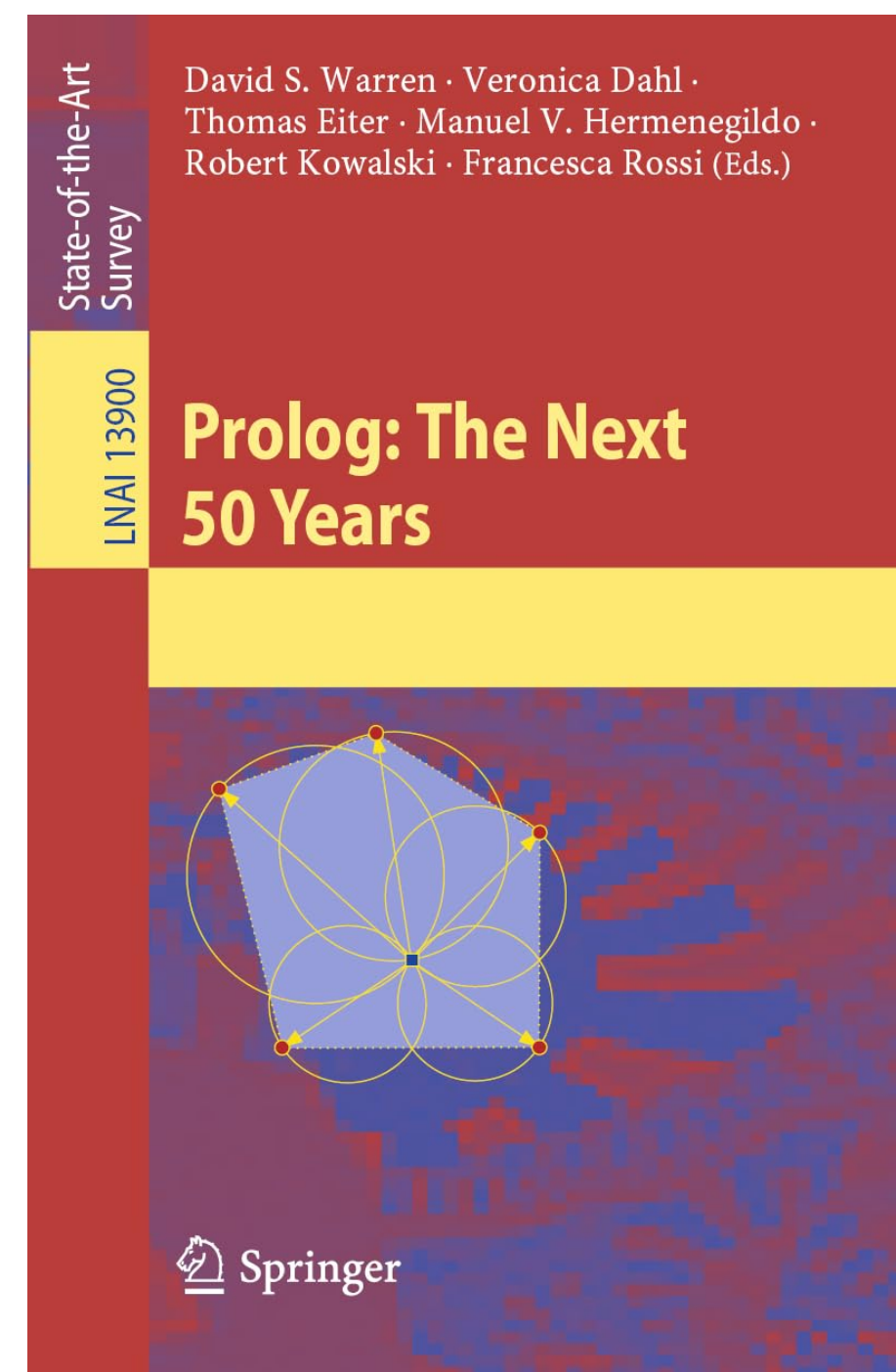






Neprocedurální programování

Prolog 6

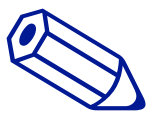

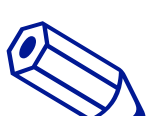

Epilog



Co bylo minule

-  Prohledávání grafů
-  Prohledávání stavového prostoru
-  Shromáždění všech výsledků dotazu
 - bagof, setof, findall
-  Vstup a výstup
 - příklad : Eliza

