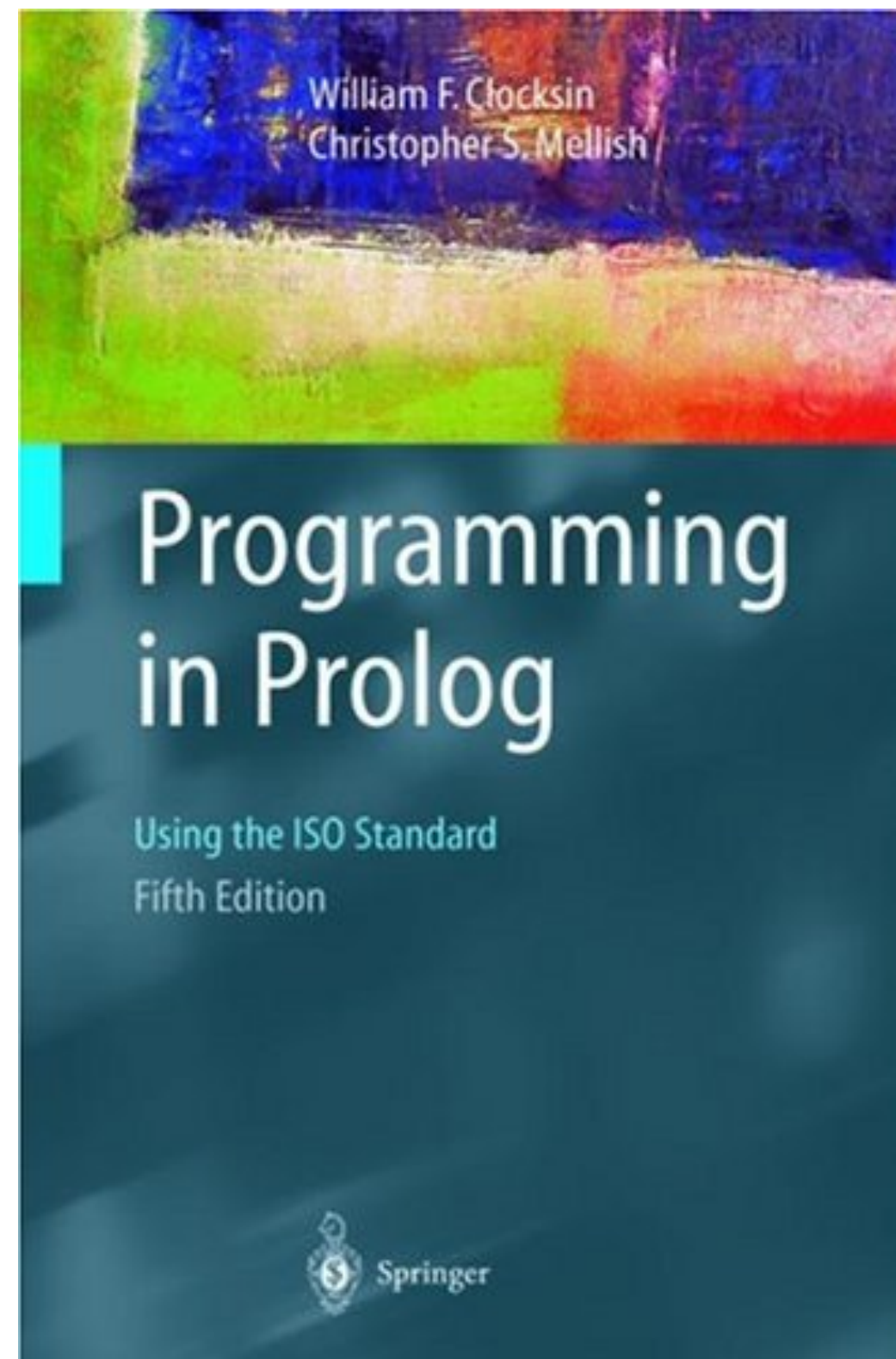








Neprocedurální programování

Prolog 2

Seznamy



Osnova

-  Tvar programu v Prologu
-  Unifikace, algoritmus splňování cíle
-  Rekurze
-  Směr výpočtu
 - příklad: predikát `predek / 2`
-  Nedeterminismus
 - příklad: axiomatizace aritmetiky přirozených čísel
-  Seznamy

