Datentyp-Prädikate (1)

Arithmetik:

- Y `is` X : Y ist das Ergebnis $|X|$ der Auswertung des arithmetischen Ausdrucks (Terms) X.

  Der arithmetische Ausdruck kann die Operatoren $+$, $-$ (unär und binär), $*$, / und `mod` enthalten, bei größeren Prologs auch noch viele weitere. Bei Prologs ohne Float-Zahlen ist / die Integer-Division, sonst muß man // oder `div` schreiben. Viele Prologs verstehen auch `<<` und `>>` als Shift und `/\` bzw. `\/` als Bit-und/oder.

  In X dürfen keine ungebundenen Variablen vorkommen, sonst gibt es einen „instantiation fault".

- X `=:=` Y : $|X| = |Y|$.
- X `=\=` Y : $|X| \neq |Y|$.
- X `<` Y : $|X| < |Y|$.
- X `>` Y : $|X| > |Y|$.
- X `=<` Y : $|X| \leq |Y|$.
- X `>=` Y : $|X| \geq |Y|$.
- In Turbo-Prolog muß man Y `=` X statt X `is` Y schreiben. In allen normalen Prologs ist dies ein beliebter Anfänger-Fehler, weil es den Term X unausgewertet an Y zuweist (per Unifikation).
- In Sepia-Prolog kann man statt `is` auch Prädikate wie `+`(X, Y, Z) verwenden, um die einzelnen arithmetischen Funktionen zu berechnen. Es gibt z.B. auch `sin/2` und `ln/2`.

Datentyp-Prädikate (2)

Funktionen für Strings:

- `string_length`(S, L): S besteht aus L Zeichen.
- `concat_strings`(S_1, S_2, S_{12}): $S_{12} = S_1 \circ S_2$.

  Hier dürfen S_1 und S_2 keine freien Variablen sein. Es gibt aber ein Bibliotheksprädikat `append_strings/3`, das die Aufspaltung von Strings erlaubt. Schließlich gibt es noch `concat_string`(L, S), mit dem eine Liste von Strings verkettet werden kann.
- `substring`(S_1, S_2, P): S_2 kommt in S_1 vor.

  S_1 und S_2 müssen gebunden sein, es wird die Position P geliefert (von 1 an gezählt). Es gibt noch ein Bibliotheksprädikat `substring`(S_1, P, L, S_2), mit dem Teilstrings gegebener Länge und Position berechnet werden können (nur S_1 muß gebunden sein).
- Alle diese Prädikate sind Spezialitäten von Sepia-Prolog, weil dort Strings ein eigener Datentyp sind. Sepia-Prolog hat allerdings auch entsprechende Prädikate für Atome, die es aber in den meisten anderen Prologs ebenfalls nicht gibt. X-Prolog hat `subatom`(S_1, S_2).
- In Turbo-Prolog gibt es das Prädikat `concat`(S_1,S_2,S_{12}), bei dem mindestens zwei Argumente gebunden sein müssen. `frontchar`(S,C,R) zerlegt einen String in das erste Zeichen und den Rest (oder setzt ihn umgekehrt zusammen). Mit `frontstr`(N,S_{12},S_1,S_2) kann man die ersten N Zeichen abspalten. `str_len`(S,L) berechnet die Länge eines Strings. Mit `searchchar`(S,C,P) kann man die erste Position P eines Zeichens C in S bestimmen, es gibt auch `searchstring`(S_1,S_2,P). `subchar`(S,P,C) liefert das P-te Zeichen von S und `substring`(S_1, P, L, S_2) den Teilstring S_2 von S_1 ab Position P mit Länge L.

Ein-/Ausgabe (1)

Ein-/Ausgabe von Termen:

- `write`(X) : Der Term X wird ausgedruckt.

  Es werden dabei Operator-Deklarationen berücksichtigt, z.B. liefert `write(1+1)` ebenso wie `write(+(1,1))` die Ausgabe `1+1`.

- `display`(X) : Ausgabe in Präfixnotation.

  `display(1+1)` liefert also +(1,1).

- `writeq`(X) : Atome werden ggf. in `'` gesetzt.

  Dies garantiert, daß die ausgegebenen Terme mit `read` wieder eingelesen werden können.

- `nl` : Es wird eine neue Zeile begonnen.

- `read`(X) : Der Term X wird eingelesen.

