

Matematická logika

Logické programování, PROLOG (10.přednáška)

Marie Duží

marie.duzi@vsb.cz



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání
pro konkurenceschopnost

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Základy (logika) Prologu

- **Metoda** (čistého) **logického programování** je speciálním případem obecné rezoluční metody. Oproti obecné rezoluční metodě splňuje následující omezení:
- Pracuje pouze s Hornovými klauzulemi (které mají ***nanejvýš jeden pozitivní literál***).
- Používá **lineární strategii generování rezolvent** spolu s tzv. **navracením (backtracking)**.

Základy (logika) Prologu

- V logickém programování používáme následující terminologii:
- Zápisy: $P \text{ :- } Q_1, Q_2, \dots, Q_n.$ \longrightarrow podmíněné příkazy (**pravidla**)
(což je ekvivalentní: $\neg Q_1 \vee \neg Q_2 \vee \dots \vee \neg Q_n \vee P$, neboli $(Q_1 \wedge Q_2 \wedge \dots \wedge Q_n) \supset P$)
- Zápis $P.$ \longrightarrow nepodmíněný příkaz (**fakt**)
- Zápisy $?- Q_1, Q_2, \dots, Q_n.$ \longrightarrow cíle (**cílové klauzule, dotazy**)
(což je ekvivalentní: $\neg Q_1 \vee \neg Q_2 \vee \dots \vee \neg Q_n$)
- Zápis \square , YES: \longrightarrow spor (**prázdňá klauzule**)

Základy (logika) Prologu

- **Logický program** je posloupnost příkazů (procedur) podmíněných (tj. pravidel) i nepodmíněných (tj. faktů). Cílová klauzule zadává **otázky**, na které má program nalézt odpovědi.
- *Pozn.:* Pojem příkazu chápeme ve smyslu předchozí poznámky. Logický program je tedy **deklarativní** (*ne imperativní*). Specifikujeme, "**co** se má provést" a **neurčujeme**, "**jak** se to má provést".

Cestu, jak odpovědět na dotazy, najde překladač, tj. určí, co vyplývá ze zadané báze znalostí, a jaké hodnoty je nutno substituovat unifikací za proměnné.

Základy (logika) Prologu

Omezení na Hornovy klauzule však může někdy činit potíže. Viz příklad – hádanka Zkuste naformulovat tuto hádanku v Prologu!

Problém: Všichni členové klubu jsou lyžaři **nebo** horolezci:

$$\forall x [SC(x) \supset (SKI(x) \vee CLIMB(x))] \Leftrightarrow \forall x [\neg SC(x) \vee \mathbf{SKI(x)} \vee \mathbf{CLIMB(x)}]$$

Příklad, hádanka – nanejvýš 1 pozitivní literál!

Tom, Peter and John are members of a sport club. Every member of the club is **a skier or a climber**. No climber likes raining. All skiers like snow. Peter does not like what Tom likes, and does like what Tom does not like. Tom likes snow and raining.

- *Question:* Is there in the club a sportsman who is a climber but not a skier?

- *Solution: Knowledge base (+ query 11):*

1. $SC(t)$
2. $SC(p)$
3. $SC(j)$
4. $\forall x [SC(x) \supset (\mathbf{SKI(x) \vee CLIMB(x)})]$ **problém:** 2 pozitivní literály.
5. $\forall x [CLIMB(x) \supset \neg LIKE(x,r)]$
6. $\forall x [SKI(x) \supset LIKE(x,s)]$
6. $\forall x [LIKE(t,x) \supset \neg LIKE(p,x)]$
6. $\forall x [\neg LIKE(t,x) \supset LIKE(p,x)]$
9. $LIKE(t,s)$
10. $LIKE(t,r)$
11. **? $\exists x [SC(x) \wedge CLIMB(x) \wedge \neg SKI(x)]$**

Základy (logika) Prologu

Shrnutí omezení: nanejvýš jeden pozitivní literál, nemůžeme vyjádřit přímo negativní fakta.

Negace = neúspěch při odvozování !