Operátory

Procedurální jazyky: $a+b*c-1$

- výrazy s operátory v infixové notaci

Prolog: $adam \backslash= eva$

- syntaktické pozlátko pro $\backslash= (adam, eva)$
- $\backslash=$ je binární funktor
 - » definovaný jako operátor
 - » lze použít infixovou notaci
- `display/1`
 - » standardní predikát, vypíše term v kanonickém tvaru
- `?- display (a+b*c-1)`
`- (+ (a, * (b, c)) , 1)`

Tvar programu v Prologu

Program se skládá z *procedur*

Procedura $\stackrel{\text{def}}{=}$ posloupnost *klauzulí* se stejným hlavním funktorem

Klauzule $\stackrel{\text{def}}{=}$ *pravidlo* nebo *fakt*

Pravidlo $\stackrel{\text{def}}{=}$ *hlava* : – *tělo* .

- **:** – $\stackrel{\text{def}}{=}$ operátor „jestliže“
- **hlava** $\stackrel{\text{def}}{=}$ term (\neq proměnná, číslo)
- **tělo** $\stackrel{\text{def}}{=}$ posloupnost termů (\neq číslo)
 - + logické spojky konjunkce (,) disjunkce (;)
 - + závorky

~~a;b :- c.~~

Tvar programu v Prologu

Fakt $\stackrel{\text{def}}{=}$ pravidlo s prázdným tělem

Direktiva $\stackrel{\text{def}}{=}$ pravidlo s prázdnou hlavou

- `:- consult(demo) .`
 - » nevypisují se hodnoty proměnných
 - » nehledají se alternativní řešení

Komentáře

- na jednom řádku: uvozené `%`
- na více řádcích: mezi `/*` a `*/`

Proměnné

Procedurální jazyky: proměnné

- jsou deklarovány
- mohou být globální, lokální
- změna hodnoty přiřazovacím příkazem

Prolog

- dynamická alokace paměti
 - » garbage collection
- platnost proměnné je omezena na klauzuli, v níž se vyskytuje
- proměnná volná / vázaná
 - » může být vázána na hodnotu při splňování cíle
 - » neúspěch → návrat → odvolání vazby

Termy

Procedurální jazyky: datový typ záznam / struktura

- položky identifikovány jménem

Prolog: složený term (struktura)

- položky identifikovány polohou
- stromová struktura

Unifikace

Základní operace na termech

Dva termy lze unifikovat, pokud

- jsou identické, nebo
- se stanou identickými po substituci vhodné hodnoty proměnným v obou termech

☀ Příklad

- $\text{datum}(D1, M1, 2025) = \text{datum}(D2, \text{unor}, R2)$
 - » např. $D1 = D2 = 24, M1 = \text{unor}, R2 = 2025$
- nejobecnější unifikace
 - » $D1 = D2$
 - » $M1 = \text{unor}$
 - » $R2 = 2025$

Unifikační algoritmus

Unifikaci vyvolá operátor =

- $term1 = term2$ uspěje, pokud oba termy lze unifikovat
- jinak selže

Při úspěchu provede nejobecnější unifikaci

Výsledkem úspěšné unifikace je substituce hodnot za proměnné

Poznámka: $term1 \neq term2$

- uspěje, pokud oba termy **nelze** unifikovat

Termy S a T lze unifikovat, pokud

S a T jsou identické konstanty

S a T jsou proměnné

- výsledkem unifikace je jejich ztotožnění

S je proměnná, T je term různý od proměnné

- výsledkem je substituce termu T za proměnnou S

T je proměnná, S je term různý od proměnné

- výsledkem je substituce termu S za proměnnou T

S a T jsou složené termy, které

- oba mají stejný hlavní funktor a odpovídající argumenty lze unifikovat

V ostatních případech S a T unifikovat nelze.