  Bei der Eingabe muß der Term mit `.`⟨Neue Zeile⟩ abgeschlossen werden. Zusammen mit passenden Operator-Deklarationen ist `read` schon ein recht mächtiger Parser. Am Datei-Ende liefern die meisten Prologs `X = end_of_file`.

- Bei Turbo-Prolog dürfen die Argumente von `write` keine Variablen enthalten (dafür sind beliebig viele Argumente erlaubt). Die Prädikate `display` und `writeq` gibt es nicht.

  Zum Einlesen hat Turbo-Prolog spezielle Prädikate für jeden Datentyp, nämlich `readchar/1`, `readint/1`, `readln/1`, `readreal/1`. Das Einlesen von allgemeinen Termen geht mit `readterm(Typ, Var)`, aber die Eingabe darf keine Variablen enthalten.

  Am Dateiende (oder wenn der Benutzer `ESC` eingibt oder eine falsche Eingabe) scheitern diese Prädikate.

Ein-/Ausgabe (2)

Ein-/Ausgabe von Zeichen:

- `put`(C) : Zeichen mit ASCII-Code C ausgeben.

	0	1	2	3	4	5	6	7	8	9
0	NULL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	␣	!	"	#	$	%	&	’
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	DEL	□	□

- `get0`(C) : Das nächste Zeichen einlesen.
  Am Dateiende liefern die meisten Prologs `-1`, X-Prolog `end_of_file`.
- `get`(C) : Vorher Leerzeichen überspringen.
  SB-Prolog scheitert am Dateiende (aber `get0` liefert `-1`).
- `skip`(C) : Eingabe bis C (inkl.) überspringen.
- `tab`(N) : N Leerzeichen ausgeben.

Ein-/Ausgabe (3)

Ein-/Ausgabe von Zeichen, Forts.:

- In Sepia-Prolog muß man `get` statt `get0` schreiben. Ein Prädikat, daß wie `get` wirkt, gibt es nicht. Auch `skip` und `tab` gibt es nicht.
- Bei Sepia-Prolog gibt es noch `put_char` und `read_char`, die statt mit ASCII-Codes mit Strings der Länge 1 arbeiten. Außerdem gibt es `read_string` zum Einlesen eines Strings bis zu einem Trennzeichen und/oder einer Maximallänge.
- TOY-Prolog arbeitet statt mit ASCII-Codes mit Atomen aus einem Zeichen (Strings sind entsprechend Listen von solchen Atomen). Die Umwandlung ist mit dem Prädikat `ordchr/2` möglich.

  Die Ausgabe solcher Zeichen geschieht mit `wch`(X). Zur Eingabe gibt es einen Puffer `lastch`(X), der mit `rch` auf das nächste Zeichen gesetzt wird.
- Bei Turbo-Prolog gibt es einen eigenen Datentyp für Zeichen (die Konstanten werden in der Form `'a'` notiert). Die Umwandlung zwischen Zeichen und ASCII-Codes ist mit dem Prädikat `char_int/2` möglich. Zum Einlesen von Zeichen dient das Prädikat `readchar/1` (schlägt am Dateiende fehl). Das Schreiben von Zeichen geschieht mit `write`.

Ein-/Ausgabe (4)

Dateiverarbeitung, klassische Lösung:

- `tell`(X) : Ausgabe auf Datei X umlenken.

  X muß dabei ein Prolog-Atom sein, z.B. `'abc.dat'`. Man kann auch mehrere Dateien gleichzeitig eröffnet haben, d.h. man muß eine Datei nicht notwendigerweise mit `told` schließen, bevor man mit `tell` auf eine andere Datei wechselt. Die Standard-Ausgabe heißt `user`.

- `telling`(X) : Liefert aktuelle Ausgabedatei.
- `told` : Schließen der Ausgabedatei.
- `see`(X) : Eingabe auf Datei X umlenken.
- `seeing`(X) : Liefert aktuelle Eingabedatei.
- `seen` : Schließen der Eingabedatei.
- In Sepia-Prolog und Turbo-Prolog gibt es diese Prädikate nicht.
- In Sepia-Prolog eröffnet man einen Eingabestrom S (s.u.) und ruft dann `set_stream(input,` S`)` auf. Mit `set_stream(input, stdin)` kann man dies wieder rückgängig machen. Mit `get_stream(input,` S`)` erhält man den aktuellen Eingabestrom. (Ausgabe entsprechend)
- In Turbo-Prolog kann man sich Datei-Variablen als Konstanten des Typs `file` deklarieren. Mit `openread(file, string)` kann man eine Datei zum Lesen eröffnen, mit `openwrite(file, string)` zum Schreiben; mit `closefile(file)` wieder schließen. Die aktuelle Eingabedatei kann man mit `readdevice(file)` setzen, die Ausgabedatei mit `writedevice(file)`. Die Standardwerte sind `keyboard` und `screen`.

