4. Eingebaute Prädikate

1. Beispiele, Motivation
2. Allgemeines zu eingebauten Prädikaten, Syntax
3. Bindungsmuster
4. Weitere Beispiele
5. Bereichsbeschränkung
6. Funktionssymbole

Beispiele, Motivation

Datentyp-Operationen:

- Bisher haben wir zwar Konstanten für Zahlen, Strings, Datumsangaben etc. in Datenbank und Programmen, aber wir können mit ihnen nicht „rechnen".
- Ein Datentyp besteht aus einer Menge und Operationen darauf (Operation mit booleschem Ergebnis: Prädikat).
- Operationen für Zahlen: `+`, `-`, `*`, `/`, `mod`, `div`, `log`, `sin`, ...
- Prädikate für Zahlen: <, =<, >, >=
- Prädikate für beliebige Datentypen: =, \=
- SQL hat derartige Datentyp-Operationen.

Eingebaute Prädikate (1)

Allgemeines:

- Prädikate wie <, >, \= haben unendliche Extensionen.
- Sie müssen daher über eine Prozedur im deduktiven DBS definiert werden, und nicht durch eine Aufzählung von Fakten in einer DB-Relation (deswegen „eingebaute Prädikate").
- Zur Definition der Semantik können wir uns natürlich ein unendliches Programm vorstellen, in dem z.B. folgende Faktenmenge enthalten ist:

$$\{\mathsf{<}(n, m) \mid n, m \in \mathbb{Z},\ n < m\}.$$

Eingebaute Prädikate (2)

Funktionen vs. Prädikate:

- Funktionen wie + können auch als Prädikat gesehen werden:

$$+(X, Y, Z) \;:\Longleftrightarrow\; X + Y = Z.$$

- Tatsächlich werden Funktionen mathematisch üblicherweise als Relationen definiert, mit der zusätzlichen Eigenschaft, daß das letzte Argument durch die anderen eindeutig definiert ist (auch: „Graph der Funktion").

- In Prolog übliche Schreibweise: `Z is X + Y`.
  Beachte: `Z = X + Y` ist dort ein Fehler, es würde den strukturierten Term `X + Y` mit `Z` unifizieren.
  Manche deduktiven DBS verstehen `Z = X + Y` aber „richtig".

Eingebaute Prädikate (3)

Funktionale Notation:

- Man kann `Z is X + Y` (bzw. `Z = X + Y`) auch einfach als andere Notation für ein Literal auffassen. Die Notation $\text{p}(\text{t}_1, \ldots, \text{t}_n)$ ist nicht unbedingt die einzige Möglichkeit.
- Z.B. wäre „`Y ist Vater von X`“ oder „`Y = vater(X)`“ eventuell auch übersichtlicher als $\text{vater}(X, Y)$.
- Allerdings kann man Funktionen auch schachteln, z.B. `X is (Y + 3) * Z`.
- Dies kann man durch Einführung von Variablen für Zwischenergebnisse auflösen: `V is Y + 3, X is V * Z`. Also nur „syntaktischer Zucker“, kann intern normalisiert werden.

Bindungsmuster (1)

- Die folgende Anfrage sollte ausgeschlossen werden:

$$\text{answer}(X, Y) \leftarrow X < Y.$$

  Unendliches Ergebnis, schlimmer noch: Nicht einmal eine Iteration des T_P-Operators kann wirklich ausgeführt werden.

- Das Prädikat < kann nur als Test verwendet werden: Beide Argumente müssen „gebunden“ sein, d.h. Konstanten oder Variablen, die schon mit einem Wert belegt sind.

- Z.B. sollte zulässig sein:

$$\text{answer}(\text{Titel}) \leftarrow \text{vorlesung}(\text{Dozent, Titel, Std}),\ \text{Std} < 4.$$

Bindungsmuster (2)

- Bei „`Z is X + Y`" müssen in Prolog `X` und `Y` schon an einen Wert gebunden sein, während `Z` „frei" sein kann.
  Sonst: „Instantiation fault".
- `Z` muß aber nicht frei sein, dann wird die Berechnung zu einem Test. Z.B. schlägt das Beweisziel „`5 is 3 + 1`" fehl.
  Auch syntaktischer Zucker: „`V is 3 + 1, 5 = V`".
- Tatsächlich wäre „`Z is X + Y`" auswertbar, wenn nur mindestens zwei der drei Variablen gebunden sind.
  Z.B. „`5 is 3 + Y`" entspricht „`Y is 5 - 3`".
- Man verwechsele „frei" und „gebunden" nicht mit der Bindung durch einen Quantor in der Logik erster Stufe. Hier: Bindung an einen Wert durch vorherige Auswertung eines anderen Literals.