- V podstatě jsou v Prologu pouze dvě možnosti, *jak ovlivnit provádění programu:*
 - *Pořadí* klauzulí a literálů
 - Příkaz ! (*řez*, cut)

Příklad

- Všichni studenti jsou mladší než Petrova matka. Karel a Mirka jsou studenti. *Kdo je mladší než Petrova matka?*

- **Zápis v PL1:**

$\forall x [St(x) \supset Ml(x, f(a))]$
St(b)
St(c)
 $\Rightarrow \exists \bar{y} \bar{Ml}(\bar{y}, \bar{f}(a)) \text{ ???}$

- **Zápis v Prologu:**

mladsi(X, matka(petr)):- student(X). pravidlo
student(karel). fakt
student(mirka). fakt
?- mladsi(Y, matka(petr)). dotaz

Základy (logika) Prologu:

Příklad

Řešení rezolucí v PL1 ($a = \text{Petr}$, $b = \text{Karel}$, $c = \text{Mirka}$):

1. $\neg \text{St}(x) \vee \text{Ml}(x, f(a))$
2. $\text{St}(b)$
3. $\text{St}(c)$
4. $(\forall y) \neg \text{Ml}(y, f(a))$
5. $\blacksquare \text{Ml}(b, f(a))$
- 6.
7. $\blacksquare \text{Ml}(c, f(a))$
- 8.

negovaný závěr
rezoluce 1.,2.; x / b
spor - YES (y / b)
rezoluce 1.,3.; x / c
spor - YES (y / c)

Příklad

- Řešení v Prologu:

```
mladsi(X, matka(petr)):- student(X)   pravidlo
student(karel).                       fakt
student(mirka).                       fakt
?- mladsi(Y, matka(petr)).            dotaz
```

Překladač provádí unifikaci a rezoluci, **lineární strategie řízená cílem**:

- 1) Cíl `?- mladsi(Y, matka(petr))` **unifikuje** s `mladsi(X, matka(petr))`, $Y=X$;
- 2) Generuje nový cíl: `?- student(Y)`
- 3) Unifikuje tento cíl s 2. faktem v databázi: úspěch při $Y=karel$
- 4) Vydá odpověď : **YES, Y = karel ;**

Můžeme zadat ; to znamená, že se ptáme „a kdo ještě?“ Vyvolá tzv. **backtracking**, tj. proces navracení. Vrátí se k poslednímu cíli a pokouší se splnit jej znovu: `?- student(Y)`.
Teď již nemůže použít 2. klausuli (pamatuje si místo, které již bylo užito), ale může použít 3. klausuli:

- 5) Vydá odpověď: **YES, Y = mirka;**
NO

Příklad

mladsi(X, matka(petr)):- student(X). pravidlo

student(karel). fakt

student(mirka). fakt

mladsi(X, matka(petr)):- dite(X, matka(petr)).
 pravidlo

dite(X,Y):-Y=matka(X). pravidlo

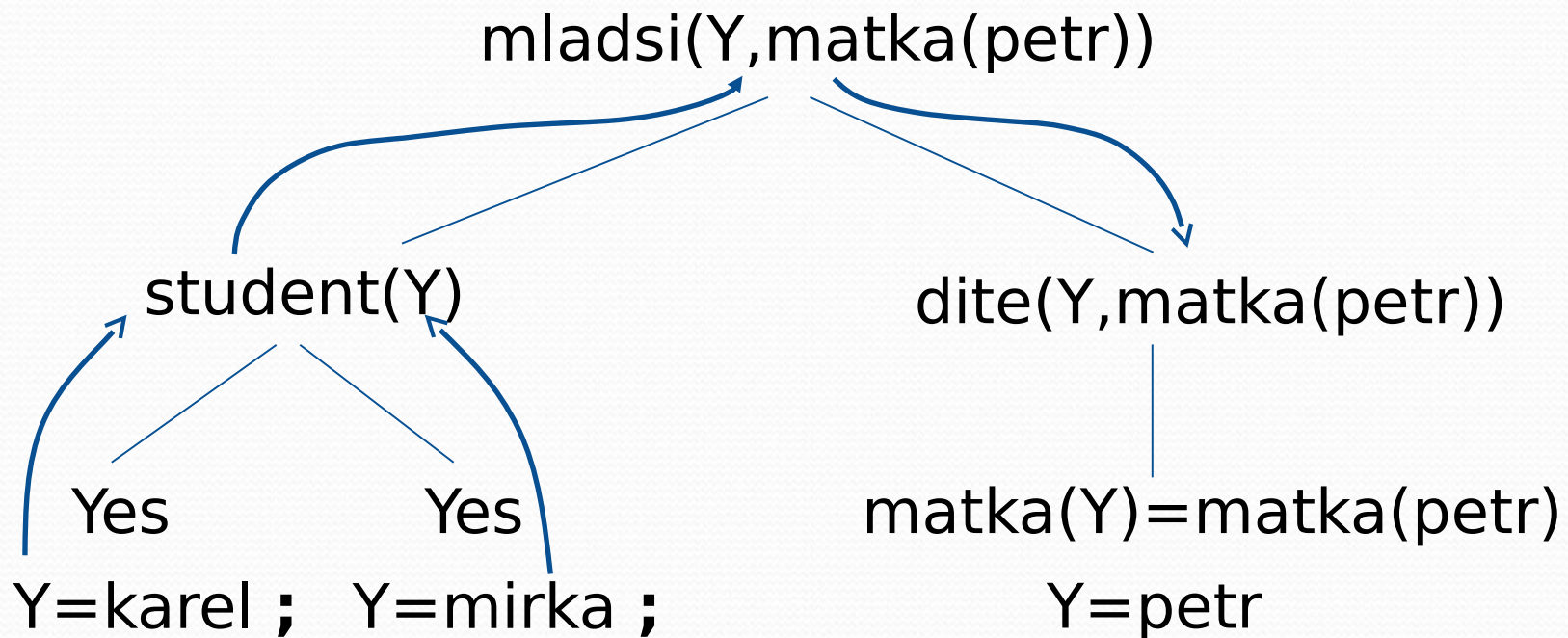
?- mladsi(Y, matka(petr)). dotaz

a) Vydá odpověď: **YES, Y = Karel ;**

b) Vydá odpověď: **YES, Y = Mirka ;**

Jak to bude fungovat dál? Měla by přijít odpověď **Y = Petr** (Petr je mladší než jeho matka).

Navracení (backtracking)



...

Základy Prologu

mladsi(X, matka(petr)):- student(X). pravidlo

student(karel). fakt

student(mirka). fakt

mladsi(X, matka(petr)):- dite(X, matka(petr)). pravidlo

dite(X,Y):-Y=matka(X). pravidlo

?- mladsi(Y, matka(petr)). dotaz

- a) Vydá odpověď : **YES, Y = Karel ;**
- b) Vydá odpověď : **YES, Y = Mirka ;**
- c) Vydá odpověď : **Yes, Y = Petr.** (Petr je mladší než jeho matka.)

Položíme další dotaz: ?- student(petr).

Odpověď: **NO**

To ovšem **neznamená**, že Petr ve skutečnosti není student, nebo že neexistují jiní studenti. Pouze není uveden v dané bázi znalostí:
předpoklad uzavřeného světa (platí jen to, co je v bázi znalostí).

Negace jako neúspěch při odvozování. Nemůžeme přímo zadat negativní fakta.

Prolog - negace

- Nemůžeme zadat přímo, že např. Marie není student. Můžeme použít predikát **not**:

```
Not(student(marie)):-  
call(student(marie)), !, fail.  
not(student(marie)).
```

- not(student(marie)).
 - Pokud z báze znalostí vyplývá, že student(marie), pak selže.
 - Pokud call(student(marie)) selže, pak uspěje.
 - cut ! – řez: odřízne možnost navracení: „nezkoušej to znovu“.
- Negace jako neúspěch při odvozování.

Příklad: Euklidův algoritmus

1. `nsd(X,X,X).`
2. `nsd(X,Y,Z):- X>Y, nsd(X-Y,Y,Z).`
3. `nsd(X,Y,Z):- Y>X, nsd(X,Y-X,Z).`

Pozn.: V běžných implementacích Prologu je vestavěna základní aritmetika.