Unifikace: Příklad

?- $f(X, a(b, c)) = f(d, a(Y, c))$.

$X = d, Y = b$.

?- $f(X, a(b, c)) = f(Y, a(Y, c))$.

$X = Y = b$.

?- $f(c, a(b, c)) = f(Y, a(Y, c))$.

false.

?- $X = f(X)$.

Algoritmus splňování cíle

Unifikační algoritmus + backtracking

- průchod do hloubky s návratem při neúspěchu

Na pořadí záleží

- klauzule i termy jsou zpracovány v pořadí, v němž jsou zapsány
- chronologický backtracking

Platnost proměnných

- platnost proměnné omezena na pravidlo, v němž se vyskytuje
- před použitím pravidla jsou v něm vždy všechny proměnné přejmenovány

Algoritmus splňování cíle

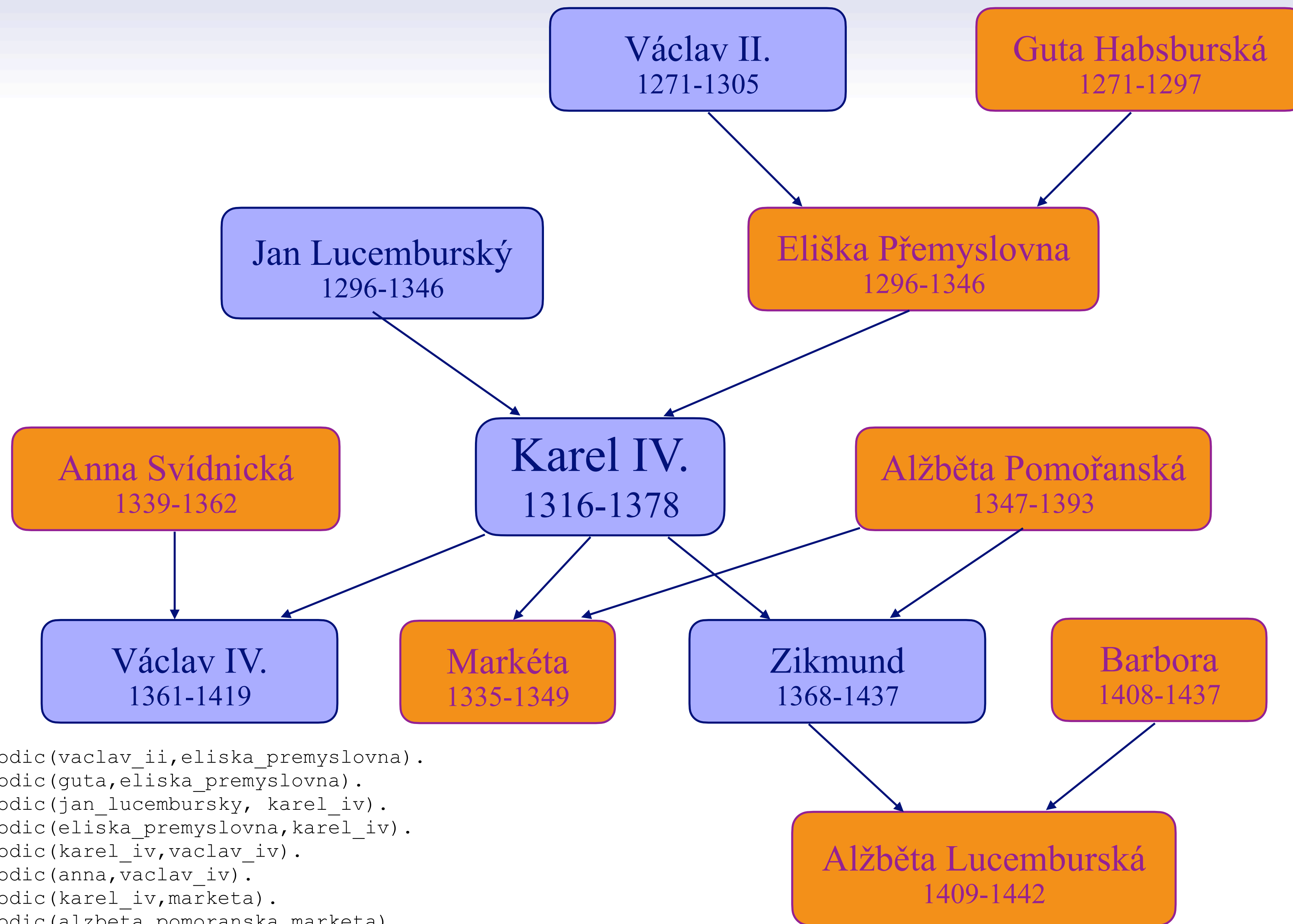
```
def splňování_cíle(program, seznam_cílů) :  
    """ program : seznam klauzulí  
        seznam_cílů : obsahuje cíle, které chceme splnit  
        vrátí : True / False  
    """  
  
    if seznam_cílů je prázdný : return True  
    else : cíl = hlava(seznam_cílů) % první cíl v seznamu  
           další = tělo(seznam_cílů) % seznam ostatních cílů  
           splněno = False  
  
           while not splněno and program obsahuje další klauzuli :  
               nechť další klauzule je tvaru  $H :- T_1, \dots, T_n$ .  
               přejmenuj všechny proměnné v této klauzuli  
               if cíl lze unifikovat s termem  $H$  a výsledkem je substituce  $S$  :  
                   nové_cíle = zřetězení  $T_1, \dots, T_n$  se seznamem další  
                   ve všech prvcích seznamu nové_cíle proved' substituci  $S$   
                   splněno = splňování_cíle(program, nové_cíle)  
  
    return splněno
```

