Übersicht

Operatoren…

… sind Funktoren, an die nicht direkt eine Klammer folgt.

- Infix-, Präfix-, Postfix-Schreibweise
- Präzedenz, Assoziativität, Position
- Vordefinierte und selbstdefinierte Operatoren
- Operator- vs. Funktor-Argument-Schreibweise
- Klauseln sind Terme!

Infix-Schreibweise

Manchmal ist der Kode lesbarer, wenn die Funktor-Argument-Schreibweise durch Operator-Schreibweise ersetzt wird.

- Zum Bespiel Infix-Schreibweise in Arithmetik

```
-(12,-(3,2))        +(*(2,3),4)
12-3-2              2*3+4
```

- Allerdings bringen Operatoren zusätzlich das Problem der Präzedenz und Assoziativität mit sich!

Operatoren: Man nehme ...

Zu jeder Definition eines Operators gehört

- **Präzedenz** *(precedence; priority)*
  - Zahl zwischen 0 und 1200
  - Je kleiner die Zahl, desto höher die Präzedenz (!).
- **Position**
  - Präfix-Operator: Operator steht *vor* Argument.
  - Infix-Operator: Operator steht *zwischen* Argumenten.
  - Postfix-Operator: Operator steht *nach* Argument.
- **Assoziativität** *(associativity)*
  - linksassoziativ: a • b • c = (a • b) • c
  - rechtsassoziativ: a • b • c = a • (b • c)
  - nicht schachtelbar: a • b • c ist kein zulässiger Term!
- **Name** – Atom

Präzedenz und Assoziativität

f: Funktor

x: nicht-assoziativ

y: assoziativ

Angabe	Bedeutung
`fx`	Präfix, nicht schachtelbar
`fy`	Präfix, rechtsassoziativ
`xf`	Postfix, nicht schachtelbar
`yf`	Postfix, linksassoziativ
`xfx`	Infix, nicht schachtelbar
`xfy`	Infix, rechtsassoziativ
`yfx`	Infix, linksassoziativ

Vordefinierte Operatoren nach ISO-Standard

Präzedenz	Position/Assoz.	Operatoren
1200	xfx	:- -->
1200	fx	:- ?-
1100	xfy	;
1050	xfy	->
1000	xfy	,
900	fy	\+ spy
700	xfx	= \= == \== @< @=< @> @>= is =:= =\= < > >= > =..
500	yfx	+ - / \ \ /
400	yfx	* / // rem mod << >>
200	xfx	^
200	fy	\ -

Selbstdefinierte Operatoren

Selbstdefinierte Operatoren brauchen Deklaration…

```
:- op(600, xfx, vater).
```

Präzedenz
Infix, nicht assoziativ
Name

```
:- op(600, xfy, vater).
hans vater gabi.
hans vater kevin.
```

Mit current_op/3 lassen sich die gegenwärtig definierten Operatoren ausgeben.

```
?- current_op(600, A, F).
A = xfx
F = vater
yes
```


Operator vs. Funktor-Argument-Notation

Die Operator-Schreibweise ist ein syntaktischer Zucker (*syntactic sugar*).

- Für jede Operator-Schreibweise eines Terms gibt es eine Schreibweise in der Funktor-Argument-Form.
- Das eingebaute Prädikat write_canonical/1 schreibt die Funktor-Argument-Notation jedes Terms heraus:

```
?- write_canonical(12/3/2).
/(/(12,3),2)
```

- Alle Operatoren-Prädikate sind auch in der Funktor-Argument-Form aufrufbar:

```
?- =(A,a).
A = a ?
```


Regelklauseln sind Terme

Regel-Klauseln in Funktor-Argument-Notation

- Die Symbole für "falls" `:-`, "und" `,` und "oder" `;` sind vordefinierte zweistellige Operatoren.

```
?- write_canonical((p :- q, r, s)).
:-(p,','(q,','(r, s)))
yes
?- write(:-(p,','(q,','(r,s)))).
p:-q,r,s
yes
```

- Nur der Punkt am Schluss von Fakten und Regeln ist kein Term!
  - Er ist das Zeichen, das Term-Enden markiert!