Logické programování

- *logický program*: libovolná konečná množina programových Hornových klauzulí
- odvozování (dokazování) cílů založeno na SLD-rezoluci
- deklarativní (specifikace programu je přímo programem)
- teoretický model, zachovává úplnost

Prolog

- konkrétní implementace logického programovacího jazyka
- strategie prohledávání stavového prostoru (SLD-stromu) do hloubky (výběr programové klauzule shora dolů, výběr podcíle zleva doprava)
- díky prohledávání do hloubky ztráta úplnosti (uváznutí v nekonečné větvi SLD-stromu)
- vhodný k řešení problémů týkajících se objektů a vztahů mezi nimi
- do značné míry využívá rekurzi
- historie: 70. l. 20. stol. – Colmerauer, Kowalski; D.H.D. Warren (WAM)

Syntax jazyka Prolog I

Datové objekty

- termy (konstanty, proměnné, složené termy)
- konstanty:
  - celá čísla (`0, 123, -12`),
  - desetinná čísla (`1.0, 4.5E7, -0.12e+8`),
  - atomy (`'Petr Parléř', [], s1, ==, 'ježek', atom`)
- proměnné (`N, _VYSLEDEK, Hodnota, A1, _12`), anonymní proměnná (`_`)
- složené termy: funktor (jméno, arita) a argumenty (`bod(X,Y,Z), tree(Value,tree(LV,LL,LR),tree(RV,RL,RR))`)
  poznámka: funktory jako operátory – použití infixového zápisu (`X < Y` místo `<(X,Y)` apod.)

Syntax jazyka Prolog II

Program

- množina programových klauzulí (pravidla, fakta)
- proměnné lokální v klauzuli, kde se vyskytují
- pravidla: hlava, tělo
  `date(D,M,Y):- day(D), month(M), year(R).`
- fakta: pravidla s prázdným tělem
  `date(14,'February',2001).`
- cíle: klauzule bez hlavy, reprezentují dotazy – spuštění výpočtu
  `?- date(29,'January',2001).`

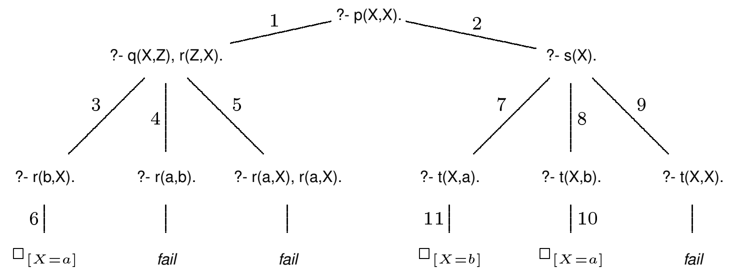
Explicitní unifikace: operátor =

Př:. `X=Y, f(g(a,X))=f(Y)`

SLD-strom pro prologovský program: srovnání

1. p(X,Y) :- q(X,Z), r(Z,Y).
2. p(X,X) :- s(X).
3. q(X,b).
4. q(b,a).
5. q(X,a) :- r(a,X).
6. r(b,a).
7. s(X) :- t(X,a).
8. s(X) :- t(X,b).
9. s(X) :- t(X,X).
10. t(a,b).
11. t(b,a).

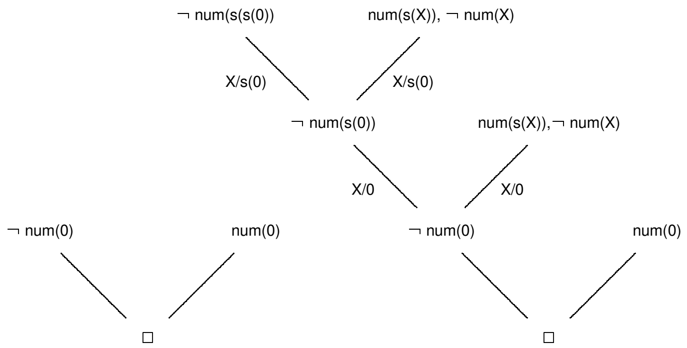
?- p(X,X).



SLD-odvození pro prologovský program

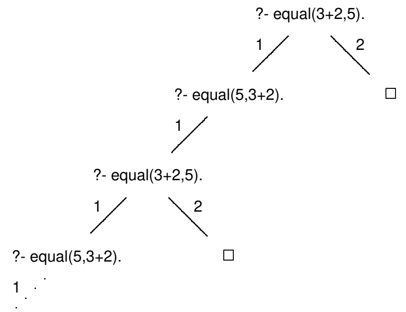
Příklad: odvození cílů pro program:

```
num(0).                  ?- num(0).
num(s(X)):- num(X).      ?- num(s(s(0))).
```



Příklad: ukázka neúplnosti systému

```
1.   equal(X,Y):- equal(Y,X).
2.   equal(3+2,5).
     ?- equal(3+2,5).
```



Seznam

- rekurzivní datová struktura
- uspořádaná posloupnost prvků (libovolných termů včetně seznamů)
- funktor `./2`; prázdný seznam `[]`
- `.(Hlava,Tělo)`, alternativně `[Hlava|Tělo]`, `Tělo` je seznam

