Abstrakte Syntax von Prolog (1)

Abstrakte und konkrete Syntax:

- Abstrakte Syntax: Nur Datenstrukturen, die der Parser anlegt (z.B. Operatorbaum).
- Konkrete Syntax: Zeichenketten, die der Benutzer eingibt (z.B. Präzedenzen, Klammern, etc.).

Programm:

- Liste von Klauseln.

Klausel:

- Faktum: Literal.
- Regel: Kopf (Literal) und Rumpf (Ziel).
- Anfrage/Kommando: Ziel.

Ziel:

- Literal.
- Konjunktion: Liste von Zielen (und "Cuts").
- Disjunktion: Liste von Zielen.
- Bedingung (Ziel), then-Teil (Ziel), ggf. else-Teil.

Abstrakte Syntax von Prolog (2)

Literal:

- Prädikat und (ggf. leere) Liste von Termen.

Term:

- Variable.
- Atom.
- Funktor und Liste von Termen.
- Zahl (integer oder real).

  Real (Fließkommazahlen) gibt es nur bei besseren Prologs.
- String

  Bei vielen Prologs ist dies eine Liste von ASCII-Codes, also kein eigener Datentyp, sondern nur eine abkürzende Schreibweise.
- Stream (offene Datei).

  Nur bei moderneren Prologs, ansonsten interne Systemvariablen.

Prädikat, Funktor:

- Atom und Stelligkeit (Anzahl Argumente).

  Die Stelligkeit ist in der konkreten Syntax nur implizit angegeben. Ein Atom mit verschiedenen Stelligkeiten sind verschiedene Funktoren.
- System-Prädikat: Atom, Stelligkeit, Prozedur.

  Es gibt keine eingebauten Funktionen, nur eingebaute Prädikate.

Lexikalische Einheiten der Prolog Syntax

Atome:

- Kleinbuchstabe (Buchstabe | Ziffer | _)*

  z.B.: diesIstEinAtom, x27, dies_ist_auch_erlaubt.

- ' (beliebige Zeichen)* '

  Falls in der Zeichenfolge ein ' vorkommt, ist es zu verdoppeln.

- (#|$|&|*|+|-|.|/|:|<|=|>|?|@|\|^|`|~)+

  Aber: "." gefolgt von Leerzeichen markiert das das Ende der Klausel.
  Weitere Ausnahme: Mit "/*" beginnt ein Kommentar.

- Spezielle Atome: `!`, `;`, `[]`, `{}`.

Zahlen:

- z.B. `23`, `-765`, `16'1F` (=31), `0'a` (=97)

  Natürlich gibt es auch Float-Zahlen (bei kommerziellen Systemen).

Variablen:

- (Großbuchstabe | _) (Buchstabe | Ziffer | _)*

  Anonyme Variable: „_“ (bei jedem Auftreten eine neue Variable)

Kommentare:

- Von „`%`“ bis zum Zeilenende (wie in TEX).
- Von „`/*`“ bis „`*/`“ (wie in C).

Operator-Syntax (1)

Beispiel:

- "`+(1, 1)`" ist ein Term in Standardsyntax.
- "`1 + 1`" ist derselbe Term in Operator-Syntax.

Allgemeines:

- Nur bequemere Syntax, Bedeutung völlig gleich.
- Auch für Literale möglich, z.B. "`X \= Y`".
- Operatoren müssen deklariert werden.
  Viele sind vordefiniert, aber der Benutzer kann die Sprache erweitern.

Operatoren:

- Name (beliebiges Prolog-Atom).
- Priorität: 1 (stark bindend) bis 1200 (schwach).
  Z.B.: `*`: 400, `+`: 500 ("Punktrechnung vor Strichrechnung").
- Assoziativität: fx, fy, xf, yf, xfx, yfx, xfy.

Assoziativitäten (1):

- fx, fy: Präfix-Operator, z.B. "`-X`".
- xf, yf: Postfix-Operator, z.B. "`7!`".
- xfx, yfx, xfy: Infix-Operator, z.B. "`1 + 1`".

Operator-Syntax (2)

Assoziativitäten (2):

- x: Term mit kleinerer Priorität.
- y: Term mit gleicher oder kleinerer Priorität.
- z.B. `1+2+3` bedeutet `+(+(1,2),3)` (weil `+`: yfx).

