IB015 Neimperativní programování

Neimperativní programování v Prologu (relace místo funkcí)

Jiří Barnat

Trocha filozofie ...

Pozorování

- Podstata neimperativního programování je v tom, že popisuje (deklaruje) vztahy mezi objekty světa. Počítač se používá k tomu, aby na základě popsaných vztahů mechanicky vypočítal výsledek (dedukoval důsledek vztahů).

Haskell

- K popisu vztahů mezi objekty se používají **funkce**.
- Funkce má pro dané vstupy nejvýše jeden výstup.

Zobecnění na relace

- Výpočetní paradigma, kde se pro popis vztahů mezi objekty použijí relace, tj. k daným vstupům existuje více výstupů.

Logické výpočetní paradigma

Logické výpočetní paradigma

- program = databáze faktů a pravidel + cíl.
- výpočet = dokazování cíle metodou SLD rezoluce.
- výsledek = pravda/nepravda a hodnoty volných proměnných, pro které je cíl dokazatelný.
- **Prolog** – jediný programovací jazyk.

SWI-Prolog

- Relativně úplná implementace Prologu.
- Freeware.
- `http://swi-prolog.org`

Úvodní intuitivní příklady

Interaktivní uživatelské prostředí Prologu

SWI-Prolog

- Interpret spuštěn příkazem `swipl`.

Textové interaktivní prostředí

- Standardní výzva: `?-`
- Veškeré povely uživatele musí být zakončeny tečkou.

Základní povely

- Ukončení prostředí: `halt.` (Ctrl+D)
- Nápověda: `help.`
- Načtení souboru `jmeno.pl`: `consult(jmeno).`
- Též alternativně: `[jmeno].`

  `consult('jmeno.pl').`

Notace databáze fakt

Struktura jednoduchých příkladů

- Databáze fakt a pravidel uvedena v externím souboru.
- Cíle zadávány skrze interpret.
- Přípona souboru: `.pl`

Notace fakt

- Fakta začínají malým písmenem a končí tečkou.
- Fakta jsou konstanty (nulární relace) a n-ární relace.
- Počet parametrů udáván u jména za lomítkem: `jmeno/N`.

Příklady

- `tohleJeFakt.`
- `tohleTaky(parametr1,parametr2,...,parametrN).`
- `fakt /* a /* zanoreny */ komentar */ .`

Jednoduché dotazy na fakta a relace

Databáze fakt

- ```
  je_teplo.
  neprsi.
  kamaradi(vincenc,kvido). /* Znají se od mateřské školy. */
  kamaradi(vincenc,ferenc). /* Poznali se na pískovišti. */
  ```

Dotazy na databázi

- ```
  ?- je_teplo.
  true.
  ```
- ```
  ?- prsi.
  ERROR: Undefined prsi/0.
  ```
- ```
  ?- kamaradi(vincenc,kvido).
  true.
  ```
- ```
  ?- kamaradi(ferenc,vincenc).
  false.
  ```
  ```

Dotazy s využitím proměnných

Proměnné

- Jména začínají velkým písmenem nebo podtržítkem.
- Je možné je (mimojiné) využít v dotazech.
- Interpreter se pokusí najít vyhovující přiřazení.

Dotazy s využitím proměnných

- ```
  ?- kamaradi(vincenc,X).
  X = kvido ;
  X = ferenc.
  ```
- ```
  ?- kamaradi(X,Y).
  X = vincenc,
  Y = kvido ;
  X = vincenc,
  Y = ferenc.
  ```

Interakce s uživatelem

Odpověď interpretru zakončená tečkou

- Indikuje, že nejsou další možnosti.

Odpověď interpretru nezakončená tečkou

- Výzva pro uživatele, zda chce hledání možných řešení ukončit (uživatel vloží tečku), nebo zda si přeje, aby bylo hledáno další řešení (uživatel vloží středník).

Porovnejte