4. `?- nsd(4,6,Z).`

Dotaz: hledáme největšího společného dělitele čísel 4 a 6.

Výpočet:

5. `?- 4>6, nsd(4-6,6,Z)` rezoluce: 4.,2.
4. `?- nsd(4,6,Z)` backtracking
5. `?- 6>4, nsd(4,6-4,Z)` rezoluce: 4,3.
6. `?- nsd(4,2,Z)` „Výpočet“ klausule 5., fakt "6>4"
7. `?- 4>2, nsd(4-2,2,Z)` rezoluce: 6.,2.
8. `?- nsd(2,2,Z)` „Výpočet“ klausule 7., fakt "4>2"
9. `ano` rezoluce: 8.,1. **výsledek: Z = 2**

Příklad: generování přirozených čísel

1. $\text{nat}(0)$. 0 je přirozené číslo
2. $\text{nat}(s(X)):-\text{nat}(X)$. následník přirozeného čísla je přirozené číslo
3. $?-\text{nat}(s(X))$. jaká jsou všechna přirozená čísla ?

Výpočet:

4. $?-\text{nat}(X)$ rezoluce: 3.,2.
5. YES, $X = 0$; neboť dotaz 3 je splněn pro $X=0$
3. $?-\text{nat}(s(X))$ backtracking
4. $?-\text{nat}(X)$ rezoluce: 3.,5.
6. YES, $X = s(0)$; neboť otázka 3. je splněna pro $X=s(0)$
7. YES, $X = s(s(0))$

.....

Pozn.: srovnej Přednáška 8 - příklad na matematickou indukci.

Lineární strategie

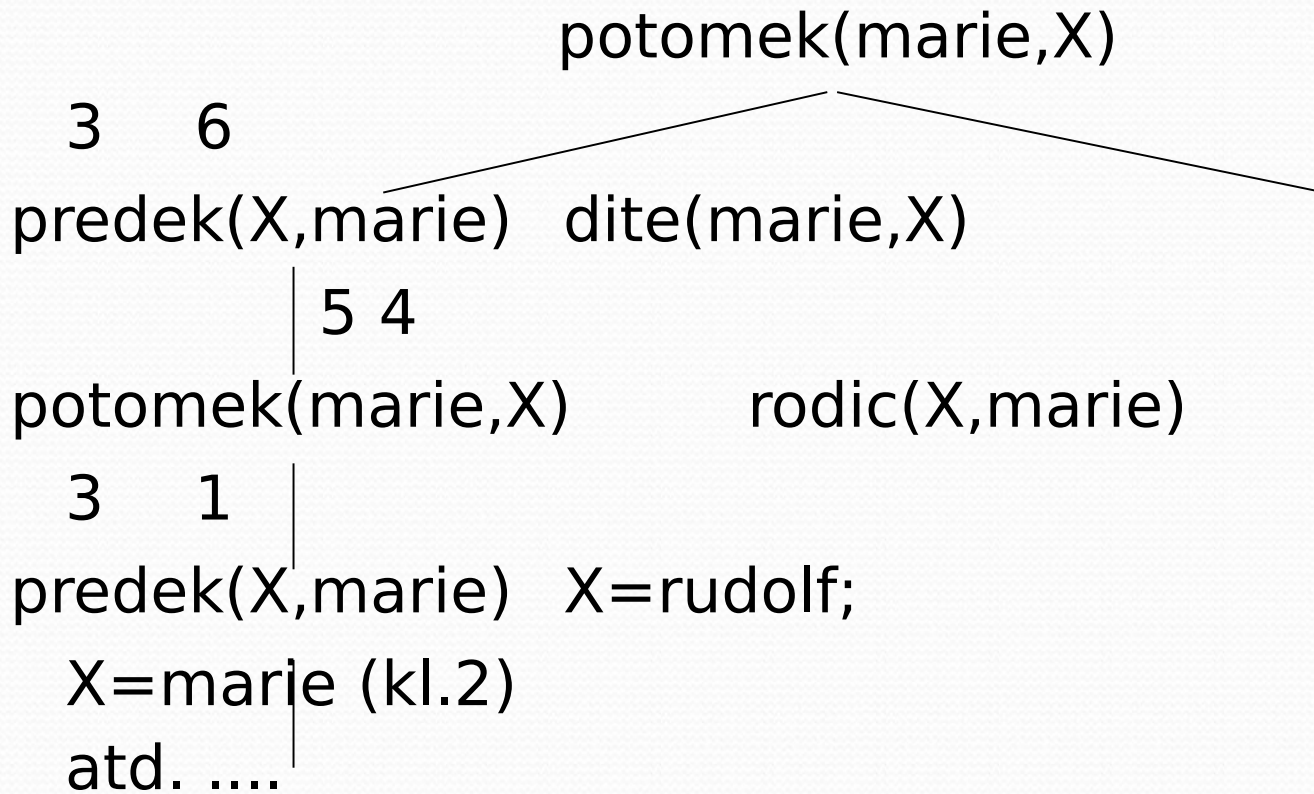
- Pořadí klauzulí může ovlivnit provádění programu („jak?“)