Rekurze

Strukturální rekurze

- řízena strukturou argumentů
- rekurzivní datová struktura
 - 👉 rekurzivní procedura, která s ní pracuje
- **báze**
 - » fakt nebo nerekurzivní pravidlo
- **krok rekurze**
 - » rekurzivní pravidlo

```
predek(X,Y) :- rodic(X,Y).           % báze
predek(X,Z) :- rodic(X,Y), predek(Y,Z). % rekurzivní krok
```

```

rodic(vaclav_ii, eliska_premyslovna).
rodic(guta, eliska_premyslovna).
rodic(jan_lucembursky, karel_iv).
rodic(eliska_premyslovna, karel_iv).
rodic(karel_iv, vaclav_iv).
rodic(anna, vaclav_iv).
rodic(karel_iv, marketa).
rodic(alzbeta_pomoranska, marketa).
rodic(karel_iv, zikmund).
rodic(alzbeta_pomoranska, zikmund).
rodic(zikmund, alzbeta_lucemburska).
rodic(barbora, alzbeta_lucemburska).

```

Směr výpočtu

```
% predek(X,Y) :- X je předkem Y.
```

```
predek(X,Y) :- rodic(X,Y).
```

```
predek(X,Z) :- rodic(X,Y), predek(Y,Z).
```

☞ Které argumenty jsou **vstupní** a které **výstupní**?

- syntaxe to nespecifikuje
- některé argumenty mohou hrát roli vstupu i výstupu
 - » relace “=” je symetrická
- ?- **predek(vaclav_ii, alzbeta_lucemburska).**
- ?- **predek(vaclav_ii, Y).**
- ?- **predek(X, alzbeta_lucemburska).**
- ?- **predek(X, Y).**

Specifikace vstupu a výstupu

Konvence pro specifikaci V/V v komentáři

- + argument je vstupní
 - » při dotazu musí být konkretizován
 - » konstatní term = term bez volných proměnných
- – argument je výstupní
 - » při dotazu nesmí být konkretizován
 - » volná proměnná
- ? argument může být vstupní i výstupní