- ```
  ?- kamaradi(vincenc,X).
  X = kvido ; /* uživatel vložil středník */
  X = ferenc.
  ```
- ```
  ?- kamaradi(vincenc,X).
  X = kvido . /* uživatel vložil tečku */
  ```

Databáze s pravidly

Pravidla v databázi

- Zápis: `clovek(X) :- zena(X).`
- Význam: Pokud platí `zena(X)`, pak platí `clovek(X)`.

Disjunkce

- Zápis: `clovek(X) :- zena(X); muz(X).`
- Alternativní zápis:

```
clovek(X) :- zena(X).
clovek(X) :- muz(X).
```

- Význam: $(zena(X) \vee muz(X)) \implies clovek(X)$.

Konjunkce

- Zápis: `unikat(X) :- zena(X), muz(X).`
- Význam: $(zena(X) \wedge muz(X)) \implies unikat(X)$.

Dotazy na databáze s pravidly

Databáze:

- ```
  clovek(X) :- zena(X).
  zena(bozena_nemcova).
  ```

Příklady dotazů

- ```
  ?-zena(bozena_nemcova).
  true.
  ```
- ```
  ?-clovek(bozena_nemcova).
  true.
  ```
- ```
  ?-zena(jirik).
  false.
  ```
- ```
  ?- clovek(X).
  X = bozena_nemcova.
  ```
  ```

Dotazy na databáze s pravidly – lokalita proměnných

Rozsah platnosti proměnných

- Použití proměnné je lokalizováno na dané pravidlo.

Příklad

- ```
  clovek(X) :- zena(X); muz(X).
  unikat(X) :- zena(X), muz(X).
  zena(bozena_nemcova).
  zena(jara_cimrman).
  muz(jara_cimrman).
  ```

- ```
  ?- clovek(X).
  X = bozena_nemcova ;
  X = jara_cimrman;
  X = jara_cimrman.
  ```

  ```
  ?- unikat(X).
  X = jara_cimrman.
  ```

Termy – Základní stavební kameny

Syntax Prologu

Pozorování

- Fakta, pravidla a dotazy jsou tvořeny z termů.

Termy

- Atomy.
- Čísla.
- Proměnné.
- Strukturované termy.

Proměnné

- Proměnné se zapisují s velkým počátečním písmenem.
- Hodnotou uloženou v proměnné může být libovolný term.
- **Prolog není striktně typovaný.**

Objekty Prologu – Atomy

Atomy

- Řetězce začínající malým písmenem, obsahující písmena číslice a znak podtržítko.
- Libovolné řetězce uzavřené v jednoduchých uvozovkách.

Příklady:

- Atomy: `pepa, 'pepa', 'Pepa', '2'.`
- Neatomy: `Pepa, 2, ja a on, holmes&watson.`

Test na bytí atomem

- `atom/1` – Pravda, pokud parametr je nestrukturovaným atom.

Objekty Prologu – Čísla

Čísla

- Celá i desetinná čísla, používá se desetinná tečka.

```
?- A is 2.5 * 1.3.
A = 3.25.
```

- Porovnání s aritmetickým vyhodnocením pomocí `=:=`.

```
?- 4 =:= 3+1.
true.
```

```
?- 4 == 3+1.
false.
```

```
?- 4 = 3+1.
false.
```

- Aritmetické vyhodnocení a přiřazení pomocí `is`.

```
?- A is 2*3.
A = 6.
```

```
?- A == 2*3.
false.
```

```
?- A = 2*3.
A = 2*3.
```

Testy na bytí číslem

- `number/1` – Pravda, pokud je parametr číslo.
- `float/1` – Pravda, pokud je parametr desetinné číslo.
- `=\=/2` – Aritmetická neekvivalence.

Objekty Prologu – Strukturované termy

Strukturované termy

- Funktor (název relace) následovaný sekvencí argumentů.
- Pro funktor platí stejná syntaktická omezení jako pro atomy.
- Argumenty se uvádějí v závorkách, oddělené čárkou.
- Mezi funktorem a seznamem argumentů nesmí být mezera.
- Argumentem může být libovolný term.
- Rekurze je možná.