Program A:

- 1.rodic(rudolf, marie).
- 2.rodic(kveta,marie).
- 3.potomek(X,Y):- predek(Y,X).
- 4.dite(X,Y):- rodic(Y,X).
- 5.predek(X,Y):- potomek(Y,X).
- 6.potomek(X,Y):- dite(X,Y).
- 7.potomek(X,Y):- dite(X,Z), potomek(Z,Y).
- 8.?- potomek(marie,X).

Pozn. Přejmenování proměnných provádí rovněž Prolog sám.

Strom výpočtu



Náš program nevydá odpověď, ačkoliv řešení existuje; **první pomoc**: dej (nedořešené) klauzule 3, 5 dozadu, a odřízni je pomocí „cut !“ dořešíš později

První pomoc (opatrně):

cut!
Program B:

1. `rodic(rudolf, marie).`
2. `rodic(kveta,marie).`
3. `potomek(X,Y):- dite(X,Y).`
4. `potomek(X,Y):- dite(X,Z), potomek(Z,Y).`
5. `dite(X,Y):- rodic(Y,X).`
6. `potomek(X,Y):- !, predek(Y,X).`
7. `predek(X,Y):- !, potomek(Y,X).`
8. `?- potomek(marie,X).`

Odřízneme „nedodělané větve“, tj. klauzule 6, 7 a umístíme je nakonec.

Pořadí klauzulí a literálů – „jak“?

1. `rodic(rudolf, marie).`
2. `rodic(kveta,marie).`
3. `potomek(X,Y):- dite(X,Y).`
4. `potomek(X,Y):- potomek(X,Z), dite(Z,Y).`
5. `dite(X,Y):- rodic(Y,X).`
6. `?- potomek(marie,W).`

Výpočet: `?- dite(marie,W), ?-rodic(W,marie), W=rudolf;`
`W=kveta; ?????`

Po druhém středníku se zacyklí:

`?-rodic(W,marie), ?- dite(marie,W), ?- potomek(marie,W),`
`?-potomek(marie,Z), ?- dite(Z',Z), ?- potomek(marie,Z'), ?-`
`dite(Z,Z'),`
`?- dite(Z',Z''), ...`

Stačí zde přehodit literály ve 4. klauzuli.

Po přehození, správně:

1. `rodic(rudolf, marie).`
2. `rodic(kveta,marie).`
3. `potomek(X,Y):- dite(X,Y).`
4. `potomek(X,Y):- dite(X,Z), potomek(Z,Y).`
5. `dite(X,Y):- rodic(Y,X).`
6. `?- potomek(marie,W).`

Pomocná zásada: v programu uvádějte **nejprve fakta, pak pravidla.**

V rekurzivních pravidlech **rekurzivní predikát až nakonec.**

Rekurzivní pravidla (obdoba cyklu v imperativních programovacích jazycích): predikát konsekventu (hlava pravidla) se opakuje v antecedentu (tělo pravidla)

Typický příklad: rekurze

Výpočet funkce **faktoriál**: $x! = x \cdot x - 1 \cdot \dots \cdot 1$, $0! = 1$

```
fact(0, Factorial, Factorial).  
fact(Pocet, Pomocna, Factorial) :-  
    P1 is Pocet - 1,  
    I1 is Pomocna * Počet,  
    fact(P1, I1, Factorial).
```

?- fact(3,1,X).