`.(a,[])`	`[a]`	`[a|[]]`
`.(a,.(b,.(c,[])))`	`[a,b,c]`	`[a,b|[c]]`
	`[a|[b,c]]`	`[a,b,c|[]]`
	`[a|[b,c|[]]]`	`[a|[b|[c|[]]]]`

- možnost reprezentace pomocí stromů

Operace se seznamy: ‚být prvkem' I

možné různé varianty:

1) pomocí unifikace:

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

?- member(a,[b,c,a]).
Yes
?- member(a,[X,b,c]).
X=a
Yes
```


Operace se seznamy: ‚být prvkem' II

2) pomocí identity:

```
member(X,[Y|_]):- X == Y.
member(X,[_|T]):- member(X,T).
?- member(a,[X,b,c]). % No
?- member(a,[a,b,a]),write(ok),nl,fail.
ok
ok
No
```

3) bez vícenásobných výskytů:

```
member(X,[Y|_]):- X == Y.
member(X,[Y|T]):- X \== Y, member(X,T).
?- member(a,[a,b,a]),write(ok),nl,fail.
ok
No
```


Spojení dvou seznamů

```
append([],L,L).
append([H|T1],L2,[H|T]):- append(T1,L2,T).
------------------------------------------
?- append([a,b],[c,d],L).
L = [a, b, c, d]
Yes
?- append(X,[c,d],[a,b,c,d]).
X = [a, b]
Yes
?- append(X,Y,[a,b,c]).
X = []          Y = [a, b, c];
X = [a]         Y = [b, c];
X = [a, b]      Y = [c];
X = [a, b, c]   Y = [];
No
```


Seznamy: obrácení pořadí prvků

```
reverse([],[]).
reverse([H|T],L):- reverse(T,L1), append(L1,[H],L).
---------------------------------------------------
?- reverse([a,b,c],L).
L = [c, b, a]
Yes
?- reverse([a,b,c],[c,b,a]).
Yes
?- reverse(L,[a,b,c]).
L = [c, b, a]
Yes
```

Další operace se seznamy: permutace, prefix, postfix, podseznam, třídění…

Seznamy: třídění I

0) naivní třídění: strategie generuj a testuj (pokud možno nepoužívat)

```
naive_sort(L,S):- perm(L,S), sorted(S).
 % perm vrací postupně všechny permutace seznamu
sorted([]).
sorted([_]).
sorted([X,Y|T]):- X=<Y, sorted([Y|T]).
```

1) třídění vkládáním (insert sort)

```
insert(X,[],[X]).
insert(X,[Y|T1],[Y|T2]):- X>Y, insert(X,T1,T2).
insert(X,[Y|T1],[X,Y|T1]):- X=<Y.

isort([],[]).
isort([X|L],S):- isort(L,Y), insert(X,Y,S).
```


Seznamy: třídění II

2) přístup ‚rozděl a panuj': quick sort

```
qsort([],[]).
qsort([H],[H]).
qsort([H|T],L):-
  divide(H,T,M,V),
  qsort(M,M1),
  qsort(V,V1),
  append(M1,[H|V1],L).

divide(_,[],[],[]).
divide(H,[K|T],[K|M],V):- K=<H, divide(H,T,M,V).
divide(H,[K|T],M,[K|V]):- K>H, divide(H,T,M,V).
```


Aritmetika v Prologu

- funkční operátory: `+ - * / mod` (a další: `^ // sqrt()`)
- vyhodnocení výrazů: `is`

```
?- A is 3*(4+2).
A=18
Yes
?- A is 3*(B+2).
chyba
```

- relační operátory: `< > >= =<`

```
?- 3*4 > 2.
Yes
?- B =< 14.
chyba
```


Příklad: symbolické derivace

```
derive(X,V):- d(X,A), simpl(A,V).
d(x, 1).
d(N, 0)              :- number(N).
d(-X, -A)            :- d(X,A).
d(X + Y, A + B)      :- d(X,A), d(Y,B).
d(X - Y, A - B)      :- d(X,A), d(Y,B).
d(X * Y, A*Y + B*X):- d(X,A), d(Y,B).
d(X / Y, (A*Y - X*B) / Y^2):- d(X,A), d(Y,B).
d(X ^ N, N*X^M * C):- number(N), M is N-1, d(X, C).
d(F^G, F^G*(B*log(F) + A*G/F)) :- d(G, B), d(F, A).
d(log(X), 1/X * Y) :- d(X,Y).
d(exp(X), exp(X)*Y):- d(X,Y).
d(sin(X), cos(X)*Y):- d(X,Y).
d(cos(X), -sin(X)*Y):- d(X,Y).
d(arctg(X), 1/(1+X^2)*Y):- d(X,Y).
```

Programování: užitečné predikáty

- načtení programu:

```
consult('program.pl').
['program.pl'].
['program.pl',program2].
reconsult(program).
```

- interaktivní zadání programu (ukončení `ctrl+D`):

```
[user].
```

- výpis interaktivně zadaného kódu:

```
listing.
```

- ukončení práce:

```
halt.
```


Prology na FI

- SICStus Prolog (module add sicstus; sicstus) – komerční
- ECLiPSe (module add eclipse; eclipse) – komerční
- SWI (module add pl; pl) – volně šiřitelný
- unixové stroje aisa, erinys, oreias, nymfe
- sicstus pod Windows (lze nainstalovat i jiné)
- další volně šiřitelné Prology (yap), možnost instalace na různé platformy