Arita

- Počet argumentů strukturovaného termu.
- Identifikace strukturovaného termu: `funktor/N`.
- Stejný funktor s jinou aritou označuje jiný term.
- Je možné současně definovat termy `term/2` i `term/3`.

Unifikace v Prologu

Unifikace

Definice

- Dva termy jsou unifikovatelné, pokud jsou identické, anebo je možné zvolit hodnoty proměnných použitých v unifikovaných termech tak, aby po dosazení těchto hodnot byly termy identické.

Operátor =/2

- Realizuje unifikaci v Prologu.
- Lze zapisovat infixově.
- Binární operátor ne-unifikace: `\=`, tj. ve standardní notaci `\=/2`.

Výsledek unifikace

- Ano/Ne
- Unifikační přiřazení do proměnných (substituce).

Unifikace – příklady

Bez unifikačního přiřazení

- ```
  ?- =(slovo,slovo).
  true.
  ```

S unifikačním přiřazením

- ```
  ?- =(slovo,X).
  X = slovo.
  ```
- ```
  ?- a(A,[ble,ble]) = a(b(c(d)),B).
  A = b(c(d)),
  B = [ble, ble].
  ```
  ```

Algoritmus unifikace

1) Pokud jsou `term1` a `term2` konstanty (atomy, čísla), pak se unifikují, jestliže jsou tyto termy shodné.
2) Pokud je `term1` proměnná a `term2` je libovolný term, pak se unifikují a proměnná `term1` je instanciována hodnotou `term2`. Podobně, pokud je `term2` proměnná a `term1` je libovolný term, pak se unifikují a proměnná `term2` je instaciována hodnotou `term1`.
3) Pokud jsou `term1` a `term2` strukturované termy tak se unifikují, pouze pokud mají stejný funktor a aritu, všechny korespondující páry argumentů se unifikují, a všechny instanciace proměnných z vnořených unifikací jsou kompatibilní.
4) Nic jiného.

Příklady unifikace

- `?- snida(karel,livance) = snida(Kdo,Co).`

  `Kdo = karel,`

  `Co = livance.`

- `?- snida(karel,Jidlo) = snida(Osoba,marmelada).`

  `Jidlo = marmelada,`

  `Osoba = karel.`

- `?- cdcko(29,beatles,yellow_submarine) = cdcko(A,B,help).`

  `false.`

- `?- fce(X,val) = fce(val,X).`

  `X = val.`

- `?- partneri(eva,X) = partneri(X,vasek).`

  `false.`

- `?- fce(X,Y) = fce(Z,Z).`

  `X = Y, Y = Z.`

Unifikace rekurzivních výrazů

Příklad

- Uvažme následující dotaz na Prolog:

```
?- X = otec(X).
```

- Jsou unifikovatelné termy, kde jeden term je proměnná a přitom je vlastním podvýrazem druhého termu?

Nekorektnost algoritmu

- Podle definice ne, neboť neexistuje hodnota této proměnné taková, aby po dosazení nastala identita termů.
- Dle algorimu na předchozím slajdu, však k unifikaci dojde:

```
?- X = otec(X).
X = otec(X).
```

Poznámka

- Některé implementace Prologu mohou při této unifikaci cyklit.

Unifikace s kontrolou sebevýskytu.

Kontrola sebevýskytu

- Algoritmus je možné modifikovat tak, aby dával korektní odpověď. Pokud se samostatná proměnná vyskytuje jako podvýraz v druhém termu, termy se neunifikují.
- V praxi se často jedná o nadbytečný test, unifikace je velice častá operace, z důvodu výkonnosti se tento test vynechává.

unify_with_occurs_check/2

- Specifický operátor unifikace s testem na sebevýskyt.
- ```
  ?- unify_with_occurs_check(X,otec(X)).
  false.
  ```
  ```

