Operators (2A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Terms (1)

Atoms

Numbers

Constants

Simple Terms

Variables

Compound Terms

functor(term$_1$, term$_2$, ... , term$_n$)

Atoms

Compound Terms

Predicates

A term without any variables

A ground term

Terms (2)

Atoms

A string of characters
Characters enclosed in single quotation
A string of special characters

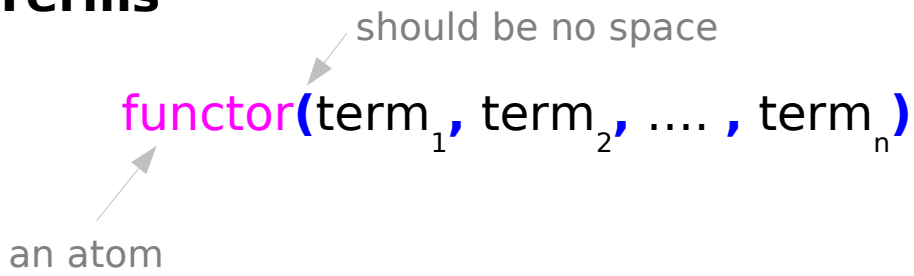
Numbers

Integer numbers

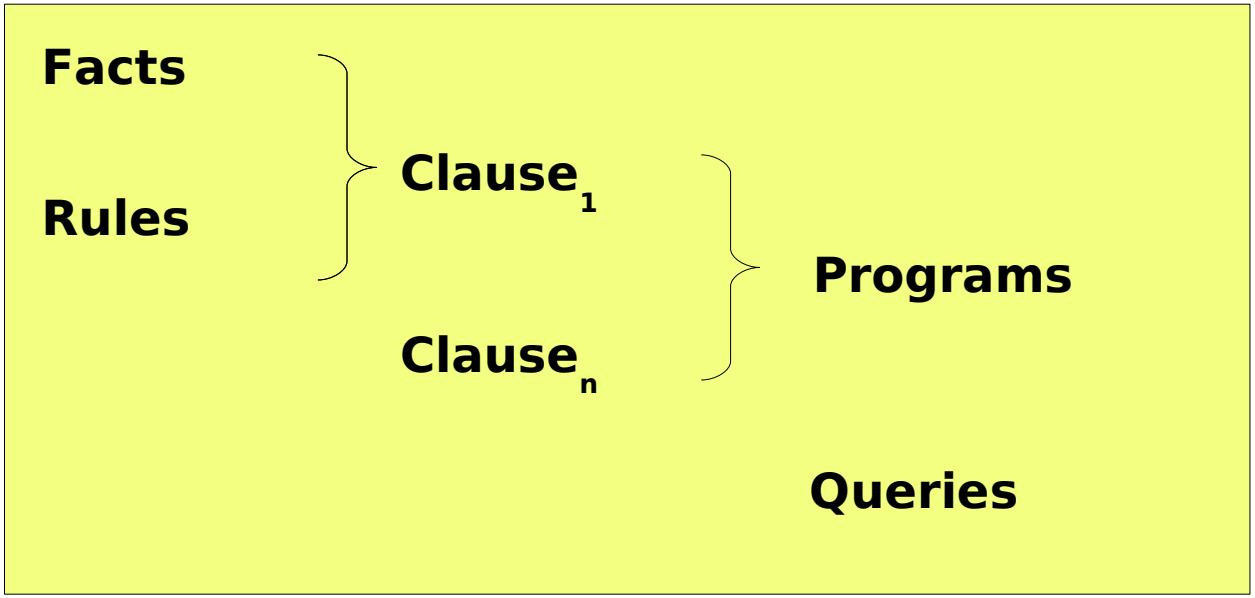
Variables

Variables:
Starting with a capital letter or _
Anonymous variable _ :
Multiple occurrence in one expression are assumed to be distinct

Compound Terms



Clause



Facts A predicate followed by a dot.
A functor, an atom

Rules
- **a head :-** A predicate (a functor, an atom)
- **a body .** A sequence of predicates separated by **,**'s

Precedence

Every operator — associated with an integer number

[0, 1200]

higher priority — lower priority

SWI-prolog

* → 400

\+ → 500

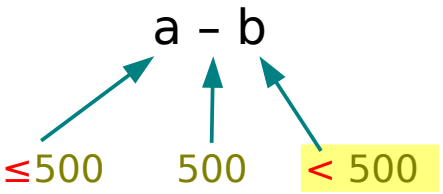
a term → If its principle functor is an operator
then the precedence of an operator