Osnova

-  Eliza : dokončení
-  Operátory
-  Predikáty pro modifikaci programu
 - příklad : implementace `findall/3`
-  Práce s množinami řešení

Minule : Zpracování přirozeného jazyka

Eliza: Dialog s psychoanalytickou

- J. Weizenbaum, ELIZA - A computer program for the study of natural language communication between man and machine.
Comm. of the ACM **9** (1966), 36-45.

```
eliza:- write('Hello. My name is Eliza.  
          How can I help you?'),  
        nl, cti_vetu(V), eliza(V), !.  
eliza(Vstup):- member('quit', Vstup),  
                write('Goodbye.  
My secretary will send you a bill. '), nl.
```



Eliza: zpracování vstupu

```
% typ_znaku(+KodZnaku,?Typ):-  
% Typ znaku se zadany kodem KodZnaku,  
% Typ je oddelovac, konec_vety nebo jiny.  
typ_znaku(Z,konc_vety) :- member(Z,[33,46,63]), !.  
                                % vykřičník, tečka, otazník  
typ_znaku(Z,oddelovac)      :- Z =< 32, !.  
                                % mezery apod.  
typ_znaku(_,jiny).
```

Eliza: načtení slova

```
% cti_slovo(+Pismeno,-S,-DalsiZnak):-  
%                               vrátí seznam S písmen slova,  
%                               které začíná Pismenem  
%                               a za ním následuje DalsiZnak.  
  
cti_slovo(Z,[ ],Z):-  
    typ_znaku(Z,konec_vety),!.    % konec věty  
  
cti_slovo(Z,[ ],Z):-  
    typ_znaku(Z,oddelovac),!.    % konec slova  
  
cti_slovo(Pis,[Pis|SezPis],DalsiZnak):-  
    get_code(Znak),  
    cti_slovo(Znak,SezPis,DalsiZnak).
```


Eliza: načtení věty

```
% cti_vetu(-SeznamSlov):- přečte na vstupu větu  
%                               a vrátí SeznamSlov věty.
```

```
cti_vetu(SezSlov):-  
    get_code(Znak), cti_zbytek(Znak,SezSlov).
```

```
cti_zbytek(Z,[ ]):-  
    typ_znaku(Z,konec_vety), !.    % konec věty
```

```
cti_zbytek(Z,SezSlov):-  
    typ_znaku(Z,oddelovac), !,    % mezera apod.  
    cti_vetu(SezSlov).
```

```
cti_zbytek(Pismeno,[Slovo|SezSlov]):-  
    cti_slovo(Pismeno,SezPis,DalsiZnak),  
    name(Slovo,SezPis),  
    cti_zbytek(DalsiZnak,SezSlov).
```

Operátory

`:- op(Priorita, Druh, Jmeno) .`

- deklarace operátoru

Priorita

- přirozené číslo $\in \langle 1, 1200 \rangle$
- nižší hodnota \rightarrow váže více

Druh

- určuje aritu (unární, binární)
- pozici (infix, prefix, postfix)
- asociativitu

Jmeno

- atom nebo seznam atomů

Předdefinované operátory

```
: - op( 200, fy, [ +, -, \ ] ).  
: - op( 200, xfy, ^ ).  
: - op( 200, xfx, ** ).  
: - op( 400, yfx, [ *, /, //, <<, >>, mod, div, rem, xor ] ).  
: - op( 500, yfx, [ -, +, \/, /\, ] ).  
: - op( 700, xfx, [ >, =, <, >=, =<, \=, @<, @=<,  
    @>, @>=, =@=, ==, =\=, is, =.., ==, \== ] ).  
: - op( 900, fy, \+ ).  
: - op( 1000, xfy, ', ' ).  
: - op( 1050, xfy, -> ).  
: - op( 1100, xfy, ; ).  
: - op( 1105, xfy, ' | ' ).  
: - op( 1200, fx, [ :-, ?- ] ).  
: - op( 1200, xfx, [ :-, --> ] ).
```