☀ Příklad genealogický

```
predek(X,Y) :- rodic(X,Y).
```

```
predek(X,Z) :- rodic(X,Y), predek(Y,Z).
```

Jak se vyhodnocují dotazy

- ?- predek(vaclav_ii, alzbeta_lucemburska).
- ?- predek(vaclav_ii, Y).
- ?- predek(X, alzbeta_lucemburska).
- % predek(+Predek, ?Potomek)
- vhodnější název by byl potomek/2

```
% predek1(?Predek, +Potomek)
```

```
predek1(X,Y) :- rodic(X,Y).
```

```
predek1(X,Z) :- rodic(Y,Z), predek1(X,Y).
```

Směr výpočtu : závěr

Poučení

- predikáty v Prologu jsou často invertibilní
» vstup \leftrightarrow výstup \Rightarrow obrácení “směru výpočtu”
- ne vždy jsou oba směry stejně efektivní
- samostatný predikát pro každý směr

☀ Axiomatizace přirozených čísel

```
% prirozene_cislo(?X) :- X je přirozené číslo.  
prirozene_cislo(0).  
prirozene_cislo(s(X)) :- prirozene_cislo(X).
```

```
% mensi(+X,+Y) :- X je ostře menší než Y.  
mensi(0,s(X)) :- prirozene_cislo(X).  
mensi(s(X),s(Y)) :- mensi(X,Y).
```

Procedura `mensi/2` je **deterministická**

- cíl lze unifikovat s hlavou nejvýše jednoho pravidla
- (dotaz `mensi(-X,+Y)` je ovšem nedeterministický)

Nedeterminismus

```
% mensi(+X,+Y):- X je ostře menší než Y.  
mensi(0,s(X)) :- prirodzene_cislo(X).  
mensi(s(X),s(Y)) :- mensi(X,Y).  
  
% alternativní definice  
mensi2(X,s(X)) :- prirodzene_cislo(X).  
mensi2(X,s(Y)) :- mensi2(X,Y).
```

Procedura `mensi2/2` je **nedeterministická**

- `?- mensi2(s(0),s(s(0)))`.
- unifikace s hlavou obou pravidel

Nedeterminismus vs. determinismus

Deterministická procedura

- při neúspěchu není nutný návrat
 - » backtracking je triviální
- snadnější ladění

Nedeterminismus

- mocný nástroj
- někdy ho potřebujeme
 - » za chvíli: generování permutací

Problém: směr výpočtu $\text{mensi}/2$

Definujte predikát $\text{gen_mensi}(-, -)$ tak, aby na dotaz

?- $\text{gen_mensi}(X, Y)$.

byly postupně vygenerovány

- všechny dvojice přirozených čísel X, Y
- takové, že X je ostře menší než Y .

Pozor na to, že predikáty $\text{mensi}/2$ a $\text{mensi}2/2$ takto nefungují !