Else the precedence is defined as 0

5 + 7	precedence of 500
5*5 + 7*7	precedence of 500
sqrt(5 + 7)	precedence of 0
man	precedence of 0
3 * +(5, 7)	precedence of 400

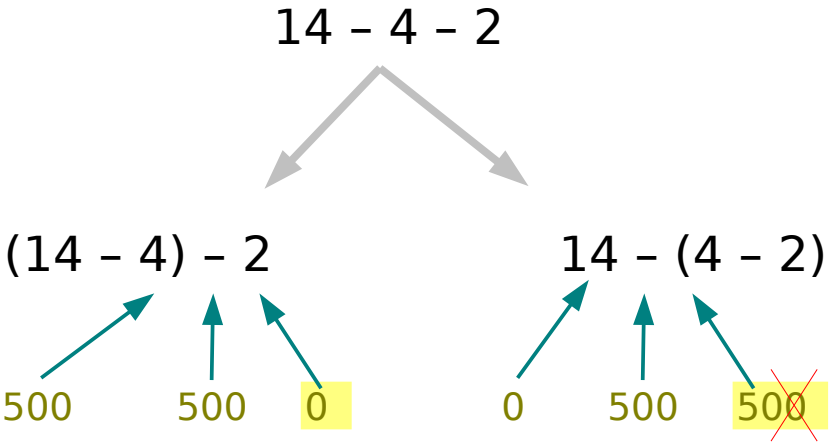
Associativity

- Infix operators
- Prefix operators
- Postfix operators



the precedence of **x** < the precedence of **f**

the precedence of **y** ≤ the precedence of **f**



Left Associative operator

2 * 3 * 4 * 5

((2 * 3) * 4) * 5

Left Associative operator

Disjunction Operator

Infix operators
Prefix operators
Postfix operators

a, b, c, d

a, (b, c, d)

a, (b, (c, d))

Right Associative operator

2 ** 3 ** 4 **5

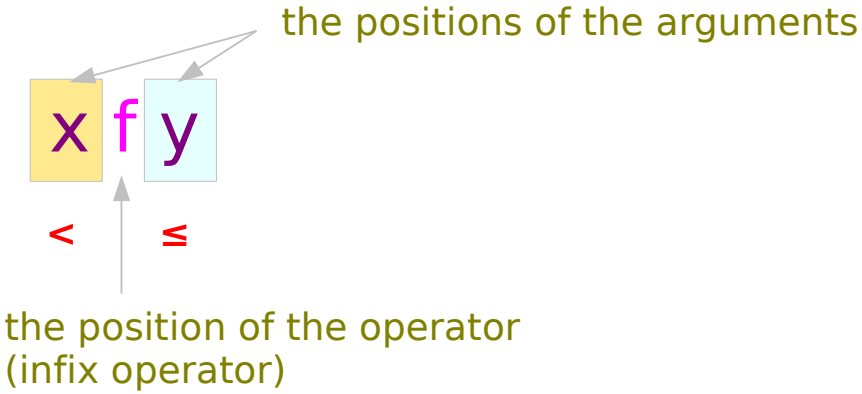
2 ^ 3 ^ 4 ^5

2 ^ (3 ^ 4 ^5)

2 ^ (3 ^ (4 ^5))

SWI-prolog ^ → **

Right Associative operator



the disjunction operator ; or |

```
head :- body1.
head :- body2.
```

⇔ `head :- body1 ; body2.`

```
head :- b1 ; b2; b3; b4.
head :- (b1 ; (b2; b3; b4)).
head :- (b1 ; (b2; (b3; b4))).
```

