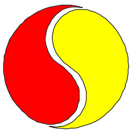
Funktional-logische Sprachen



BABEL

Deklarative Sprachklassen

Funktionale Sprachen
- Funktionen höherer Ordnung
- Polymorphe Typen
- Pattern Matching
- Unendliche Datenstrukturen

Funktionen
Reduktionsprinzip
Normalform

Logische Sprachen
- Nichtdeterminismus (suchbasierte Berechnungen)
- Bidirektionalität von Prädikats-argumenten
- partielle Strukturen

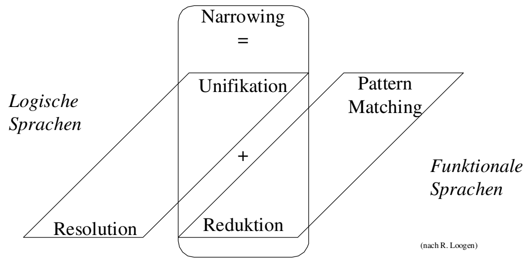
Hornklauseln
SLD-Resolution/Unifikation
Antwortsubstitution

Idee der Integration

	kompositionelle Semantik	**operationelle Semantik**
funktionale Sprachen	funktional	reduktionsbasiert
	$\cup$	$\cap$
logische Sprachen	prädikativ	logisch

(nach R. Loogen)

Operationelle Semantik



(nach R. Loogen)

Typen, Terme, Ausdrücke

Typsystem entspricht dem in funktionalen Sprachen verwendeten polymorphen von Hilner/Milner (vgl. ML).

Basistypen (nat, bool) und **Typkonstruktoren** (->, *, ...)

Typausdrücke:

Bsp.	A, nat, list(A), A ->B, ...	/* allg. Typen */
	nat, list(nat), nat ->bool, ...	/* Monotypen (variablenfrei) */
	A, list(A), list(nat), ...	/* Strukturtypen (ohne →) */

Typdeklarationen:

Bsp.	type nat	= 0 | (succ nat).	
	type bool	= true | false.	/* vordefiniert */
	type list(A)	= [] | [A | list(A)].	/* polymorph */

[], [true], ... , Y, [true|Y], [false,Y], ... /* **Terme** vom Typ list(bool) */

if ((X=(succ 0)) then (succ X) else 0. /* **Ausdruck** vom Typ nat */

Syntax von BABEL

Funktionale Sprache
$f\ \ t_1 \dots t_m = M;$

Logische Sprache
$p\ (t_1, \dots, t_m)$:- B.

Funktional-logische Sprache
$f\ \ t_1 \dots t_m = M$:- B.

$$\underbrace{f\ \ t_1 \dots t_m}_{\text{linke Seite}} := \underbrace{\underbrace{B}_{\substack{\text{Bedingung}\\ \text{(optional)}}} :\!-\ \underbrace{M.}_{\text{Ziel}}}_{\text{rechte Seite}}$$ /* Babel-Syntax */

Regeln

$$\underbrace{f\ \ t_1 \ldots t_m}_{\text{linke Seite}} := \underbrace{\underbrace{B :-}_{\text{Bedingung (optional)}}\ \ \underbrace{M.}_{\text{Ziel}}}_{\text{rechte Seite}}$$

Die Definition von Funktionen erfolgt in BABEL durch bedingte Regeln, wobei

a) auf der linken Seite keine Variable mehrfach auftritt (*Linkslinearität*),
b) Variablen *frei* heißen, wenn sie nur auf der rechten Seite auftreten,
c) der Typ einer freien Variablen ein *Strukturtyp* sein muss und
d) freie Variablen nur in der Bedingung auftreten dürfen (*lokaler Determinismus*).

Programme

$$f\ \ t_1 \ldots t_m \quad := \{B :- \}\ M.$$
$$f\ \ s_1 \ldots s_m \quad := \{C :- \}\ N.$$