Seznamy

[] prázdný seznam

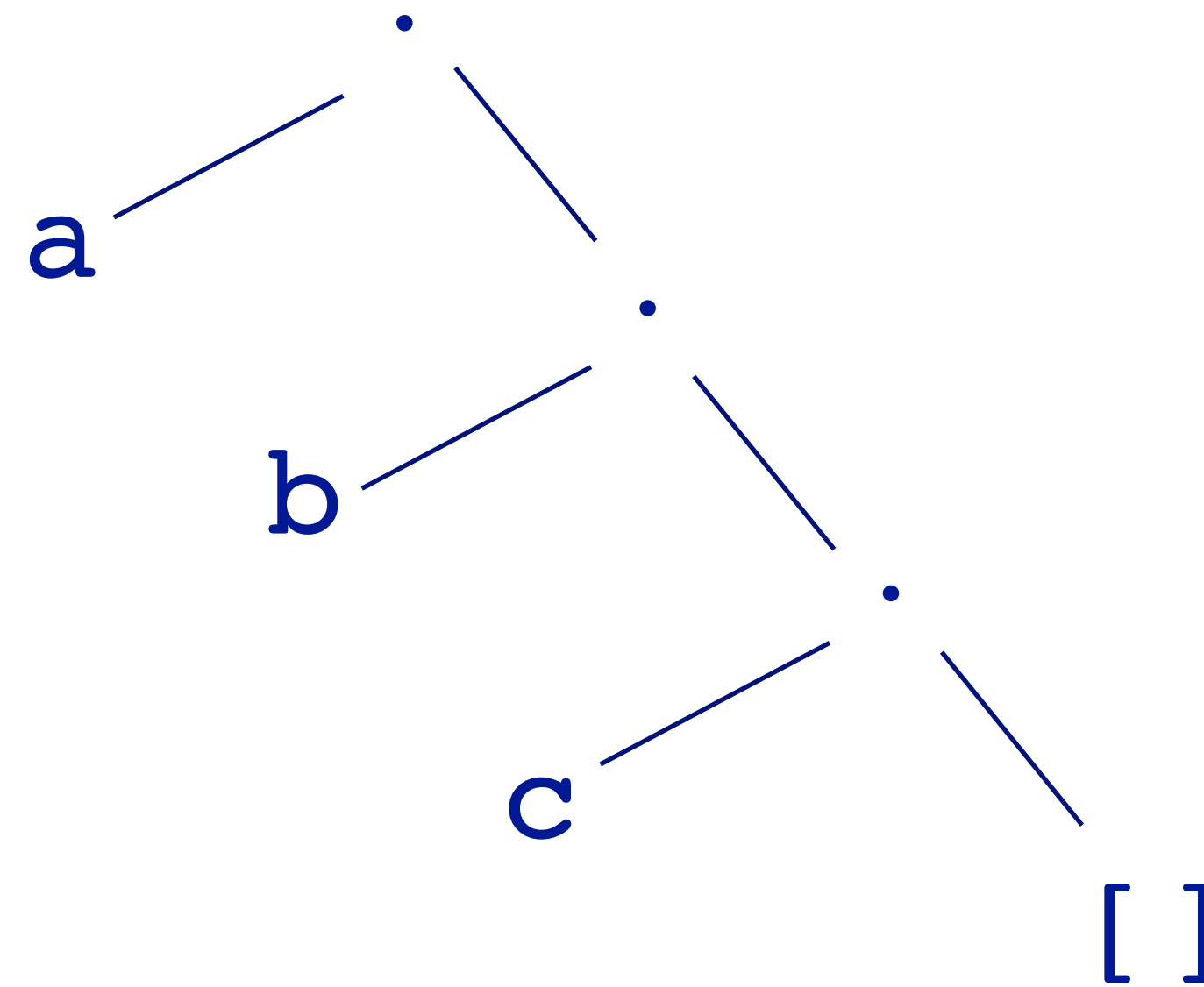
[a , b , c] příklad neprázdného seznamu

Neprázdný seznam

- tečka-dvojice *.(Hlava, Tělo)*
- *Hlava* - první prvek seznamu
- *Tělo* - seznam tvořený zbylými prvky
- navrženo pro jazyk LISP

Seznamy: Příklad

$[a, b, c] = .(a, .(b, .(c, []))) =$



Seznamy: notace

SWI-Prolog verze 6

- `?- display([a,b,c])`
- `.(a,.(b,.(c,[])))`

SWI-Prolog od verze 7

- `./2` má jiné využití
- `?- display([a,b,c])`
- `'[|]'(a,'[|]'(b,'[|]'(c,[])))`