Programování s unifikací

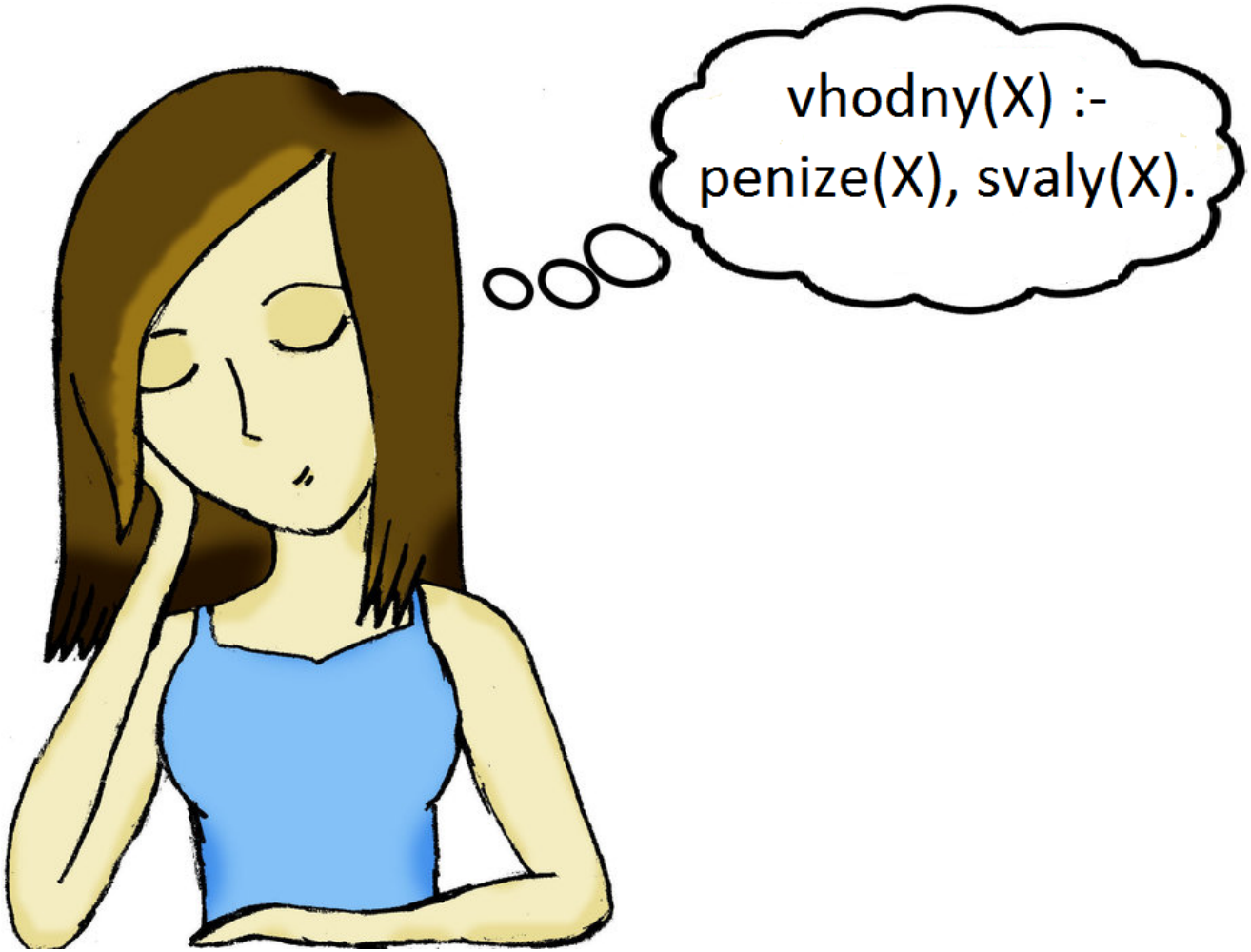
Pozorování

- Unifikace je jeden z fundamentů logického programování.
- Pomocí unifikace můžeme odvozovat i sémantickou informaci.

