Einführung in das Programmieren – Prolog
Sommersemester 2006

Teil 1: Ein motivierendes Beispiel

Version 1.0

Gliederung der LV

Teil 1: Ein motivierendes Beispiel

Teil 2: Einführung und Grundkonzepte
- Syntax, Regeln, Unifikation, Abarbeitung

Teil 3: Arithmetik

Teil 4: Rekursion und Listen

Teil 5: Programmfluß
- Negation, Cut

Teil 6: Verschiedenes
- Ein-/Ausgabe, Programmierstil

Teil 7: Wissensbasis
- Löschen und Hinzufügen von Klauseln

Teil 8: Fortgeschrittene Techniken
- Metainterpreter, iterative Deepening, PTTP, Differenzlisten, doppelt verkettete Listen

Unser erstes Prolog-Programm

```
diff(x,1).

diff(C, 0) :-
            atomic(C),
            C \= x.

diff(F + G, DF + DG) :-
            diff(F, DF),
            diff(G, DG).

diff(F - G, DF - DG) :-
            diff(F, DF),
            diff(G, DG).

diff(F * G, DF * G + F * DG) :-
            diff(F, DF),
            diff(G, DG).

diff(F / G, (DF * G - F * DG) / (G * G)) :-
            diff(F, DF),
            diff(G, DG).
```

Bestandteile

- Wir haben ***Regeln*** (auch genannt ***Klausel***)

  ```
  diff(F + G, DF + DG) :- diff(F, DF), diff(G, DG).
  ```

  – linke Seite: ***Kopf*** oder ***Head***

  ```
  diff(F + G, DF + DG)
  ```

  – rechte Seite: ***Rumpf*** oder ***Body***

  ```
                                  diff(F, DF), diff(G, DG)
  ```

  – wird gelesen als *Wenn Body, dann Head.*

  Wenn diff(F, DF) *und* diff(G, DG)*, dann diff(F + G, DF + DG)*

  $\longrightarrow$ Das ***Komma*** steht in Prolog für ***Und***

- Es gibt auch Regeln ohne Rumpf

  ```
  diff(x,1).
  ```

  – wird gelesen als *Head gilt immer.*

- Regeln werden immer mit einem ***Punkt*** abgeschlossen.

Bestandteile

- Die einzelnen Einträge in der Regel heißen ***Literale***

  diff(F + G, DF + DG), atomic(C), C \= x, ...

- Das äußerste Element (Funktor) heißt ***Prädikat***

  diff, atomic, \=

  - mathematisch: ein Prädikat definiert eine Relation
  - Jedes Prädikat hat ***Argumente***
    - Anzahl der Argumente wichtig!
    - schreiben deshalb manchmal die Argumentzahl mit auf

      diff/2, atomic/1, \=/2, ...

- Als Argumente der Prädikate können beliebige ***Terme*** auftreten
  - Konstanten und Variablen sind Terme
  - Wenn $t_1, \ldots, t_n$ Terme, dann auch $f(t_1, \ldots, t_n)$
  - Manche Funktoren in ***Infixnotation***

    (DF * G - F * DG) / (G * G) statt /(-(*(DF,G),*(F,DG)),*(G,G))

Bestandteile

- Jede Regel bezieht sich (definiert) auf genau ein Prädikat (***Kopfprädikat/Zielprädikat***)

  diff(C, 0) :- atomic(C), C \= x.
  heißt also:
  Wenn das Prädikat atomic *für das Argument* C *wahr ist und das Prädikat* \= *für die Argumente* C *und* x *wahr ist, dann ist auch das Prädikat* diff *für die Argumente* C *und* 0 *wahr*

- Wichtig: die Argumentterme werden nicht weiter ausgewertet!
  - Die Funktionszeichen +, -, *, / in unserem Beispiel haben *keinerlei* Bedeutung für das Prologprogramm

Wir experimentieren

Wir lassen das Programm einmal abarbeiten:

- Prolog starten: `pl`

```
> pl
Welcome to SWI-Prolog (Version 3.4.2)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).

?-
```

- Prolog ist per Default interaktiv, d.h. wir bekommen einen Prompt (hier: `?-`.
- Wir laden nun das File mit unserem Programm:

```
?- [diff].
% diff compiled 0.00 sec, 1,676 bytes

Yes
```

- Die Antwort ist `Yes`, es hat alles geklappt

Wir experimentieren

- Nun berechnen wir einmal `diff`:

```
?- diff(x+x,1+1).
Yes
```

- Aha, das haben wir uns schon gedacht.
- Und nun:

```
?- diff(x*x,1+1).
No
```

- Ok, aber was ist so toll daran???