Ein-/Ausgabe (5)

Dateiverarbeitung, moderne Lösung:

- Es gibt einen neuen Datentyp „Stream“.
  Ein-/Ausgabestrom, d.h. offene Datei.
- Alle Prädikate zur Ein-/Ausgabe gibt es auch in einer Version mit einem zusätzlichen Argument vom Typ „Stream“.
  Dies steht immer an erster Stelle, also z.B. `put(`S`, `C`)`.
- `open`(F,M,S): Eröffne Datei F im Modus M als E/A-Strom S.
  Erlaubte Modi sind bei Sepia-Prolog `read`, `write`, `append`, `string`, `string(`N`)`.
- `current_stream`(F,M,S): Es gibt einen offenen E/A-Strom S im Modus M zur Datei F.
- `close`(S): Schließe den E/A-Strom S.
  Wenn man das bei Ausgabeströmen vergißt, wird der Puffer nicht geleert und es kommt zu Datenverlusten.
- `at_eof`(S): Stream S ist am Dateiende.
  Bei Quintus-Prolog heißt dieses Prädikat `at_end_of_file`.
- X-Prolog und viele weitere Prologs kennen keine Streams.

Prädikate zur Ablaufsteuerung

Kontrolle:

- P `,` Q : X und Y.
- P `;` Q : P oder Q.

  Bei manchen Prologs wirkt `!` durch `;` hindurch, bei anderen nicht.
- `true` : wahr (gelingt immer).
- `fail` : falsch (erzwingt Backtracking).

  Viele Prologs verstehen `false` als Synonym.
- `!` : Ignoriere Alternativen (s.o.).
- `not` P : P ist nicht beweisbar.

  Bei Quintus-Prolog muß man `\+` schreiben, bei Turbo-Prolog `not(P)`.

  ```
  not(P) :- call(P), !, fail.
  not(P).
  ```
- `once`(P) : Nur erste Lösung von P berechnen.

  `once/1` gibt es nicht in allen Prologs. `once(P) :- call(P), !.`
- P `->` Q `;` R: Wenn P, dann Q, sonst R.

  Nicht alle Prologs haben dieses Prädikat.
- P `->` Q : Wenn P, dann Q, sonst `fail`.

  Auch dies gibt es nicht in allen Prologs. Es entspricht P`, !,` Q.
- repeat : Gelingt beliebig häufig.

  ```
  repeat.   repeat :- repeat.
  ```

Prädikate zur Metaprogrammierung (1)

Term-Klassifikation (Typ-Bestimmung):

- `atom`(T) : T ist ein Prolog-Atom.
- `atomic`(T) : T ist ein Atom oder eine Zahl.

  In Sepia-Prolog zählen auch Strings als `atomic`.
- `compound`(T): T hat die Form $f(t_1, \ldots, t_n)$.

  In X-Prolog heißt es `structure`, gilt aber auch für Atome.
- `integer`(T) : T ist eine ganze Zahl.
- `nonvar`(T) : T ist keine freie Variable.
- `number`(T) : Es gilt `integer`(T) oder `real`(T).

  Dies gibt es nur bei Prologs, die beide Arten von Zahlen haben.
- `real`(T) : T ist eine Gleitkomma-Zahl.

  In Quintus-Prolog heißt dieses Prädikat `float`.
- `string`(T) : T ist ein String.

  Dies gibt es nur in Sepia-Prolog, wo Strings Konstanten sind.
- `var`(T) : T ist eine freie Variable.
- In TOY-Prolog gibt es nur `atom`, `integer`, `var` und `nonvarint`, was für einen Term steht, der weder eine freie Variable noch eine ganze Zahl ist.
- Die meisten dieser Prädikate gibt es in Turbo-Prolog nicht, sie sind wegen der statischen Typisierung auch überflüssig. Anstelle von `var` muß man `free` schreiben und `nonvar` heißt `bound`.