Bedingungen

- Die Stelligkeit einer Regel für ein Funktionssymbol f ist immer kleiner oder gleich der Stelligkeit des Funktionssymbols und
- alle Regeln für ein Funktionssymbol f müssen dieselbe Stelligkeit haben.,
- je zwei variablendisjunkte Regelvarianten für ein Funktionssymbol f müssen mindestens eine der drei Bedingung erfüllen (Eindeutigkeit):
  - a) *keine Überlagerung*: die linken Seiten $f\ \ t_1 \ldots t_m$ und $f\ \ s_1 \ldots s_m$ sind nicht unifizierbar
  - b) *Verschmelzung der Rümpfe*: $f\ \ t_1 \ldots t_m$ und $f\ \ s_1 \ldots s_m$ haben einen allgemeinsten Unifikator sub_U und M sub_U und N sub_U sind syntaktisch identisch
  - c) *Unverträglichkeit der Bedingungen*: $f\ \ t_1 \ldots t_m$ und $f\ \ s_1 \ldots s_m$ haben einen allgemeinsten Unifikator sub_U und B sub_U und C sub_U sind inkompatibel, d. h. sie ergeben niemals beide den Wert true

Bsp. BABEL-Programm

```
append [] Ys         := Ys.
append [X|Xs] Ys     := [X|append Xs Ys)].

sublist Xs Ys    :=    ((append L1 (append Xs L2)) = Ys) :- true.
```

Operationelle Semantik - Narrowing

Auch freie bzw. logische Variablen in Zielausdrücken*

Berechnung mittels
- Narrowing-Regeln
- Narrowing-Relation

Bedingter Ausdruck + Bindungen freier Variablen

*Zielausdrücke: Menge aller eventuell bedingten Ausdrücke, in denen nur Variablen mit einem Strukturtyp auftreten.

Narrowing-Regeln

Lokale Auswertungsschritte

1) Verzweigungsregeln:

Fall a: reine Reduktionsschritte

$(\text{true} \rightarrow M) \Rightarrow_{\varepsilon} M$	if true then M_1 else $M_2 \Rightarrow M_1$
	if false then M_1 else $M_2 \Rightarrow M_2$

Fall b: Bedingungsausdruck ist eine Variable, die gebunden wird

$(X \rightarrow M) \Rightarrow_{[X/tru]} M$	if X then M_1 else $M_2 \Rightarrow_{[X/tru]} M_1$
	if X then M_1 else $M_2 \Rightarrow_{[X/tru]} M_2$

2) Funktionsanwendung:

$(f\ M_1 \ldots M_m) \Rightarrow_{sub_U} R\theta$	mit $(f\ t_1, \ldots, t_m := R)$ ist Regel in P

$(sub_U \cup \theta)$ ist allgemeinster Unifikator der Parameterterme t_i und der Argumentausdrücke M_i mit $t_i\, \theta = M_i\, sub_U$

Bsp. Narrowing-Regeln

Bsp. 1: append L [(succ 0)]

1. Regel	append	[]	Ys	:=	Ys
Unifikation		↓	↑		
Narrowing:	(append	L	[(succ 0)]	$=>_{[L/[]]}$	[(succ 0)]
2. Regel	append	[X|Xs]	Ys	:=	[X|(append Xs Ys)]
Unifikation		↓	↑		
Narrowing:	(append	L	[(succ 0)])	$=>_{[L/[X\|Xs]]}$	[X|(append Xs [(succ 0)])]

Bsp. 2: append [0] [(succ 0)]

2. Regel	append	[X|Xs]	Ys	:=	[X|(append Xs Ys)]
Matching		↑	↑		
Reduktion:	(append	[0 | []]	[(succ 0)]	$=>_{\varepsilon}$	[0|(append [] [(succ 0)])]

Narrowingrelation

1) M $\rightarrow_{sub_U}$ N falls M $\Rightarrow_{sub_U}$ N

2) (B -> M) $\rightarrow_{sub_U}$ (B‘ -> (M sub_U)) und
if B then M_1 else M_2 $\rightarrow_{sub_U}$ if B‘ then (M_1 sub_U) else (M_2 sub_U),

falls B $\rightarrow_{sub_U}$ B‘

3) (M N) $\rightarrow_{sub_U}$ ((M sub_U) N‘) und (N M) $\rightarrow_{sub_U}$ (N‘ (M sub_U)),

falls N $\rightarrow_{sub_U}$ N‘

Bsp. Narrowing

Zielausdruck: (sublist [0] [(succ 0), 0)]