Operátory: druh

f reprezentuje operátor

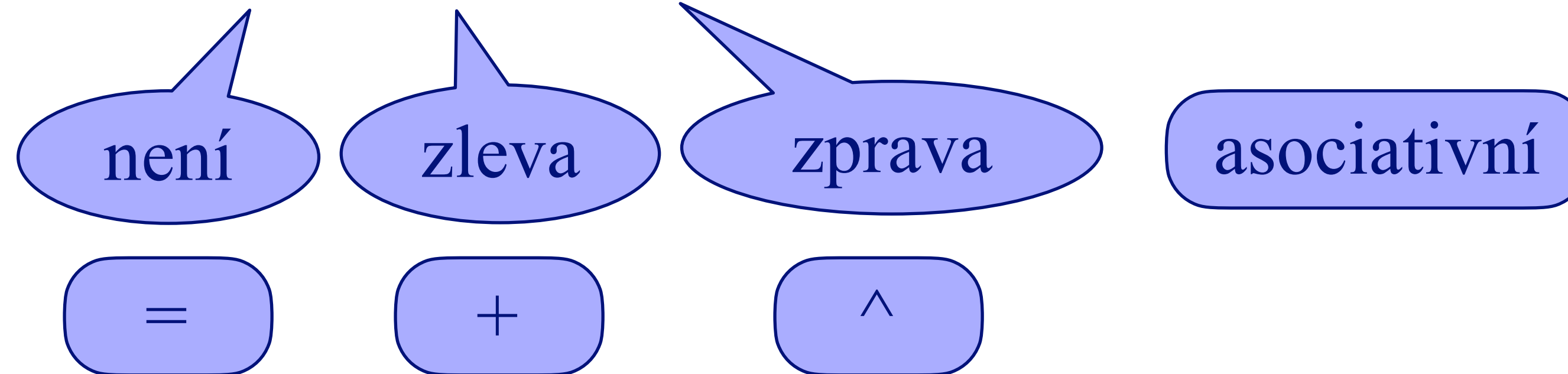
x y reprezentují operandy

- unární

» v prefixové notaci: **fx fy**

» v postfixové notaci: **xf yf**

- binární: **xfx yfx xfy**



Zjištění definovaných operátorů

- `current_op(?Pri, ?Druh, ?Jmeno)`

☀ Příklad: formule výrokového počtu

Můžeme vytvořit obvyklým způsobem z

- konstant **true/0**, **false/0**
- spojek **non/1**, **and/2**, **or/2**, **xor/2**, **imp/2**, **ekv/2**
- závorek
- výrokových proměnných **p/1**, např. **p(a)**

Spojky lze definovat jako operátory

```
:- op(200, fy, non).  
:- op(210, yfx, and).  
:- op(215, xfx, xor).  
:- op(220, yfx, or).  
:- op(230, xfy, imp).  
:- op(240, xfx, ekv).
```

Predikáty pro modifikaci programu

Umožní **přidávat** nové či **vyřazovat** existující klauzule programu

- mění deklarativní význam programu
- zpomalení výpočtu
- možnost simulace přiřazovacího příkazu

Predikát definovaný modifikovanou procedurou je třeba označit jako **dynamický**

- `:- dynamic predikat/2, jiny_predikat/1.`

Predikáty `assert/1`, `retract/1`

`asserta(+T)` přidá term **T** jako novou klauzuli na začátek programu v paměti

`assertz(+T)` přidá term **T** jako novou klauzuli na konec programu v paměti

- `assert/1` ekvivalentní `assertz/1`

`retract(?T)` odstraní z programu v paměti první výskyt klauzule, kterou lze unifikovat s **T**