Prädikate zur Metaprogrammierung (2)

Term-Manipulation (Typ-Umwandlung):

- `functor`(T, F, N): T hat den Funktor F/N.

  Hiermit kann man entweder den Funktor aus einem Term extrahieren oder einen Term mit dem Funktor erzeugen (und anonymen Variablen als Argumenten).

- `arg`(N, T, A): A ist das N-te Argument von T.
- T `=..` L: L ist die Liste bestehend aus Funktor und Argumenten von T.

  Auch dieses Prädikat kann in beiden Richtungen verwendet werden.

- `name`(A, L): A ist ein Atom bestehend aus den Buchstaben der Liste L.

  L enthält ASCII-Codes. Wieder funktioniert es in beiden Richtungen.

  In TOY-Prolog heißt es `pname` und liefert Atome statt ASCII-Codes.

- Sepia-Prolog hat folgende Umwandlungsfunktionen für Strings:
  `atom_string`(A, S): A ist ein Atom mit Namen S.
  `char_int`(C, I): I ist ASCII-Code des Zeichens in S.
  `string_list`(S, L): S besteht aus Zeichen mit Codes L.
  `number_string`(N, S): S ist Dezimalrepräsentation von N.
- In Turbo-Prolog sind Prädikate wie `functor`, `arg` und `=..` aufgrund der Typisierung nicht möglich. Es gibt folgende Prädikate zur Typ-Umwandlung: `char_int(char, integer)`, `str_char(string, char)`, `str_int(string, integer)` und `str_real(string, real)`.

Prädikate zur Metaprogrammierung (3)

Term-Vergleich:

- T_1 `=` T_2 : T_1 und T_2 sind unifizierbar.
- T_1 `==` T_2 : T_1 und T_2 sind textuell identisch.

  Zum Beispiel gilt X `=` a, aber nicht X `==` a.
- T_1 `\=` T_2 : T_1 und T_2 sind nicht unifizierbar.

  Gibt es nicht bei Quintus-Prolog (nur als Bibliotheksprädikat).
- T_1 `\==` T_2: T_1 und T_2 sind nicht identisch.
- T_1 `@<` T_2 : T_1 kommt vor T_2.

  Die Ordnung ist folgendermaßen definiert: Erst Variablen (in undefinierter Reihenfolge), dann Gleitkomma-Zahlen (der Größe nach), dann ganze Zahlen, dann Strings (lexikographisch), dann Atome und schließlich zusammengesetzte Terme (zuerst nach Stelligkeit, dann nach dem Funktor-Namen, zuletzt nach den Argumenten von links nach rechts).
- T_1 `@>` T_2 : T_1 kommt nach T_2.
- T_1 `@=<` T_2: T_1 kommt vor T_2 oder ist identisch.
- T_1 `@>=` T_2: T_1 kommt nach T_2 oder ist identisch.
- `compare`(R, T_1, T_2): R ist $<$, $=$ oder $>$ entsprechend T_1 `@<` T_2, T_1 `==` T_2 und T_1 `@>` T_2.
- TOY-Prolog: `@<` usw. nur für Atome, kein `compare`.
- Bei Turbo-Prolog wertet `=` arithmetische Ausdrücke aus, führt aber ansonsten die Unifikation durch. Die anderen Prädikate gibt es alle nicht. Allerdings kann man Strings lexikographisch mit `<` etc. vergleichen.

Prädikate zur Metaprogrammierung (4)

Zur Laufzeit konstruierte Anfragen:

- `call`(P) : Führe Anfrage P aus.

  Gibt es nicht bei Turbo-Prolog, aber bei allen anderen Prologs.

- Bei vielen Prologs kann man auch einfach `X` statt `call(X)` schreiben.

Alle Lösungen:

- `findall`(X, P, L): L ist Liste der X mit $P(X)$.

  Z.B. gegeben: p(a). p(b). `findall(X, p(X), Y)` liefert `Y=[a,b]`.

- `bagof`(X, P, L): (ähnlich `findall`).

  Der Unterschied besteht in der Behandlung von Variablen, die in P vorkommen, aber nicht in X. Bei `findall` sind sie existenzquantifiziert, d.h. sie werden nicht gebunden, und `findall` kann nur ein einziges mal erfüllt werden. Bei `bagof` werden diese Variablen dagegen ggf. an einen Wert gebunden, und es werden nur Lösungen mit diesem Wert gefunden. Beim Backtracken erhält man dann auch andere Lösungen.