V Prologu pro oddělení hlavy a těla seznamu slouží operátor |

Seznamy: operátor |

.(Hlava, Telo) se v jazyce Prolog zapíše jako *[Hlava | Telo]*

Operátor | má dokonce ještě obecnější význam

- umožňuje oddělit nejen hlavu
- ale i začátek seznamu
- *[Začátek | Tělo]*
- *Začátek* je **výčet** prvků na začátku seznamu oddělených **čárkami**
- *Tělo* je **seznam** zbývajících prvků seznamu

$$[a,b,c]=[a|[b,c]]=[a,b|[c]]=[a,b,c|[]]$$

Seznamy stejné délky

```
% stejne_delky(?Xs,?Ys) :- Xs a Ys jsou  
%                             stejne dlouhe seznamy.  
stejne_delky([], []).  
stejne_delky([_|Xs],[_|Ys]) :- stejne_delky(Xs,Ys).  
• ?- stejne_delky([a,b,c],[X,Y,Z]).  
• ?- stejne_delky(Xs,[a,b,c]).  
• ?- stejne_delky(Xs,Ys).
```

Předdefinován jako standardní predikát

`same_length/2`

Seznamy: predikát prvek/2

```
% prvek(?X,?Xs):- X je prvkem seznamu Xs.
```

```
prvek(X, [X|_]).
```

```
prvek(X, [_|Xs]) :- prvek(X,Xs).
```

- ?- prvek(a, [a,b,c]).
- ?- prvek(X, [a,b,c]).
- ?- prvek(a, Xs).

Předdefinován jako standardní predikát

`member/2`

Vyhledávání v asociativním seznamu

Seznam položek s **klíčem** a **hodnotou**, např.

- `p(klíč, hodnota)`
- `klíč : hodnota`

```
?- ASeznam = [p(nprg031,holan), p(nprg031,pergel),  
p(nprg005,hric), p(nprg005,dvorak), p(nmin112,topfer)],  
member(p(nprg005,Ucitel), ASeznam).
```

```
Ucitel = hric ;
```

```
Ucitel = dvorak
```

```
?- ASeznam = [nprg031 : holan, nprg031 : pergel, nprg005 : hric,  
nprg005 : dvorak, nmin112 : topfer],  
member(nprg005 : Ucitel, ASeznam).
```

Vypuštění prvku ze seznamu

```
% vypust(X,Xs,Ys):- Seznam Ys vznikne vypuštěním  
%                  prvku X ze seznamu Xs.
```

```
vypust(X,[X|Xs],Xs).
```

```
vypust(X,[Y|Ys],[Y|Zs]) :- vypust(X,Ys,Zs).
```

- ?- vypust(a,[a,b,a,c,a],V).

- » vypustí vždy jen jeden výskyt

- » postupně vrátí všechny možnosti

- ?- vypust(a,[b,c],V).

- » není-li vypouštěný prvek v zadaném seznamu, selže

Předdefinován jako standardní predikát

select/3

Problém: další varianty vypouštění

Definujte následující predikáty

- `vypustvse(X,Xs,Ys)`:- Seznam Ys vznikne vypuštěním všech výskytů prvku X ze seznamu Xs, vždy úspěje

Předdefinován jako standardní predikát

`delete(Xs,X,Ys)`

- `vypustvsel(X,Xs,Ys)`:- varianta `vypustvse/3`, není-li X prvkem Xs, selže

Vložení prvku do seznamu

```
vypust (X, [X | Xs], Xs) .
```

```
vypust (X, [Y | Ys], [Y | Zs]) :- vypust (X, Ys, Zs) .
```

- co lze říci o dotazu
- `?- vypust(z, Xs, [a,b,c]) .`

» vloží prvek do seznamu

» postupně na všechna možná místa

```
% vloz(+X,+Xs,?Ys):- Seznam Ys vznikne vložním  
%                       prvku X do seznamu Xs.
```

```
vloz (X, Xs, Ys) :- vypust (X, Ys, Xs) .
```

👉 vestavěný `select/3` lze použít i pro vkládání

První a poslední prvek

První prvek

- hlava seznamu
- přístupný přímo pomocí |

Poslední prvek

`% posledni(+Xs,?X):- X je posledním prvkem Xs.`

`posledni([X],X).`

`posledni([_|Xs],Y) :- posledni(Xs,Y).`

Předdefinován jako standardní predikát

`last/2`

Prostřední prvek

❶ Naivní řešení

- odstrañ první a poslední prvek
- ve zbytku najdi prostřední prvek rekurzivně
- báze pro 1 a 2-prvkové seznamy
- kvadratická časová složitost