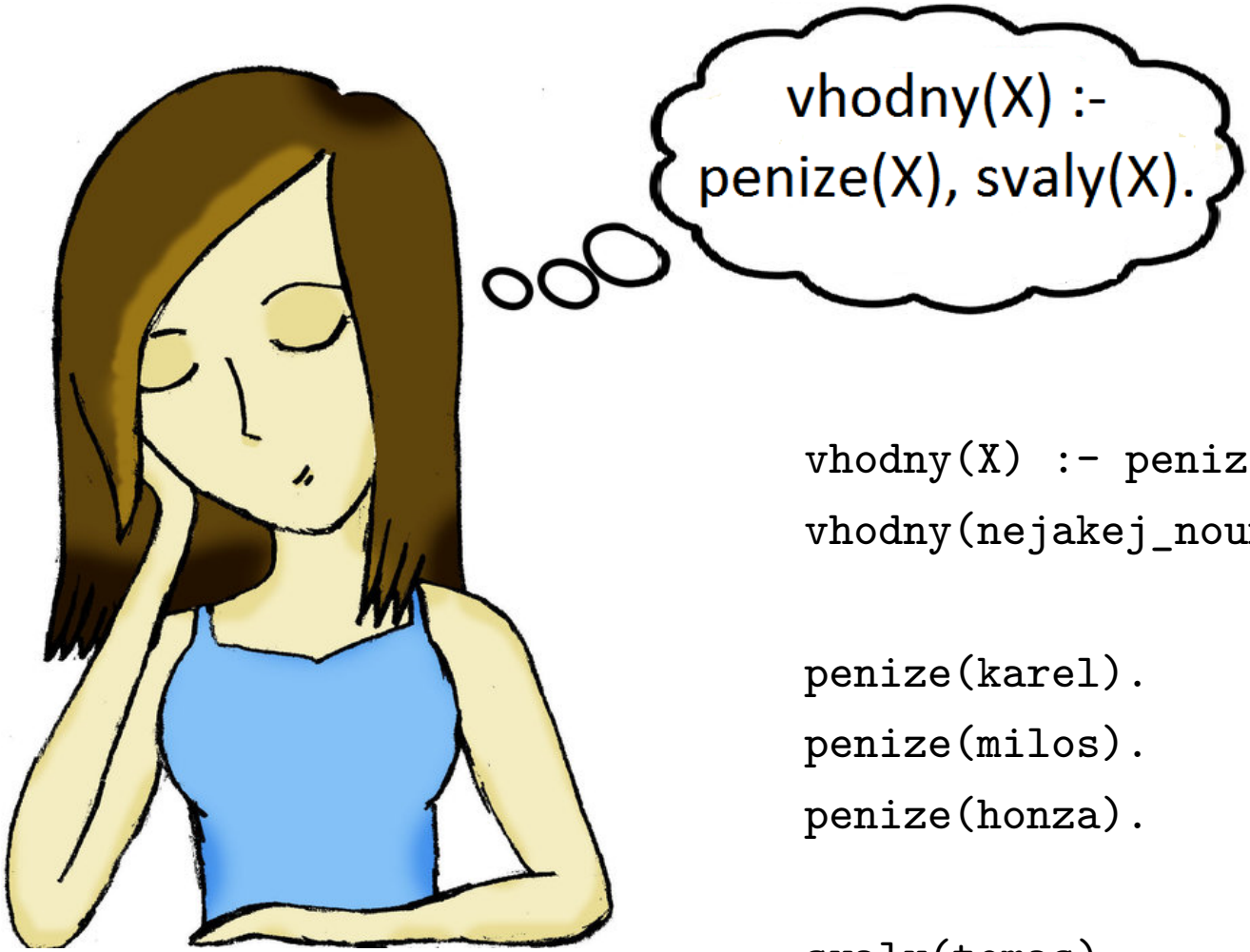
Příklad

- ```
  vertical(line(point(X,Y),point(X,Z))).
  horizontal(line(point(X,Y),point(Z,Y))).

  ?- horizontal(line(point(2,3),point(12,3))).
  true.

  ?- vertical(line(point(1,1),X)).
  X = point(1, _G2240).
  ```
  ```

Jak Prolog počítá

Výpočet v Prologu

Teorie

- Výpočet = dokazování.
- Kódování problému pomocí Hornových klauzulí.
- Dokazování Selektivní Lineární Definitní rezolucí.
- Při výpočtu Prologu se konstruuje a prochází SLD strom.

V rámci IB015

- Princip výpočtu s využitím příkladů a neformální demonstrace postupu bez intelektuální zátěže odpovídajícího teoretického fundamentu.

Pořadí faktů v databázi

Pozorování

- Při výpočtu Prolog vždy využívá fakta v tom pořadí, v jakém jsou uvedeny v programu.

Příklad