- `setof`(X, P, L): L ist Menge der X mit $P(X)$.

  Die "Menge" ist dabei eine geordnete Liste ohne Duplikate.

- TOY-Prolog nur `bagof`, X-Prolog hat keines dieser Prädikate. Allerdings kann man sich `findall` mit `call`, `fail`, `assert` und `retract` (s.u.) leicht selbst definieren (Übungsaufgabe).

- Turbo-Prolog hat nur `findall` mit der zusätzlichen Einschränkung, daß X eine Variable sein muß (normalerweise ist hier auch ein Term mit mehreren Variablen möglich). Eine „domain" für die Liste L muß deklariert sein.

Datenbank-Prädikate (1)

Modifikation der Datenbasis:

- **`dynamic`** P/N: P/N ist zur Laufzeit änderbar.

  Dieses Prädikat gibt es nur bei Prolog-Compilern, bei Interpretern sind alle Prädikate dynamisch. Bei Turbo-Prolog muß man stattdessen das Prädikat in dem `database`-Abschnitt deklarieren.

- **`assert`**(C): Klausel C einfügen.

  Die Position relativ zu den schon existierenden Klauseln ist hier undefiniert. `asserta`(C) fügt am Anfang ein, `assertz`(C) am Ende. Sepia-Prolog hat kein `assertz`, dafür fügt `assert` dort immer am Ende ein. Bei Turbo-Prolog können nur Fakten ohne Variablen in der Datenbank abgelegt werden, was eine starke Einschränkung ist.

- **`retract`**(C): Klausel C löschen.

  Es wird die erste auf C passende Klausel gelöscht. Bei den meisten Prologs kann man durch Backtracing weitere Klauseln löschen.

- Es ist von Prolog zu Prolog verschieden, inwieweit `assert` und `retract` noch auf schon begonnene Beweise wirken (bei moderneren Prologs tuen sie es meist nicht).

  ```
  test :- assertz((test :- write(hallo))), fail.
  test :- fail.
  ```

  Ohne die zweite Klausel finden nur wenige Prologs die neue Klausel.

- **`retractall`**(H): Klauseln mit Kopf H löschen.

  Genauer gesagt werden alle Klauseln gelöscht, deren Kopf mit H unifizierbar ist. In Sepia-Prolog muß man `retract_all` schreiben.

- **`abolish`**(P, N): Definition von P/N löschen.

Datenbank-Prädikate (2)

Einlesen und Ausgeben der Datenbasis:

- `consult`(F): Klauseln aus Datei F einlesen.

  F muß der Dateiname als Atom sein. Bei Sepia-Prolog heißt es `compile` und erlaubt auch Strings als Dateinamen. Ausschlaggebend für die Kompilierung/Interpretation sind immer die `dynamic`-Deklarationen. Bei Turbo-Prolog ist der Dateiname ein String und die Datei muß vorher mit den inversen Kommando `save`(F) erstellt sein.

- `reconsult`(F): Vorher alte Klauseln löschen.

  Es werden Klauseln der Prädikate p gelöscht, für die F eine Regel der Form $p(\ldots) \leftarrow \ldots$ enthält.

- `clause`(H, B): Es gibt eine Klausel $H \leftarrow B$.

  Fakten haben den Rumpf `true`. `clause` gibt es nicht in Turbo-Prolog.

- `listing`(P): Definition von P ausgeben.

  Nicht alle Prologs haben dieses Prädikat. In Sepia-Prolog kann man es zu `ls` abkürzen. Natürlich muß P ein dynamisches Prädikat sein.

- `listing`: dasselbe für alle dyn. Prädikate.

Interne Datenbank:

- Bei größeren Prologs gibt es daneben noch eine „internal indexed database“, in die man Terme unter einem Schlüssel abspeichern kann. Man erhält dann eine „Datenbank-Referenz“. Bei vielen Prologs gibt es aber ohnehin einen Index über dem ersten Argument eines Prädikats.
- Das Abspeichern geschieht mit `record`, das Suchen mit `recorded` und das Löschen mit `erase` (siehe Handbuch).