Výpočet:

```
fact(3,1,Factorial)  
fact(2,3,Factorial)  
fact(1,6,Factorial)  
fact(0,6,Factorial)  
YES, Factorial = 6
```

pomocná proměnná = 1

Lineární strategie řízená cílem + backtracking (navrácení) - shrnutí

- Interpret Prologu prohledává strom výpočtu programu **zleva doprava a do hloubky** (prohledávání do šířky by byla strategie korektní a úplná, ale vyžadovala by při implementaci více zásobníků).
- Tato lineární strategie není úplná, nemusí vydat řešení, i když existuje, pokud program uvízne v nekonečné větvi, která pořadím klauzulí předchází větvi se správným řešením.

Algoritmus (lineární strategie řízená cílem)

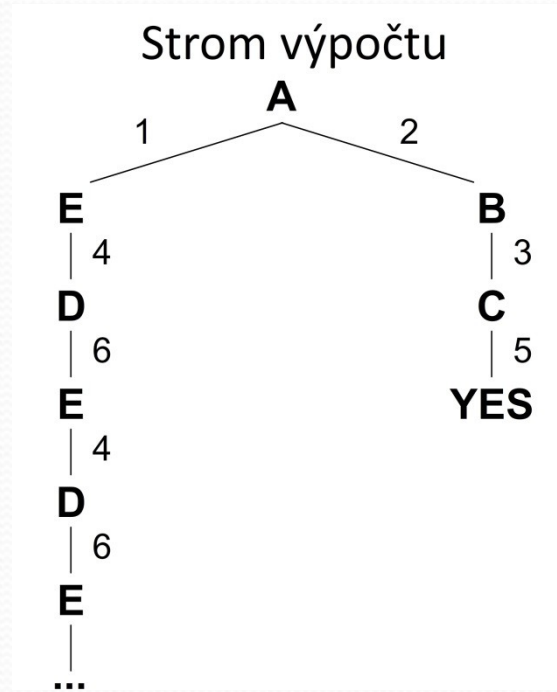
interpretace logického programu

1. Za aktuální cílovou klauzuli vezmi výchozí cílovou klauzuli (dotaz).
2. Je-li aktuální cílová klauzule prázdná, ukonči výpočet s odpovědí "ano" na otázku položenou výchozí cílovou klauzulí. (Byly-li ve výchozí cílové klauzuli volné proměnné, pak poslední substituce termů za tyto proměnné je řešením - součástí odpovědi.) Není-li aktuální cílová klauzule prázdná, přejdi k bodu (3).
3. Vezmi nejlevější cíl v aktuální cílové klauzuli a hledej v programu příkaz se stejným jménem, který dosud nebyl s tímto cílem konfrontován (neúspěšně). Při hledání tohoto cíle postupuj v programu shora dolů (podle pořadí příkazů). Nenalezneš-li takový příkaz, ukonči výpočet s odpovědí "ne" na otázku položenou aktuální cílovou klauzulí. Nalezneš-li, přejdi k bodu (4).
4. Pokus se unifikovat hlavu vybraného cíle s hlavou nalezeného stejnojmenného příkazu. Jestliže unifikace neexistuje, vrať se k bodu (3). Jestliže existuje, vezmi za novou aktuální cílovou klauzuli rezolventu dosavadní cílové klauzule s tělem nalezeného příkazu (při užití nejobecnější unifikace hlavy cíle a hlavy příkazu). Přejdi k bodu (2).

Lineární strategie řízená cílem + backtracking (navracení) - příklad neúplnost

Program:

1. A:- E.
2. A:- B.
3. B:- C.
4. E:- D.
5. C.
6. D:- E.
7. ?- A.



Procedura A má dvě možnosti (větve) výpočtu.