Bindungsmuster (3)

- Bei DB-Relationen sind Ein- und Ausgabe-Argumente nicht wichtig, z.B. sind alle folgenden Anfragen möglich:

$\mathrm{vater}(\mathrm{julia}, \mathrm{arno})$.	$\mathrm{vater}(X, \mathrm{arno})$.
$\mathrm{vater}(\mathrm{julia}, X)$.	$\mathrm{vater}(X, Y)$.

- Bei IDB-Prädikaten bisher auch nicht.
  Es ist ein Vorteil von Prolog/Datalog, daß man mit einer Funktion auch gleich die Umkehrfunktion mitdefiniert.
- Eingebaute Prädikate können dagegen nur für bestimmte „Bindungsmuster" von Ein- und Ausgabe-Argumenten ausgewertet werden.

Bindungsmuster (4)

Definitionen:

- Ein Bindungsmuster (Modus) für ein n-stelliges Prädikat ist ein String der Länge n über dem Alphabet $\{\mathtt{b}, \mathtt{f}\}$.
  D.h. formal eine Abbildung $\{1, \ldots, n\} \rightarrow \{\mathtt{b}, \mathtt{f}\}$.
- Z.B. Bindungsmuster für vater: `bb`, `bf`, `fb` und `ff`.
- Gegeben ein Bindungsmuster $\beta_1 \ldots \beta_n$ für ein Prädikat p, so sagt man
  - falls $\beta_i = \mathtt{f}$: das i-te Argument von p ist frei (oder: es ist ein Ausgabe-Argument),
  - falls $\beta_i = \mathtt{b}$: das i-te Argument von p ist gebunden (oder: es ist ein Eingabe-Argument).

Bindungsmuster (5)

Mögliche Bindungsmuster:

- Sei $\mathcal{I}_{\mathsf{pred}}(\mathsf{p})$ die beabsichtigte Interpretation von p.
  Ein Bindungsmuster β für p heißt möglich gdw. folgende Bedingung erfüllt ist:

  Sei $\{i_1, \ldots, i_k\} := \{i \mid 1 \leq i \leq n,\ \beta_i = \mathsf{b}\}$.
  Dann ist für alle $\mathsf{c}_1, \ldots, \mathsf{c}_k \in \mathcal{I}_{\mathsf{dom}}$ die Menge

  $$\big\{(\mathsf{d}_1, \ldots, \mathsf{d}_n) \in \mathcal{I}_{\mathsf{pred}}(\mathsf{p}) \;\big|\; \mathsf{d}_{i_1} = \mathsf{c}_1, \ldots, \mathsf{d}_{i_k} = \mathsf{c}_k\big\}$$

  endlich. Außerdem gibt es einen Algorithmus, der für alle $\mathsf{c}_1, \ldots, \mathsf{c}_k$ diese Menge effektiv berechnet.

- Beispiel: bb ist das einzige mögliche Bindungsmuster für <.

Bindungsmuster (6)

Allgemeineres / Spezielleres Bindungsmuster:

- Ein Bindungsmuster β der Stelligkeit n heißt allgemeiner als ein Bindungsmuster β' der gleichen Stelligkeit gdw.

  $$\text{für } i = 1, \ldots, n\text{:} \quad \beta_i = \texttt{b} \implies \beta'_i = \texttt{b}.$$

  Umgekehrt heißt β' dann spezieller als β.

- Beispiel: `bff` ist allgemeiner als `bfb`.

Bindungsmuster (7)

Bemerkungen:

- Für jedes unterstütze Bindungsmuster β eines Prädikates p muß der Entwickler des DDBS im allgemeinen eine eigene Prozedur p_β schreiben.
- Ausgenommen speziellere Bindungsmuster, die kann das System automatisch über Einführung von Variablen und einen Gleichheitstest (s.o.) implementieren. Allerdings ist das häufig nicht so effizient wie eine eigene Prozedur.
- Ein System muß nicht alle möglichen Bindungsmuster wirklich unterstützen.