Deklaration:

- op(Priorität, Assoziativität, Name).
  Dies muß als Anfrage ausgeführt werden, im Programm ":- op(...)".
- z.B. "op(700, xfx, ist_kind_von).
- Dann zulässig: "anke ist_kind_von christoph."
  Dies bedeutet exakt dasselbe wie "ist_kind_von(anke, christoph)".

Einschränkungen bei gemischter Syntax:

- In Standard-Syntax: kein Leerzeichen zwischen Funktor und "(".
  Ein Leerzeichen ist nötig, wenn Argument eines Präfix-Operators mit Klammer beginnt: Z.B. "`-(X) mod 2`" vs. "`- (X) mod 2`".
- Standard-Syntax: Argument-Priorität < 1000.
  Weil "," Operator mit Priorität 1000. Notfalls klammern.
- Präfixoperator als Atom ohne Argumente: (op).

Vordefinierte Operatoren (1)

Logik und Kontrolle:

Operator	Präzedenz	Assoz.	Sepia	Quintus	X-Prolog	TOY
`:-`	1200	xfx	•	•	255	•
`:-`	1200	fx	•	•	255	•
`-->`	1200	xfx	•	•	—	•
`?-`	1200	fx	•	•	—	—
`if`	1200	xfx	•	—	—	—
`delay`	1190	fx	•	—	—	—
`if`	1170	fx	•	—	—	—
`else`	1160	xfx	•	—	—	—
`then`	1150	xfx	•	—	—	—
`;`	1100	xfy	•	•	254	•
`->`	1050	xfy	•	•	—	—
`,`	1000	xfy	•	•	253	•
`~`	900	fy	•	—	—	—
`neg`	900	fy	•	—	—	—
`not`	900	fy	•	—	60, fx	•
`\+`	900	fy	•	•	—	—
`=..`	700	xfx	•	•	40	•
`~=`	700	xfx	•	—	—	—
`*`	300	xfx	•	—	—	—

Vordefinierte Operatoren (2)

Arithmetik:

Operator	Präzedenz	Assoz.	Sepia	Quintus	X-Prolog	TOY
`<`	700	xfx	•	•	40	•
`>`	700	xfx	•	•	40	•
`>=`	700	xfx	•	•	40	•
`=<`	700	xfx	•	•	40	•
`=:=`	700	xfx	•	•	40	•
`=\=`	700	xfx	•	•	40	•
`is`	700	xfx	•	•	40	•
`+`	500	yfx	•	•	31	•
`-`	500	yfx	•	•	31	•
`+`	500	fx	•	•	—	•
`-`	500	fx	•	•	31	•
`\/`	500	yfx	•	•	31	•
`/\`	500	yfx	•	•	31	•
`*`	400	yfx	•	•	21	•
`/`	400	yfx	•	•	21	•
`//`	400	yfx	•	•	—	—
`>>`	400	yfx	•	•	11, xfx	—
`<<`	400	yfx	•	•	11, xfx	—
`mod`	300	xfx	•	•	11	•
`div`	300	xfx	—	•	—	—
`^`	200	xfy	•	•	—	—
`\`	200	fx	•	•	60	—
`<=`	100	xfx	•	—	—	—
`=>`	100	xfx	•	—	—	—

Vordefinierte Operatoren (3)

Term-Vergleich:

Operator	Präzedenz	Assoz.	Sepia	Quintus	X-Prolog	TOY
`=`	700	xfx	•	•	40	•
`==`	700	xfx	•	•	40	•
`\=`	700	xfx	•	—	40	•
`\==`	700	xfx	•	•	40	•
`@<`	700	xfx	•	•	40	•
`@=<`	700	xfx	•	•	40	•
`@>`	700	xfx	•	•	40	•
`@>=`	700	xfx	•	•	40	•

Deklarationen:

Operator	Präzedenz	Assoz.	Sepia	Quintus	X-Prolog	TOY
`dynamic`	1000	fy	•	1150, fx	—	—
`export`	1000	fy	•	—	—	—
`from`	1050	xfx	•	—	—	—
`global`	1000	fy	•	—	—	—
`import`	1050	fy	•	—	—	—
`local`	1000	fy	•	—	—	—
`mode`	1000	fy	•	1150, fx	—	—
`nospy`	1000	fy	•	900, fy	250, fx	—
`spy`	1000	fy	•	900, fy	250, fx	—

Zusätzliche Operatoren von Sepia-Prolog: `help`, `listing`, `ls`, `abolish`, `skipped`, `spied`, `traceable`, `no_spy`, `unskipped`, `untraceable`.