Sonstiges

Prolog-Umgebung:

- `halt`: Prolog-System verlassen.
- `abort`: Prolog-Programm beenden.

  Man kehrt dann zum Haupt-Prompt des Prolog-Systems zurück. Nicht alle Prologs haben dieses Prädikat.
- `help`(P/N): Anleitung zu P/N anzeigen.

  Falls P ein Operator ist, sollte man ihn in Klammern einschließen. Nur große Prolog-Systeme haben so ein „Online-Manual". Bei Quintus-Prolog rufe man `manual` auf.
- system(C): Betriebssystemkommando aufrufen.

  Nicht alle Prologs haben dieses Prädikat. Bei Quintus-Prolog muß man `unix(sh(`C`))` aufrufen.
- `statistics`: Speicherbelegung etc. anzeigen.

  Nicht alle Prologs haben dieses Prädikat.
- `cputime`(T): Verbrauchte Rechenzeit (in sec).

  Zeitmessungen funktionieren bei jedem Prolog-System anders. Bei Quintus-Prolog muß man `statistics(runtime, [`T`|_])` aufrufen, bei X-Prolog T is cputime (und erhält bei beiden Systemen msec). Bei Turbo-Prolog gibt es `time`(S, M, SEK, H), was die aktuelle Uhrzeit in Stunden, Minuten, Sekunden und Hundertstel liefert.
- `trace`: Debugger anschalten.
- `notrace`: Debugger ausschalten.
- `spy`(P): „Breakpoint" auf Prädikat P setzen.

Dokumentation von Prolog-Prädikaten

Bedeutung des Prädikates:

- Zweck/Funktion des Prädikates („synopsis").

  Wofür steht der abgekürzte Prädikatname?

  Aussagekräftige Argument-Namen sind auch nützlich.

- Gründe für Wahrheitswert falsch („fails").

  Eigentlich müßte man angeben, welche mathematische Relation durch das Prädikat beschrieben wird.

- Verhalten beim Backtracking („resatisfiable?").

  Gibt es mehrere Lösungen?

- Seiteneffekte.

  Ein-/Ausgaben, Änderungen an Datenbasis und Interpreter-Verhalten.

Gründe für Fehlermeldungen (exceptions):

- Typ-Einschränkungen für die Argumente.

  Z.B. es muß eine Zahl sein, oder ein aufrufbarer Term, etc.

- Welche Argumente müssen gebunden/frei sein?

  Häufig werden Argumente, die keine Variablen sein dürfen, mit „+" gekennzeichnet. Uninstanziierte Variablen entsprechend mit „−".

Sonstiges:

- Beispiele.
- Verwandte Prädikate („see also").

Aufgaben (1)

Arithmetik:

Definieren Sie ein Prädikat zur Berechnung der Fibonacci-Funktion f:

$$f(n) := \begin{cases} 1 & n = 0,\, n = 1 \\ f(n-1) + f(n-2) & n \geq 2 \end{cases}$$

Arithmetik, Term-Klassifikation:

Definieren Sie ein Prädikat `sum`(X, Y, Z), das genau dann gilt, wenn $X + Y = Z$. Im Unterschied zu „Z `is` $X+Y$“ soll es ausreichen, daß zwei beliebige Argumente gebunden sind.

Arithmetik, Ein-/Ausgabe:

Schreiben Sie einen „Taschenrechner“: Bis zur Eingabe `ende` wird jeweils ein arithmetischer Ausdruck gelesen und der entsprechende Wert ausgegeben.

Datei-Verarbeitung:

Definieren Sie ein Prädikat `lines`(D, L), das zu einer Datei D die Anzahl Zeilen L liefert.

Aufgaben (2)

Term-Klassifikation:

Definieren Sie ein Prädikat `makeground`(T), das alle in einem Term T auftretenden Variablen durch `x` ersetzt.

Modifikation der Datenbasis:

Definieren Sie ein Prädikat `next`(X), das bei jedem Aufruf eine neue natürliche Zahl liefert (0,1,2,...).

Meta-Programmierung, Datenbank:

Definieren Sie sich `findall` selbst.

Zur Vereinfachung können Sie davon ausgehen, daß die Anfrage von `findall` nicht selber `findall` aufruft.

Prolog-Umgebung:

Definieren Sie ein Prädikat `ediere`(D), das den Editor für „D.pl“ aufruft und diese Datei anschließend mit `consult` neu lädt.