Bindungsmuster (8)

Aufgaben:

- Gegeben sei das Prädikat `square`$(X, Y) \iff Y = X^2$. Ist das Bindungsmuster `fb` möglich? Welche Bedeutung hat es? Wie viele Lösungen gibt es für einen gegebenen Wert von Y?
- Welche Bindungsmuster sind möglich/sinnvoll für:

```
date(Tag, Monat, Jahr, Datum).
```

- Welche Bindungsmuster von

```
substring(TeilStr, VonPos, BisPos, String)
```

würden Sie als Systementwerfer unterstützen?

Bindungsmuster (9)

Bemerkung zu Indexen:

- Im Prinzip können auch EDB-Prädikate unterschiedliche Implementierungen für verschiedene Bindungsmuster haben — wenn nämlich mit einem Index (Hashverfahren, B-Baum) auf die Relation zugegriffen wird.
- Wenn es z.B. bei $\mathrm{eltern}(X, Y, Z)$ einen Index über dem ersten Argument gibt, dann gibt es Implementierungen für
  - `bff`: Zugriff auf die Relation eltern mit Index.
  - `fff`: „Full Table Scan" (kompletter Durchlauf).

  Alle anderen Bindungsmuster werden durch zusätzliche Gleichheitsabfragen hieraus gebildet.

Bindungsmuster (10)

Bemerkung zu Indexen, Forts.:

- Ein Index ist aber allgemeiner als ein Bindungsmuster, z.B. kann ein B-Baum auch die folgende Art von Anfragen implementieren:

$$\mathsf{p}(\mathsf{a}, \mathsf{Y}, \mathsf{Z}),\ \mathsf{Y} < 10.$$

- Ein Index ist also eine spezielle Implementierung für eine parametrisierte Anfrage (meist Konjunktion).

Bindungsmuster (11)

Weitere Angaben zu Bindungsmustern:

- Es wäre nützlich zu Bindungsmustern zusätzlich anzugeben, ob es immer maximal ein Ergebnistupel gibt ($\mathfrak{b}$-Positionen sind Schlüssel).
- Auch die Information, daß es immer minimal ein / genau ein Ergebnistupel gibt (wie z.B. bei $+/\mathfrak{bbf}$) kann für die Übersetzung von Regeln nützlich sein.
- Auch eine Kostenschätzung (Laufzeit, Ergebnisgröße) für jedes unterschiedlich implementierte Bindungsmuster könnte vom Optimierer genutzt werden.

Weitere Beispiele (1)

WWW-Interface:

- `wget(URL, Text, Length, Date, Type)`
  Holt WWW-Seite. Bindungsmuster `bffff`. Die drei letzten Argumente stammen aus dem HTTP-Header. Eventuell könnte man diese Daten auch anders liefern, falls der Benutzer sie überhaupt möchte.

- `link(Text, URL)`
  Der HTML-`Text` enthält einen Verweis auf `URL`. Bindungsmuster `bf`. Damit dies auch bei relativen Links funktioniert, muß man dem Text wohl auch die ursprüngliche URL noch beifügen.

- Aufgabe: Denken Sie sich ein besseres WWW-Interface aus, inklusive Prädikaten zum Parsen von HTML.

Weitere Beispiele (2)

Datei-Interface:

- `dir(Directory, Filename, Kind, Date,`
  `Owner, Group, AccessRights)`

  `Kind` würde unter anderem auch Subdirectories anzeigen. Dann kann man mit Datalog sehr leicht ein rekursives `find` ausführen.

- `line(File, LineNo, Text)`

- `open(Filename, FirstPtr)`
  `getc(Ptr, Character, NextPtr)`
  `token(StartPtr, Text, NextPtr)`
  `eof(Ptr)`

- Ausgaben sind dagegen schwieriger deklarativ zu beschreiben.

Weitere Beispiele (3)

Zugriff auf Datenstrukturen:

- Eine endliche Relation muß keineswegs immer als „File of Records“ abgespeichert sein.
- Z.B. könnte man ein Array `arr` auch als Prädikat auffassen:

  ```
  arr(Index, Element).
  ```

  Das Bindungsmuster `bf` entspricht hier dem Array-Zugriff.
- Auf eine verkettete Liste im Heap könnte man mittels

  ```
  listelem(Ptr, Contents, Next)
  ```

  zugreifen. Bindungsmuster `bff`.