$\rightarrow_{\varepsilon}$ ((append L1 (append [0] L2)) = [(succ 0), 0)]) -> true
$\rightarrow_{[L1/[X|Xs]]}$ ([X|(append Xs (append [0] L2))] = [(succ 0), 0)]) -> true
$\rightarrow_{\varepsilon}$ ((X = (succ 0)) ∧ ((append Xs (append [0] L2)) = [0])) -> true
$\rightarrow_{[X/(succ X‘)]}$ ((X‘ = 0) ∧ ((append Xs (append [0] L2)) = [0])) -> true
$\rightarrow_{[X‘/0]}$ (true ∧ ((append Xs (append [0] L2)) = [0])) -> true
$\rightarrow_{\varepsilon}$ ((append Xs (append [0] L2)) = [0])) -> true
$\rightarrow_{[Xs/[]]}$ ((append [0] L2)) = [0] -> true
$\rightarrow_{\varepsilon}$ ([0]|append [] L2)] = [0]) -> true
$\rightarrow_{\varepsilon}$ ([0]|L2] = [0]) -> true
$\rightarrow_{\varepsilon}$ (0 = 0) ∧ (L2 = []) -> true
$\rightarrow_{\varepsilon}$ (true ∧ (L2 = []) -> true
$\rightarrow_{\varepsilon}$ (L2 = []) -> true
$\rightarrow_{[L2/[]]}$ true -> true
$\rightarrow_{\varepsilon}$ true

Programmbeispiel

```
/* Suche nach Ländern mit gleicher Bevölkerungsdichte */

population china := 8250.
population india  := 5863.
...
area china := 3380.
area india := 1139.
...
density Country := div (mult 100 (population Country)) (area Country)
samedensity (C1, D1, C2, D2) :-
     D1 = (density C1),
     D2 = (density C2),
     D1 > D2,
     (20*D1) < (21*D2).

/* samedensity1 (C1, C2) :=
     letpar D1 = (density C1),
            D2 = (density C2),
     in if (D1 > D2, (20*D1) < (21*D2)) then true.  */
```

Bsp. Fibonacci

```
/* fibonacci with infinite datatypes  */

fibnbs X Y := [X| fibnbs Y (X+Y)].

fiblist  := fibnbs 1 1.

fib X := nth X fiblist.

nth X [Y|Ys]  := if (X=0) then Y else nth (X-1) Ys.


/*  goal: fib 20   */
```


Bsp. N-Damen-Problem

```
%import append from "append".

allqueens(N,Ds):- queens1(range 1 N,[],Ds),false.
queens(N,Ds):- queens1(range 1 N,[],Ds).

queens1([],Ds,Es):-Ds=Es.
queens1(Dsunp,Dssicher,Ds):- select(D, Dsunp, Ds1unp) , noattack(D,Dssicher),
                    queens1(Ds1unp,[D|Dssicher],Ds).
range N1 N2 :=if (N1 = N2) then [N1].
range N1 N2 :=if (N1 ~= N2) then [N1|range (N1+1) N2].

sicher([D|Ds]):- sicher(Ds),noattack(D,Ds).
sicher([]).

select(Y,[X|Xs],Zs) :- X=Y,Zs=Xs.
select(Y,[X|Xs],[Z|Zs]) :- X=Z,select(Y,Xs,Zs).

noattack(X,Xs):-noattack1(X,1,Xs).
noattack1(X,N,[]).
noattack1(X,N,[Y|Ys]):-  ((X ~= (Y+N) ) , ( X~= (Y-N) ) ) , noattack1(X,(N+1),Ys).
```


Alternative Ansätze

Vorgehensweisen:

1) die Entwicklung symmetrischer Kombinationen separater Sprachen
   - keine echte Integration, sondern Aufruf der einen Sprache von der anderen Sprache aus

2) die Erweiterung von logischen Sprachen um funktionale Konzepte
   - Hanus, M.: Integration of functions into logic programming: from theory to practice, Journal of logic programming, Vol. 19&20, 1994, 583-628

3) die Erweiterung funktionaler Sprachen um logische Konzepte
   - zusätzlich log. Variablen und Nichtdeterminismus im Auswertungsmodell

Erweiterung funktionaler Sprachen

funktional-logische Sprachen
- Narrowing-Mechanismus
- syntaktische Restriktionen => Vermeidung nicht-deterministischer Funktionen
- Erweiterung um logische Variablen in den Bedingungsausdrücken

Mengensabstraktionen
relative:
[Ausdruck(Variablen) | Generator(Variablen), Bedingungen(Variablen)]
absolute:
[Ausdruck(Variablen) | Bedingungen(Variablen)]

Bsp.
Zerlegung von Listen: (append L1 L2) = [...] (*z. B. in BABEL *)
als Bedingungsausdruck in funkt. Sprachen mit Mengenabstraktion:
split L := [(L1,L2) | (append L1 L2) = L].