University of Auckland

07.363

Functional and Logic Programming

Christian Collberg

October 7, 1994

Operators

Operators I

User-defined operators

- work like built-in operators.
- add no power to Prolog.
- make programs easier to read.
- provide the same functionality as functors, but with more natural syntax:

likes(chris, björk)	chris likes björk
goalie(ravelli)	goalie ravelli
implies(a, b)	a `==>` b

Slide 7–1

Operators II

position Does the operator come before, after, or inbetween the arguments?

- infix, prefix, postfix

precedence What is the binding power of the operators?

associativity How are operators of equal binding power grouped?

Slide 7–2

Position

infix `xfx`, `xfy`, `yfx`, `yfy`

- $a * b$

prefix `fx`, `fy`

- `not` d

postfix `xf`, `yf`

- $a!$

Slide 7–3

Precedence

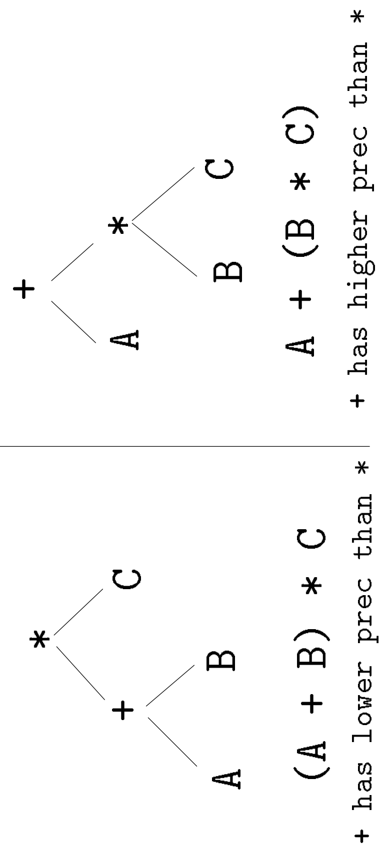
- Used to disambiguate expressions
- Is $a * b + c = (a * b) + c$ or $a * (b + c)$?
- Higher precedence means *lower* binding power (in Prolog).
- Goes from 1 to `Max`, where `Max` is implementation defined.

PRECEDENCE	OPERATOR
500	$+, -$
400	$*, /,$ `div`
300	`mod`

Slide 7–4

Precedence

`A + B * C`



Slide 7–5

Operators V – Associativity

- Used to disambiguate expressions with operators of same precedence.
- Is $100/25/5 =$
  - $(100/25)/5 = \frac{4}{5}$ (left associative), or
  - $100/(25/5) = 20$ (right associative)?
- In `xfx, yfx, xfx, yfx`

  y **means that** argument can contain operators of the same or lower precedence than f.

Slide 7–6

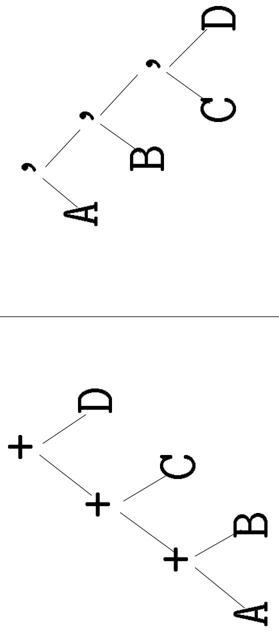
Associativity II

- In `xfx, yfx, xfx, yfx`

  x **means that** argument can contain operators of strictly higher precedence than f.
- If $+$ is `yfx` then
  - $a + b + c = (a + b) + c$
  - $a + b + c \neq a + (b + c)$ ($(b + c)$ has *same* precedence as a, rather than higher).
- An operator declared
  - `yfx`, is left associative
  - `xfy`, is right associative
  - `xfx`, has no associativity ($a = b = c$ is illegal)

Slide 7–7

Associativity



op(1000, xfy, ',')
Right Associative

op(500, yfx, +)
Left Associative

Slide 7–8

Example – Bedtime Story I

"Helder, a poor scientist, was in love with the daughter of an admiral. One day, a general captured the girl. Helder rode to the general's barrack and killed the general. The girl was grateful and fell in love with Helder. The admiral was so happy to have his daughter back he gave Helder half of all his boats."

- "Who is the father of the girl?"
- "Who is rich?"
- "Who loves who?"
- "Who is poor?"
- "Who captured who?"
- "Who killed who?"
- "Who lived happily ever after?"

Slide 7–9

Example – Bedtime Story II

```
:- op(500, xfy, 'is_').
:- op(500, yfx, 'loves').
:- op(500, yfx, 'kills').
:- op(500, yfx, 'to').
:- op(500, yfx, 'captures').
:- op(500, yfx, 'rides_to').
:- op(500, yfx, 'gives').
:- op(500, yfx, 'is_father_of').
:- op(800, yfx, 'and').

X and Y :- X, Y.
```

Slide 7–10

Example – Bedtime Story III

```
helder is_ poor.  admiral is_ happy.
admiral is_father_of girl.
helder loves girl.
girl loves helder.
general captures girl.
helder kills general.
admiral gives half_boats to helder.

% Who loves who?
q2 :- Z loves Y, write(Z),
      write(' loves '),
      write(Y), nl, fail.
q2.
?- q2.
   helder loves girl
   girl loves helder
```

Slide 7–11

Example – Bedtime Story IV

```
% Who captures who?
?- Z captures Y.
   Z = general
   Y = girl

% Who kills who?
?- Z kills Y.
   Z = helder
   Y = general

% Who loves who's daughter?
?- Z loves G and F is_father_of G.
   Z = helder
   G = girl
   F = admiral
```

Slide 7–12

Example – Theorem Proving I

```
:- op(700, xfy, <==>).
:- op(650, xfy, ==>).
:- op(600, xfy, v).
:- op(550, xfy, &).
:- op(500, fy, ~).

thm(T) :-
   (false(T, U),
      write('not valid'), !;
   false(~T, U),
      write('valid')),
   write('; rules:'), write(U),nl.
```

Slide 7–13

Example – Theorem Proving III

```
false(f, [rule1]) :- !.
false(~ t, [rule2]) :- !.

false(P <==> Q, [rule3|U]) :-
   false((P ==> Q) &
         (Q ==> P), U).

false(P ==> Q, [rule4|U]) :-
   false(~P v Q, U).

false(P v Q, [rule5|U]) :-
   false(P, U1), false(Q, U2),
   append(U1, U2, U).
false(P & Q, [rule6|U]) :-
   (false(P, U) ; false(Q, U)).
```

Slide 7–14

Example – Theorem Proving IV

```
false(~ ~ P, [rule7|U]) :-
   false(P, U).

false(~(P <==> Q), [rule8|U]) :-
   false(~(P ==> Q) v
   ~(Q ==> P), U).

false(~(P ==> Q), [rule9|U]) :-
   false(~(~P v Q), U).

false(~(P v Q), [rule10|U]) :-
   false(~P & ~Q, U).

false(~(P & Q), [rule11|U]) :-
   false(~P v ~Q, U).
```

Slide 7–15

Slide 7–16

Example – Theorem Proving V

```
?- thm(f & t).
not valid; rules:  [rule6,rule1]

?- thm(f v t).
   valid; rules:  [rule10,rule6,rule2]

?- thm(t <==> f).
   not valid;
   rules:  [rule3,rule6,rule4,rule5,rule2,rule1]
```

Slide 7–17

Example – Theorem Proving VI