`retractall(?T)` odstraní z programu v paměti všechny klauzule, jejichž hlavu lze unifikovat s termem **T**

findall pomocí assert & retract

```
findall(X,Cil,SezVys):- zapis(X,Cil),  
                        seber([],SezVys).
```

```
zapis(X,Cil):- Cil,  
               asserta(data999(X)),  
               fail.
```

```
zapis(_,_).
```

```
seber(S,SezVys) :- data999(X),  
                  retract(data999(X)),  
                  seber([X|S],SezVys),!.
```

```
seber(SezVys,SezVys).
```


Práce s množinami řešení bez `findall`

```
% komb(+N,+Mnozina,-Komb):- Komb je kombinace
%                                radu N prvku Mnoziny.

komb(0,_,[]).
komb(N,[X|Xs],[X|Ys]):- N>0, N1 is N-1,
                        komb(N1,Xs,Ys).
komb(N,[_|Xs],Ys):- N>0, komb(N,Xs,Ys).
```

Všetchny kombinace

```
% skomb(+N,+Mnozina,-SKomb):- SKomb je seznam  
%      vsech kombinaci radu N prvku Mnoziny.  
  
skomb(0,_,[[ ]).  
  
skomb(N,[ ],[ ]):- N>0.  
  
skomb(N,[X|Xs],SKomb):- N>0, N1 is N-1,  
                          skomb(N1,Xs,Yss),  
                          map_insert(X,Yss,Yss1),  
                          skomb(N,Xs,Zss),  
                          append(Yss1,Zss,SKomb).
```

vloží X do hlavy
každé kombinace
v Yss

Pomocný predikát map_insert

```
% map_insert(?X,?Xss,?Yss):- vlozi X do hlavy  
%           kazdeho seznamu v Xss a vrati v Yss.  
  
seznam(X,Xs,[X|Xs]).  
  
map_insert(X,Xss,Yss):- maplist(seznam(X),Xss,Yss).
```

Srovnání: kombinace v LISPu (Scheme)

≡ komb.scm

```
1 #lang scheme
2 ;;;;;;;;;;;;;;;;;;;;;;;;;; Kombinace bez opakování ;;;;;;;;;;;;;;;;;;;;;;;;;;
3 (define (komb rad seznam)
4   (cond ((zero? rad) '())
5         ((null? seznam) '())
6         (else (append (map (λ (rad-1) (cons (car seznam) rad-1))
7                               (komb (- rad 1) (cdr seznam)))
8                         (komb rad (cdr seznam))
9                     )
10      )
11 )
12 )
```

PROBLÉMY VÝSTUP TERMINÁL PORTY KONZOLA LADĚNÍ

rackety

- (base) td@MacBookPro sources % racket
Welcome to Racket v8.16 [cs].
> (enter! "komb.scm")
"komb.scm"> (komb 2 '(1 2 3))
'((1 2) (1 3) (2 3))

Srovnání: kombinace v Haskellu

» komb.hs > ...

```
1  -- kombinace bez opakovani
2  komb :: Int -> [a] -> [[a]]
3
4  komb 0 _ = [[]]
5  komb _ [] = []
6  komb n (x:xs) | n>0 = map (x:) (komb (n-1) xs) ++ komb n xs
7  komb _ _ | otherwise = error "wrong argument"
```

PROBLÉMY VÝSTUP TERMINÁL PORTY KONZOLA LADĚNÍ

ghc-9.10.1 -

```
◦ (base) td@MacBookPro sources % ghci
GHCi, version 9.10.1: https://www.haskell.org/ghc/  :? for help
ghci> :l komb
[1 of 2] Compiling Main                ( komb.hs, interpreted )
Ok, one module loaded.
ghci> komb 2 [1,2,3]
[[1,2],[1,3],[2,3]]
```

Kam dále?

NAIL120 Úvod do umělé inteligence
(Roman Barták)

NOPT042 Programování s omezujícími podmínkami
(Roman Barták)

NAIL076-7 Logické programování I,II
(Jan Hric)