Weitere Beispiele (4)

Erweiterbarkeit:

- Im Prinzip kann jede Seiteneffekt-freie Prozedur als Prädikat in ein deduktives DBS eingebunden werden.
- Manche deduktiven DBS bemühen sich speziell um solche Erweiterbarkeit. Besonders für deduktive DBS als „Middelware“: Integrationen von Datenbanken und Programmen. Auch bei relationalen Systemen wichtig: Hinzufügen neuer Datentypen (mit Operationen und Zugriffsstrukturen). „Universal Server“.
- Prozeduren mit Seiteneffekten schwierig, weil Optimierer keinerlei Garantien über Aufruf-Reihenfolge und Aufruf-Häufigkeit macht.
- Aufgabe: Definieren Sie ein Interface.

Bereichsbeschränkung (1)

- Da die möglichen Bindungsmuster für die Prädikate eingeschränkt sind, müssen auch die zulässigen Regeln entsprechend eingeschränkt werden.
- Zum Beispiel soll folgende Regel ausgeschlossen werden:

$$\mathsf{succ}(\mathsf{X}, \mathsf{Y}) \leftarrow \mathsf{Y}\ \texttt{is}\ \mathsf{X} + 1.$$

- Hier ist nicht einmal eine einzige Iteration des T_P-Operators möglich, man würde unendlich viele Tupel herleiten müssen. Bzw. einen „Instantiation fault“ Fehler von dem eingebauten Prädikat bekommen, weil das Bindungsmuster `fbf` für $+$ nicht implementiert ist (und nach obiger Definition auch unmöglich ist).

Bereichsbeschränkung (2)

- Diese Einschränkung gilt aber nur für die naive/semi-naive Auswertung, bei der die Extensionen der IDB-Prädikate vollständig berechnet werden.
  Hier müssen IDB-Prädikate das Bindungsmuster `ff...f` erlauben.
- Wir werden aber mit den „magischen Mengen" ein Verfahren kennenlernen, das Extensionen der IDB-Prädikate nur für gegebene Eingabewerte berechnet.
- Für das Bindungsmuster `bf` soll obige Regel dann doch erlaubt sein. Für `fb` eventuell auch. Nur nicht für `ff`.
  Fäßt man IDB-Prädikate als Prozeduren/Makros auf, so wäre es unverständlich, daß man nicht die Abkürzung succ einführen darf. Bei RDBen sich solche „Sichten" dagegen tatsächlich verboten.

Bereichsbeschränkung (3)

Voraussetzung:

- Sei für jedes Prädikat p eine Menge BIND(p) von möglichen Bindungsmustern für p gegeben:
  - Für eingebaute Prädikate: Implementierte/mögliche Bindungsmuster.
  - Für EDB-Prädikate: `ff...f`.
  - Für IDB-Prädikate: Auch `ff...f`, sofern vom Programmierer nicht eingeschränkt.

    Wie oben erläutert, kann eine Einschränkung nötig sein, um die Bereichsbeschränkung zu erreichen. Eventuell kann das System solche Einschränkungen auch automatisch berechnen.
  - Für answer: `ff...f`.

Bereichsbeschränkung (4)

Notationen:

- Sei vars(A) die Menge der im Literal A vorkommenden Variablen:

$$\mathsf{vars}\big(\mathsf{p}(\mathsf{t}_1, \ldots, \mathsf{t}_n)\big) = \big\{\mathsf{t}_i \bigm| 1 \le i \le n,\ \mathsf{t}_i \in \mathcal{VAR}\big\}.$$

- Sei input_vars(A, β) die Menge der in Eingabeargumenten von A (bezüglich β) vorkommenden Variablen:

$$\mathsf{input\_vars}\big(\mathsf{p}(\mathsf{t}_1, \ldots, \mathsf{t}_n),\ \beta\big) = \big\{\mathsf{t}_i \bigm| 1 \le i \le n,\ \mathsf{t}_i \in \mathcal{VAR},\ \beta_i = \mathtt{b}\big\}.$$

Bereichsbeschränkung (5)

Eine Regel $\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_m$ heißt bereichsbeschränkt gdw. für jedes Bindungsmuster $\beta \in \mathsf{BIND}\bigl(\mathsf{pred}(\mathsf{A})\bigr)$ folgende Bedingungen erfüllt sind:

1. Es gibt eine Permutation $\pi\colon \{1, \ldots, m\} \rightarrow \{1, \ldots, m\}$ der Rumpfliterale und Bindungsmuster $\beta_i \in \mathsf{BIND}\bigl(\mathsf{pred}(\mathsf{B}_i)\bigr)$ für $i = 1, \ldots, m$, so daß

   $$\mathsf{input\_vars}(\mathsf{B}_{\pi(i)}, \beta_{\pi(i)}) \subseteq \mathsf{input\_vars}(\mathsf{A}, \beta) \cup \bigcup_{j=1}^{i-1} \mathsf{vars}(\mathsf{B}_{\pi(j)}).$$

2. $\mathsf{vars}(\mathsf{A}) \subseteq \mathsf{vars}(\mathsf{B}_1) \cup \cdots \cup \mathsf{vars}(\mathsf{B}_m) \cup \mathsf{input\_vars}(\mathsf{A}, \beta)$.

Bereichsbeschränkung (6)

Erläuterungen:

- Ein Literal B_i ist (bzgl. β_i) auswertbar gdw. die Variablen in $\mathfrak{b}$-Positionen schon an einen Wert gebunden sind.
- Das Binden an einen Wert geschieht dadurch, daß die gleiche Variable schon in einem früher ausgewerteten Rumpfliteral aufgetreten ist, oder aber in einem Eingabeargument des Kopfes.
- Die erste Bedingung fordert, daß man die Rumpfliterale so umordnen kann, daß in der neuen Reihenfolge jedes Rumpfliteral auswertbar ist.
- Die zweite Bedingung fordert, daß man das Kopfliteral explizit konstruieren kann, nachdem man den Rumpf ausgewertet hat. Dazu dürfen dort keine ungebundenen Variablen mehr vorkommen.

Bereichsbeschränkung (7)

Aufgaben:

- Ist die folgende Regel bereichsbeschränkt?

$$\begin{array}{lcl} \mathsf{end\_preis}(\mathsf{Ware}, \mathsf{X}) & \leftarrow & \mathsf{angebot}(\mathsf{Ware}, \mathsf{Preis}) \wedge \\ & & \mathsf{X}\ \mathtt{is}\ \mathsf{Preis}\ *\ 1.16. \end{array}$$

Dabei sei:

$$\begin{array}{lcl} \mathsf{BIND}(\mathsf{end\_preis}) & = & \{\mathtt{ff}\}, \\ \mathsf{BIND}(\mathsf{angebot}) & = & \{\mathtt{ff}\}, \\ \mathsf{BIND}(*) & = & \{\mathtt{bbf}, \mathtt{bfb}, \mathtt{fbb}, \mathtt{bbb}\}. \end{array}$$

Bereichsbeschränkung (8)

Aufgaben, Forts.:

- Welche Bindungsmuster für append sind zulässig?

$$\begin{array}{lcl} \text{append}([\,],\ \text{L},\ \text{L}) & \leftarrow & \text{true.} \\ \text{append}(\text{HT},\ \text{L},\ \text{HX}) & \leftarrow & \text{cons}(\text{H},\ \text{T},\ \text{HT}) \wedge \\ & & \text{append}(\text{T},\ \text{L},\ \text{X}) \wedge \\ & & \text{cons}(\text{H},\ \text{X},\ \text{HX}). \end{array}$$

Es sei $\text{BIND}(\text{cons}) = \{\texttt{bbf},\ \texttt{ffb}\}$.
$\text{cons}(X, Y, Z) \;:\Longleftrightarrow\;$ X ist erstes Element der Liste Z, Y der Rest.

Bereichsbeschränkung (9)

- Sei ein Programm P gegeben, dessen Regeln bereichsbeschränkt sind, wobei jedem IDB-Prädikat das Bindungsmuster `ff...f` zugeordnet ist.
- Dann liefert jede Anwendung des T_P-Operators nur endlich viele Fakten, die auch effektiv berechenbar sind.
- Man kann die T_P-Iteration also ausführen und bekommt „bessere und bessere Approximationen an das minimale Modell".
- Die Iteration muß allerdings nicht unbedingt terminieren:

$$\mathsf{nat}(0).$$

$$\mathsf{nat}(\mathsf{Y}) \leftarrow \mathsf{nat}(\mathsf{X}) \land \mathsf{Y}\ \texttt{is}\ \mathsf{X} + 1.$$

Funktionssymbole (1)

- Prolog und alle neueren deduktiven DBS erlauben allgemeinere Terme, als wir in Kapitel 3 eingeführt haben.
- Ein Term ist nun:
  - eine Konstante,
  - eine Variable, oder
  - ein strukturierter Term $\mathrm{f}(\mathrm{t}_1, \ldots, \mathrm{t}_n)$ bestehend aus einem n-stelligen Funktionssymbol f und Termen $\mathrm{t}_1, \ldots, \mathrm{t}_n$. $\mathrm{t}_1, \ldots, \mathrm{t}_n$ können selbst strukturiert sein (rekursive Definition).
- D.h. Terme sind Baumstrukturen mit Funktionssymbolen in den inneren Knoten und Variablen oder Konstanten in den Blättern.