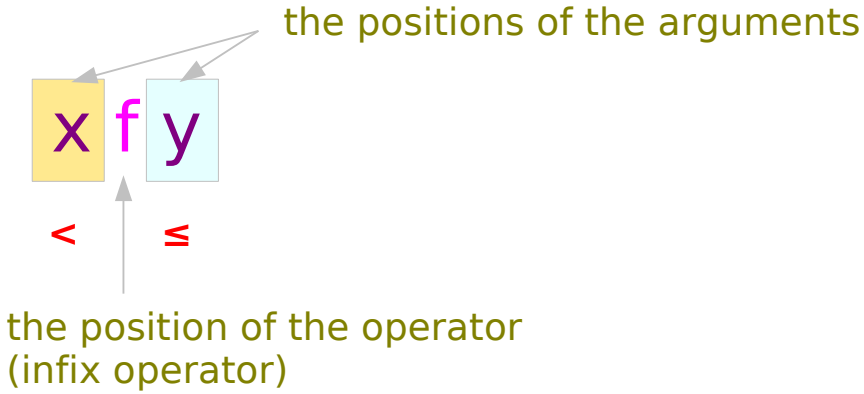
Right Associative operator

Comma Operator

Comma Sequences

No empty sequence
The shortest sequence
– one element

:- op(1000, xfy, ‘,’);



?- (H, T) = (1,2,3);
H = 1
T = 2, 3, 4

?- (a) = a.
Yes

?- (A,B,C) = (1,2,3,4,5,6).
A = 1
B = 2
C = 3,4,5

Right Associative operator

Prolog clauses use comma sequences.

(A,B,C) = (1,(2,(3,4,5,6)))

Negation As Failure

```
bachelor(P) :- male(P), not(married(P)).

male(henry).

male(tom).

married(tom).
```

```
not not not married(P)
not (not not married(P))
not (not (not married(P)))
not (not (not (married(P)))
```

Right Associative operator

```
?- bachelor(henry).
Yes

?-bachelor(tom).
No

?-bachelor(Who).
Who=henry;
No
```

```
?- not(married(Who)).
No.
```

For the variable binding **Who**=tom,
married(**Who**) succeeds
not(married(**Who**)) fails

Negative goals with variables **cannot** be expected to produce **bindings** of the variables for which the goals fails

NAF (Negation As Failure)

PLANNER

if (not (goal p)), then (assert ¬p)

If the goal to prove p fails, then assert ¬p

If NAF used to derive not p (p is assumed not to hold) from failure to derive p

Not p can be different from the statement ¬p of the logical negation of p, depending on the **completeness** of the inference algorithm and thus also on the formal logic system

Prolog

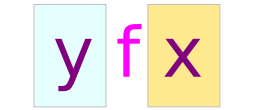
NAF literals of the form of not p can occur in the body of clauses

Can be used to derive other NAF literals

p ← q ∧ not r
q ← s
q ← t
t

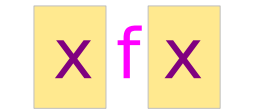
Infix Operators

INFIX Operators

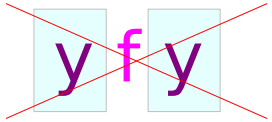


left-associative +, –, *, /

right-associative , (subgoals)



non-associative =, is, < (no nesting)



Not possible

Prefix & Postfix Operators

PREFIX Operators

f x non-associative – (--3 not possible)

f y right-associative , (subgoals)

POSTFIX Operators

x f non-associative

y f left-associative

Operator Examples

```
?- current_op(Precedence, Associativity, *).
Precedence = 400
Associativity = yfx       left-associative
Yes

?- current_op(Precedence, Associativity, **).
Precedence = 200
Associativity = xfx ;     non-associative
No

?- current_op(Precedence, Associativity, -).
Precedence = 500
Associativity = yfx ;     left-associative
Precedence = 500
Associativity = fx ;
No

?- current_op(Precedence, Associativity, <).
Precedence = 700
Associativity = xfx ;     non-associative
No
```

```
?- current_op(Precedence, Associativity, =).
Precedence = 700
Associativity = xfx ;    non-associative
No

?- current_op(Precedence, Associativity, :-).
Precedence = 1200
Associativity = xfx ;    non-associative
Precedence = 1200
Associativity = fx ;
No
```

Operator Examples

:=	xfx, fx	non-associative
?-	fx	non-associative
;	xfy	right associative
,	xfy	right associative
not	fy	right associative
is, =.., <, etc.	xfx	non-associative
+, –	yfx, fx	left associative
*, /	yfx	left associative
^	xfy	right associative

Large P

Small P

References

[1] U. Endriss, “Lecture Notes : Introduction to Prolog Programming”
[2] http://www.learnprolognow.org/ Learn Prolog Now!
[3] http://www.csupomona.edu/~jrfisher/www/prolog_tutorial
[4] www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/intro.html