apa gurih.`

Examples of Operator Definition: Precedence

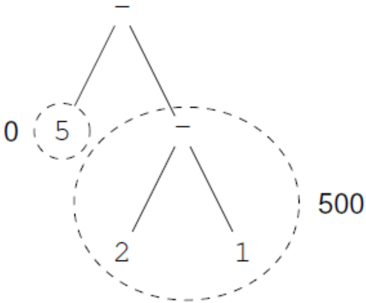
- Suppose we define an infix operator '-' of type yfx as follows:
```
:- op(500, yfx, -).
```

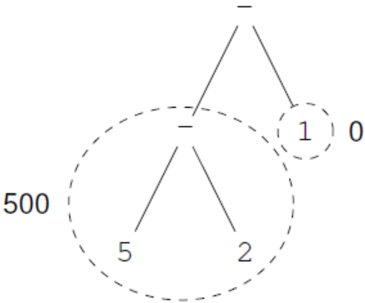
Examples of Operator Definition: Precedence

- Suppose we define an infix operator '-' of type yfx as follows:
  `:- op(500, yfx, -).`
- The operator '-' is defined with precedence 500 and type yfx.

Examples of Operator Definition: Precedence

- Suppose we define an infix operator '-' of type yfx as follows:
  `:- op(500, yfx, -).`
- The operator '-' is defined with precedence 500 and type yfx.
- What is the value of `X` if the query `?- X is 5-2-1.` is executed? Would be this interpreted as $5 - (2 - 1)$ or $(5 - 2) - 1$?





Contents

1. Operators in Prolog
2. Predefined Operators in Prolog
3. Defining New Operators in Prolog
4. Exercise
5. Defining Logical Operators

Exercise

Suppose we have the following knowledge base.

```
:- op(650, xfx, suka).
alia suka burger.
alia suka mie.
alia suka balado.

amin suka burger.
amin suka sate.
amin suka permen.

bambang suka bakso.
bambang suka mie.
bambang suka coklat.

caca suka sate.
caca suka rendang.
caca suka eskrim.
```

```
:- op(650,xf,pedas).
mie pedas.
rendang pedas.
balado pedas.

:- op(650,xf,gurih).
burger gurih.
bakso gurih.

:- op(650,xf,manis).
sate manis.
coklat manis.
permen manis.
eskrim manis.
```

Define a predicate `dan` so that we can write following queries:

1. `?- alia suka burger dan alia suka mie.` returns **true**.
2. `?- amin suka burger dan amin suka balado.` returns **false**.
3. `?- Siapa suka Apa dan Apa pedas.` returns `Siapa = alia, Apa = mie; Siapa = alia, Apa = balado; Siapa = bambang, Apa = mie; Siapa = caca, Apa = rendang;`
4. `?- Siapa suka sate dan Siapa suka rendang dan Siapa suka eskrim.` returns `Siapa = caca`.

Problem (Challenging Problem)

Define the predicate `dan` so the following query is possible:
`Siapa suka sate dan rendang dan eskrim.` returns `Siapa = caca`.

Contents

1. Operators in Prolog
2. Predefined Operators in Prolog
3. Defining New Operators in Prolog
4. Exercise
5. Defining Logical Operators

Defining Logical Operators

Prolog allows us to define logical operators such as: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ implication, and biimplication ($\leftrightarrow$). From Mathematical Logic, we know the following predence order of logical operators: $\neg$ (highest), $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ (lowest).

```
:- op(900,xfx, <=>).
:- op(800,xfy,=>).
:- op(700,xfy,v).
:- op(600,xfy,&).
:- op(500,fy,~).

~A:- not(A).
A & B:- A,B.
A v B:- A;B.
A => B:- ~A v B.
A <=> B:- A => B, B => A.
```

- We use `=>` and `<=>` instead of `->` and `<->` (respectively) because Prolog the operator `->` is one of the Prolog built-in predicate for static procedure.
- We use `true` and `false` in Prolog to denote the propositional constants true ($\top$) and false ($\perp$), respectively.

Exercise

Exercise

Test the previously defined logical operators using the propositional constants `true` and `false`, e.g.:

1. ∼`true` returns `false`, ∼`false` returns `true`,
2. `true & true` returns `true` and returns `false` otherwise,
3. `false v false` returns `false` and `true` otherwise,
4. `true => false` returns `false` and `true` otherwise,
5. `true <=> true` and `false <=> false` returns `true` and `false` otherwise.

Is there any problem with our definitions?

Exercise

Define an operator `xor` such that `A xor B` returns true whenever `A` and `B` have different truth values.