Funktionssymbole (2)

Beispiele:

- $+(3, 5)$, oder in Infix-Schreibweise: $3 + 5$.
- date$(19,\ 6,\ 98)$.
- .$\big($a, .(b, [ ])$\big)$, oder in Prolog übliche Notation: [a, b].

Bemerkung:

- In Prolog können beliebige Atome als Funktionssymbole verwendet werden (mit beliebiger Stelligkeit).
  Das ist praktisch, weil es dann keinen formalen Unterschied zwischen Literalen und Termen gibt, ja auch ganze Regeln als Term aufgefaßt werden können (für „Meta-Programmierung").

Funktionssymbole (3)

Interpretation:

- Eine allgemeine Interpretation $\mathcal{I}$ interpretiert jedes n-stellige Funktionssymbol f durch eine Funktion $\mathcal{I}_{\mathsf{fun}}(\mathsf{f})\colon \mathcal{I}_{\mathsf{dom}}^n \to \mathcal{I}_{\mathsf{dom}}$.
- Z.B. $\bigl(\mathcal{I}_{\mathsf{fun}}(+)\bigr)(3,5) = 8$.
- Herbrand-Interpretationen verwenden dagegen die „freie" Interpretation von Funktionssymbolen als Termkonstruktoren.
- Das Herbrand-Universum ist jetzt die Menge aller variablenfreien Terme.
- $\bigl(\mathcal{I}_{\mathsf{fun}}(+)\bigr)(3,5)$ liefert dann den Term $3+5$.
  Das ist allgemein genug, weil man die Interpretation bei Bedarf noch in den Prädikaten vornehmen kann.

Funktionssymbole (4)

Modell-Begriff:

- Bei der Definition von Modell kommt nur in der Auswertung von Termen der Fall für strukturierte Terme hinzu:
  - $\mathsf{eval}(\mathcal{I}, \alpha, \mathsf{t}) := \big(\mathcal{I}_{\mathsf{fun}}(\mathsf{f})\big)\big(\mathsf{eval}(\mathcal{I}, \alpha, \mathsf{t}_1), \ldots, \mathsf{eval}(\mathcal{I}, \alpha, \mathsf{t}_n)\big)$, falls t ein strukturierter Term $\mathsf{f}(\mathsf{t}_1, \ldots, \mathsf{t}_n)$ ist.

Anwendung von Substitutionen:

- Entsprechend steigt auch eine Substition rekursiv im Baum des Termes ab, um rekursiv alle vorkommenden Variablen zu ersetzen:
  - $\theta\big(\mathsf{f}(\mathsf{t}_1, \ldots, \mathsf{t}_n)\big) := \mathsf{f}\big(\theta(\mathsf{t}_1), \ldots, \theta(\mathsf{t}_n)\big)$.

Funktionssymbole (5)

Vorkommende Variablen:

- Auch die Berechnung der in einem Term vorkommenden Variablen muß jetzt rekursiv im Baum absteigen:
  - $\mathsf{vars}(\mathsf{c}) := \emptyset$.
  - $\mathsf{vars}(\mathsf{X}) := \{\mathsf{X}\}$.
  - $\mathsf{vars}\big(\mathsf{f}(\mathsf{t}_1, \ldots, \mathsf{t}_n)\big) := \mathsf{vars}(\mathsf{t}_1) \cup \cdots \cup \mathsf{vars}(\mathsf{t}_n)$.
  - $\mathsf{vars}\big(\mathsf{p}(\mathsf{t}_1, \ldots, \mathsf{t}_n)\big) := \mathsf{vars}(\mathsf{t}_1) \cup \cdots \cup \mathsf{vars}(\mathsf{t}_n)$.
- $\mathsf{input\_vars}\big(\mathsf{p}(\mathsf{t}_1, \ldots, \mathsf{t}_n),\ \beta\big) := \bigcup_{i\in\{1,\ldots,n\},\ \beta_i=\mathtt{b}} \mathsf{vars}(\mathsf{t}_i)$.