❷ Řešení s aritmetikou

- spočítej délku seznamu n
- vrať prvek na pozici $\lceil (n+1)/2 \rceil$ nebo $\lfloor (n+1)/2 \rfloor$

❸ Elegantní řešení

- pomocí dvou signálů v lineárním čase

Prostřední prvek pomocí 2 signálů

```
% prostredni(+Xs,?X):- X je prostředním prvkem Xs.  
prostredni(Xs,X):- prostredni(Xs,Xs,X).  
prostredni([_],[X|_],X).  
prostredni([_,_],[X|_],X).  
prostredni([_,_|Xs],[_|Ys],X) :-  
                                prostredni(Xs,Ys,X).
```

Časová složitost **lineární**

Problém

- který prvek bude vrácen ze seznamu sudé délky?

Zřetězení seznamů

```
% zretez(?Xs,?Ys,?Zs):- Zs je zřetězením seznamů  
%                               Xs a Ys.
```

```
zretez([ ],Ys,Ys).
```

```
zretez([X|Xs],Ys,[X|Zs]):- zretez(Xs,Ys,Zs).
```

Dotazy

- ?- zretez([a,b,c], [d,e], [a,b,c,d,e]).
- ?- zretez([a,b,c], [d,e], Zs).
- ?- zretez(Xs, Ys, [a,b,c,d,e]).

Předdefinován jako standardní predikát

append/3

Využití predikátu zřetězení

`prvek(X,Ys) :- append(_, [X|_], Ys).`

`posledni(X,Ys) :- append(_, [X], Ys).`

 **Problém:** Využijte `append/3` k definici predikátů:

- `prefix(?Xs,+Ys) :-` Xs je předponou seznamu Ys
- `sufix(?Xs,+Ys) :-` Xs je příponou seznamu Ys
- `faktor(?Xs,+Ys) :-` Xs je souvislý podseznam Ys

Otočení seznamu

❶ Naivní řešení v kvadratickém čase

```
% otoc(+Xs,-Ys):- Ys je otočením seznamu Xs.
```

```
otoc([],[]).
```

```
otoc([X|Xs],Zs) :- otoc(Xs,Ys), append(Ys,[X],Zs).
```

❷ Otočení v lineárním čase

```
otocAk(Xs,Ys) :- otocAk(Xs,[],Ys).
```

```
% otocAk(+Xs,+A,-Ys):- Ys je zřetězením otočeného  
% seznamu Xs se seznamem A.
```

```
otocAk([],A,A).
```

```
otocAk([X|Xs],A,Ys) :- otocAk(Xs,[X|A],Ys).
```


Otočení seznamu v lineárním čase

Vlastnosti řešení

- řešíme obecnější problém
- technika akumulátoru
- lineární čas

Problém

- jaká bude odpověď na dotaz `otocAk (-Xs, +Ys)`?

Předdefinován jako standardní predikát `reverse/2`

- funguje oběma směry
- `reverse(+, -)` i `reverse(-, +)`

Permutace

```
% permutace(+Xs,-Ys) :- Seznam Ys je permutací  
%                          seznamu Xs.
```

```
permutace([],[ ]).
```

```
permutace(Xs,[X|Zs]) :- select(X,Xs,Ys),  
                        permutace(Ys,Zs).
```

☀ Idea

- **nedeterministický** výběr prvního prvku permutace
- permutace zbylých prvků rekurzivně

Permutace: alternativní řešení

```
permutace2 ( [ ] , [ ] ) .
```

```
permutace2 ( [ X | Xs ] , Zs ) :- permutace2 ( Xs , Ys ) ,  
                                  select ( X , Zs , Ys ) .
```

☀ Idea

- oddělení hlavy vstupního seznamu
- permutace těla vstupního seznamu
- **nedeterministické** vložení hlavy

Standardní predikát `permutation(?Xs, ?Ys)`