Wir experimentieren

- Wie muß eigentlich das zweite Argument bei diff(x * x, ·) lauten?
  - Anmerkung: Variablen beginnen mit Großbuchstaben, alles andere sind Konstanten

```
?- diff(x*x,ARG2).

ARG2 = 1*x+x*1
```

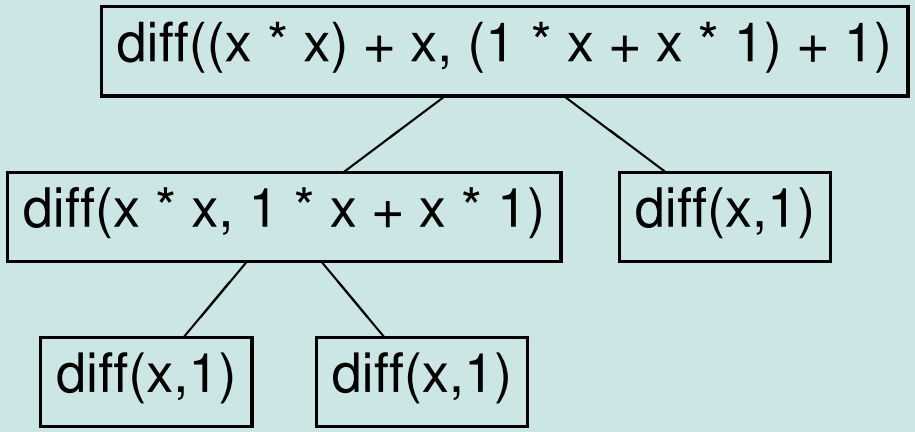
- Aha, schon viel besser.
- Und wie für diff((x * x) + x, ·)?

```
?- diff((x * x) + x, ARG2).

ARG2 = 1*x+x*1+1
```

Wie kommt das Ergebnis zustande?

```
diff(x,1).

diff(F + G, DF + DG) :-
        diff(F, DF),
        diff(G, DG).

diff(F * G, DF * G + F * DG) :-
        diff(F, DF),
        diff(G, DG).
```

```
diff(C, 0) :-
        atomic(C),
        C \= x.

diff(F - G, DF - DG) :-
        diff(F, DF),
        diff(G, DG).

diff(F / G, (DF * G - F * DG) / (G * G)) :-
        diff(F, DF),
        diff(G, DG).
```



Wir experimentieren

- Geht das eigentlich auch andersrum?

```
?- diff(ARG1, 1 - 1).

ARG1 = x-x
```

- Und ganz ohne Vorgaben?

```
?- diff(ARG1, ARG2).

ARG1 = x
ARG2 = 1
```

- Gibt es noch mehr Lösungen?
  - Wir drücken `;`

```
ARG1 = x+x
ARG2 = 1+1 ;

ARG1 = x+ (x+x)
ARG2 = 1+ (1+1) ;

ARG1 = x+ (x+ (x+x))
ARG2 = 1+ (1+ (1+1)) ;
```

Wir experimentieren

- Gibt es Lösungen mit Minus?

```
?- diff(ARG11 - ARG12, ARG2).

ARG11 = x
ARG12 = x
ARG2 = 1-1 ;

ARG11 = x
ARG12 = x+x
ARG2 = 1- (1+1) ;

ARG11 = x
ARG12 = x+ (x+x)
ARG2 = 1- (1+ (1+1)) ;

ARG11 = x
ARG12 = x+ (x+ (x+x))
ARG2 = 1- (1+ (1+ (1+1)))
```

Wir experimentieren

- Was besagt eigentlich die zweite Regel diff(C, 0) :- atomic(C), C \= x.?
  - Was bedeutet `atomic`?

```
?- help(atomic).

atomic(+Term)
    Succeeds  if Term is bound to  an atom, string,
    integer or  floating point number.

Yes
```

  - Aha, also C darf kein ”echter” Term sein

- \= bedeutet ”nicht =”

```
?- diff(y,ARG2).

ARG2 = 0

?- diff(2*y*x,ARG2).

(0*y+2*0)*x+2*y*1
```

Die wichtigen Wirkprinzipien

- ***Regeln*** definieren Prädikate
- ***Terme*** sind die eigentlichen Daten
- ***Variablen*** können ersetzt werden

Was macht Prolog?

- Wendet die Regeln an.
- Sucht geeignete Ersetzungen der Variablen.
- Stellt grundlegende Prädikate bereit

Was lernen wir eigentlich noch?

- Verständnis des Regelabarbeitungsmechanismus $\longrightarrow$ ***Resolution***
- Verständnis der Ersetzungen $\longrightarrow$ ***Unifikation***
- Kennenlernen grundlegender Built-In-Prädikate
- Tricks und Kniffe