Funktionssymbole (6)

Bemerkung:

- Alle anderen Definitionen bleiben textuell unverändert.
  Sie ändern sich natürlich indirekt durch die obigen Modifikationen.

Aufgabe:

Berechnen Sie das minimale Modell des folgenden Programms (durch Iteration des T_P-Operators):

$$\mathrm{answer}(\mathrm{yes}) \leftarrow \mathrm{p}\big(\,.(\mathrm{b}, \mathrm{X})\,\big).$$

$$\mathrm{p}(\mathrm{T}) \leftarrow \mathrm{p}\big(\,.(\mathrm{H}, \mathrm{T})\,\big).$$

$$\mathrm{p}\big(\,.(\mathrm{a},\ .(\mathrm{b},\ .(\mathrm{c},\ [\,])))\,\big).$$

Funktionssymbole (7)

Aufgaben:

Definieren Sie folgende Prädikate zur Listenverarbeitung und geben Sie jeweils Bindungsmuster an, für die Ihre Regeln bereichsbeschränkt sind:

- append(X, Y, Z): Z ist die Konkatenation der Listen X und Y.
- length(X, Y): Y ist die Länge der Liste X.
- member(X, Y): X ist ein Element der Liste Y.

Funktionssymbole (8)

Funktionssymbole sind nicht nötig:

- Statt Funktionssymbolen kann man auch entsprechende Prädikate verwenden:
  - $\mathsf{cons}(\mathsf{H}, \mathsf{T}, \mathsf{L}) \;:\Longleftrightarrow\; \mathsf{L} = \;.(\mathsf{H}, \mathsf{T})$.
  - $\mathsf{date}(\mathsf{T}, \mathsf{M}, \mathsf{J}, \mathsf{D}) \;:\Longleftrightarrow\; \mathsf{D} = \mathsf{date}(\mathsf{T}, \mathsf{M}, \mathsf{J})$.
- Man kann dann strukturierte Terme intern durch neue Variablen ersetzen, die durch ein neues Rumpfliteral dieser Art an den entsprechenden Term gebunden werden.
- Für den Benutzer sind die kürzeren und natürlicheren strukturierten Terme natürlich wichtig.

Funktionssymbole (9)

Auswertbare Funktionen vs. Termkonstruktoren:

- Wenn auch Termkonstruktoren wie . intern mit eingebauten Prädikaten (`cons`) behandelt werden, so kann man die abkürzende Termschreibweise auch für auswertbare Funktionen wie + verwenden.
- Die beiden Arten von Funktionssymbolen können dann sehr ähnlich verwendet werden.
- Aber: Auswertbare Funktionen/Prädikate erlauben meist nur das Bindungsmuster `bb...bf`, Termkonstruktoren dagegen sowohl `bb...bf` als auch `ff...fb` (eindeutig invertierbar).

Funktionssymbole (10)

- Daher muß die Definition von „bereichsbeschränkt“ angepaßt werden, wenn Terme auch auswertbare Funktionen enthalten können.

- Zum Beispiel sind H, T und L gebunden, wenn

$$\mathsf{append}([\mathsf{H}|\mathsf{T}],\ \mathsf{L},\ [\mathsf{H}|\mathsf{TL}]) \leftarrow \mathsf{append}(\mathsf{T},\ \mathsf{L},\ \mathsf{TL})$$

  mit dem Bindungsmuster `bbf` aufgerufen wird.

- Ruft man aber

$$\mathsf{p}(\mathsf{X} + \mathsf{Y}) \leftarrow \dots$$

  z.B. mit $\mathsf{p}(5)$ auf, so sind die Werte von X und Y nicht klar.

Funktionssymbole (11)

Funktionssymbole sind manchmal doch nötig:

- Prolog erlaubt es auch, mit Termen zu arbeiten, die im Innern noch Variablen haben (verletzt Bereichsbeschränkung).
- Bei uns kommen solche Terme nur in Programmen vor, aber nicht in den verarbeiteten Daten.
- So aber keine Meta-Programmierung möglich.
- Z.B. auch in Aditi und manchen Versionen von CORAL entsprechende Einschränkungen.