Zusätzliche Operatoren von Quintus-Prolog: `multifile`, `volatile`, `meta_predicate` `initialization`, `public`.

Formale Syntax von Prolog (1)

Eingabe:

- Term(1200) “.”.

  Es muß ein Leerzeichen folgen, damit “.” als “full stop” erkannt wird.

- Wird als Klausel interpretiert.

  “:-” und “,” sind ja als Operatoren deklariert. Es gibt aber (Typ-) Einschränkungen, z.B. darf der Kopf keine Variable oder Zahl sein.

Term(N):

- Operator(N,fx) Term(N-1).

  Ausnahme: “-1” ist Zahlkonstante, kein zusammengesetzter Term. Außerdem: Beginnt Term(N-1) mit ”(”, so ist Leerzeichen nötig (s.o.).

- Operator(N,fy) Term(N).
- Term(N-1) Operator(N,xfx) Term(N-1).
- Term(N-1) Operator(N,xfy) Term(N).
- Term(N) Operator(N,yfx) Term(N-1).
- Term(N-1) Operator(N,xf).
- Term(N) Operator(N,yf).
- Operator(N,fx/fy).

  Präfixoperatoren, die als Atom gebraucht werden, zählen als Term ihrer Priorität (und nicht als Term der Priorität 0 wie andere Atome).

- Term(N-1).

Formale Syntax von Prolog (2)

Term(0):

- Variable.
- Zahl.
- Atom.

  Wobei das Atom nicht als Präfix-Operator deklariert ist (s.o.).
- Funktor "(" Argumente ")".

  Wobei kein Leerzeichen zwischen Funktor und "(" stehen darf.
- "[" Liste "]".
- String: "" beliebige Zeichen "".
- "(" Term(1200) ")".

Argumente:

- Term(999).
- Term(999) "," Argumente.

Liste:

- Term(999).
- Term(999) "," Liste.
- Term(999) "|" Term(999).

Turbo-Prolog / PDC-Prolog (1)

Allgemeines:

- Prädikat-Deklarationen notwendig (mit Typen).
- Compiler überprüft Argument-Typen.
- Effizienter Compiler, eingeschränkte Sprache.
  Vieles muß zur Compilezeit bekannt sein (Variablen-Positionen).
- Strings: nicht Liste, sondern eigener Datentyp.
- Schreibweise `'abc'` für Atome nicht zulässig.
- Keine Operator-Deklarationen.
- Keine Grammatiken.
- Keine Metaprogrammierung.
- Viele Prädikate heißen anders, z.B. `=` statt `is`.

Datentypen:

- `char` (`'a'`), `integer`, `real`, `string` (`"abc"`).
- `symbol` (`abc` oder `"abc"`).
  Symbole werden in eine Hashtabelle eingetragen, und dann nur durch den Index repräsentiert. Strings dagegen durch die Zeichenfolge.
- Aufzählungs-Typen.
- Variante Records.
- Listen von beliebigen Datentypen.

Turbo-Prolog / PDC-Prolog (2)

Beispiel:

- domains

```
person = anke; bernd; christoph
geoobj = quadrat(integer);
         rechteck(integer, integer)
list = reference symbol*
/* "reference" bedeutet, daß in der Datenstruktur
   ungebundene Variablen vorkommen, z.B. [a,X,c]. */
```

- predicates

```
nondeterm vater(person, person)
flaeche(geoobj, integer)
nondeterm member(symbol, list)
/* "nondeterm" bedeutet, daß das Prädikat mehrere Werte
   liefern kann, also nicht deterministisch ist */
```

- clauses

```
vater(anke, christoph).
vater(bernd, christoph).

flaeche(quadrat(X), F) :- F = X * X.
flaeche(rechteck(X,Y), F) :- F = X * Y.

member(X, [X|_]).
member(X, [_|Y]) :- member(X, Y).
```