První pomoc: přehodit větve, tj. klauzule 1 a 2.
„Druhá pomoc“ - odříznout nekonečnou větev 1 pomocí Cut !

Klauzule řez: ! – obdoba „go to“

- Kontroluje a omezuje navracení.
- Odřízne „nepotřebné větve“.
- Tento cíl je splněn jen jednou, při pokusu o návrat na cíl ! vrací program až na cíl, který předcházel proceduře, která řez obsahuje (**procedura je množina pravidel a faktů se stejnou hlavou** = konsekvent).
- **Červený** řez - změní deklarativní sémantiku programu (**špatná programátorská praxe**).
- **Zelený** řez - nemění deklarativní sémantiku, pouze (pokud možno) **zefektivní program**:
 - a) Realizace „if, then, else“ (vylučující se nebo)
 - b) Zpracování výjimek (chyb)
 - c) Omezení prohledávání velkých databází (po určitém počtu průchodů a úspěšném nalezení)

Realizace „if, then, else“, cyklu pomocí „fail“

- **Opakovaně prováděj:**
Je-li teplota vysoká (více než 30%), **pak** vypni topení, **jinak je-li** teplota nízká (méně než 15%), **pak** zapni topení, **jinak** nedělej nic.

termostat(Akce) :-

```
    teplota(X),  
    akce(X,Akce),  
    write('Proved:', Akce), nl,  
    fail.           % selže, tedy vyvolá se navrácení  
(backtraching).
```

akce(X,'zapni topeni') :- X < 15, !.

akce(X,'vypni topeni') :- X > 30, !.

akce(_, 'nedelej nic').

?- termostat(X).

Realizace „if, then, else“, cyklu pomocí „fail“

```
termostat(Akce) :-  
    teplota(X),  
    akce(X,Akce),  
    write('Proved:', Akce), nl,  
    fail. % selže, tedy vyvolá se navrácení (backtraching).
```

```
akce(X,'zapni topeni') :- X < 15, !.
```

```
akce(X,'vypni topeni') :- X > 30, !.
```

```
akce(_, 'nedelej nic').
```

Podtržení `_` je **anonymní proměnná**, nepotřebuji její hodnotu.

?- termostat(X).

Výpočet, možné výpisy: zapni topeni, vypni topeni, nedelej nic.

Kdybychom nezařadili řez za každý test, vypisovalo by se vždy: zapni topeni, nedelej nic, vypni topeni, nedelej nic, nedelej nic, ...

Realizace „if, then, else“, řešení bez řezu

akce(X,'zapni topeni') :- X < 15.

akce(X,'vypni topeni') :- X > 30.

akce(X,'nedelej nic') :- not(X < 15), not(X > 30).

- Méně efektivní řešení, zbytečně znovu testujeme na platnost obou (již testovaných) podmínek.

Zpracování výjimek, chyb

```
test(X) :- X = chyba, !, fail.
```

```
test(X) :- write('platne:', X).
```

- Pokud dojde k chybě, pak řez způsobí průchod na fail, test selže, ale návrat na druhou klauzuli (pokus o znovu-splnění cíle test(X)) se již neuskuteční, vrátí se až na klauzuli předcházející test.
- Ekvivalentní, ale *mnohem přehlednější program:*

```
test(X) :- not(X = chyba), write('platne:', X).
```

Datová struktura Seznam

- „potenciálně nekonečná uspořádaná n -tice“.
- Např. [ryba, želva, krab, chobotnice, ...]
- Zadání tvaru [Hlava|Telo], kde **Hlava je prvek seznamu** a **Tělo je opět seznam**.
- Prázdný seznam [].
 - Procedura `member` (většinou je vestavěna) testuje zda první argument je prvek druhého argumentu (seznamu)

`member(X,[X|_]).` je prvkem, pokud = 1.
`member(X,[_|Y]) :- member(X,Y).` jinak testuj tělo

- Procedura `append` (spojuje dva seznamy)

`append([],L,L).`
`append([H,T],L,[H,T1]) :- append(T,L,T1).`