- ```
  players(flink,diablo_III).
  players(flink,world_of_tanks).
  players(flink,master_of_orion).
  ```
- ```
  ?- players(flink,X).
  X = diablo_III;                    /* uživatel vložil ; */
  X = world_of_tanks ;               /* uživatel vložil ; */
  X = master_of_orion.
  ```
- ```
  ?- players(flink,X).
  X = diablo_III .                   /* uživatel vložil . */
  ```
  ```

Příběh slečny Prology, aneb jak s pravidly



Příběh slečny Prology, aneb jak s pravidly




```
vhodny(X) :- penize(X), svaly(X).
vhodny(nejakej_nouma).

penize(karel).
penize(milos).
penize(honza).

svaly(tomas).
svaly(honza).
svaly(karel).
```

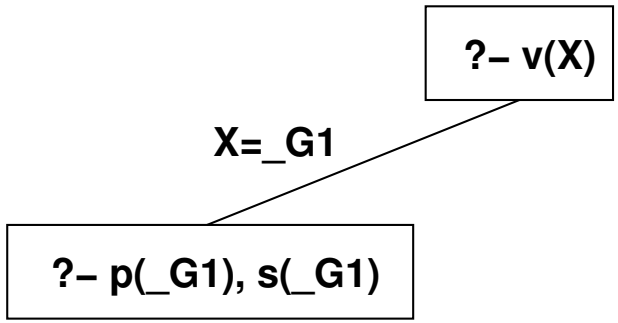
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
```

?– v(X)

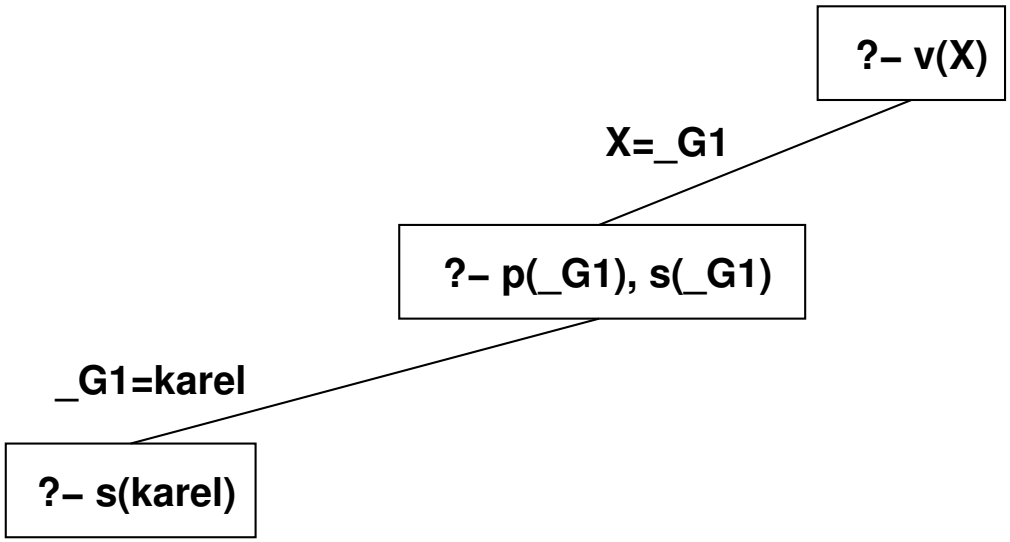
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
```



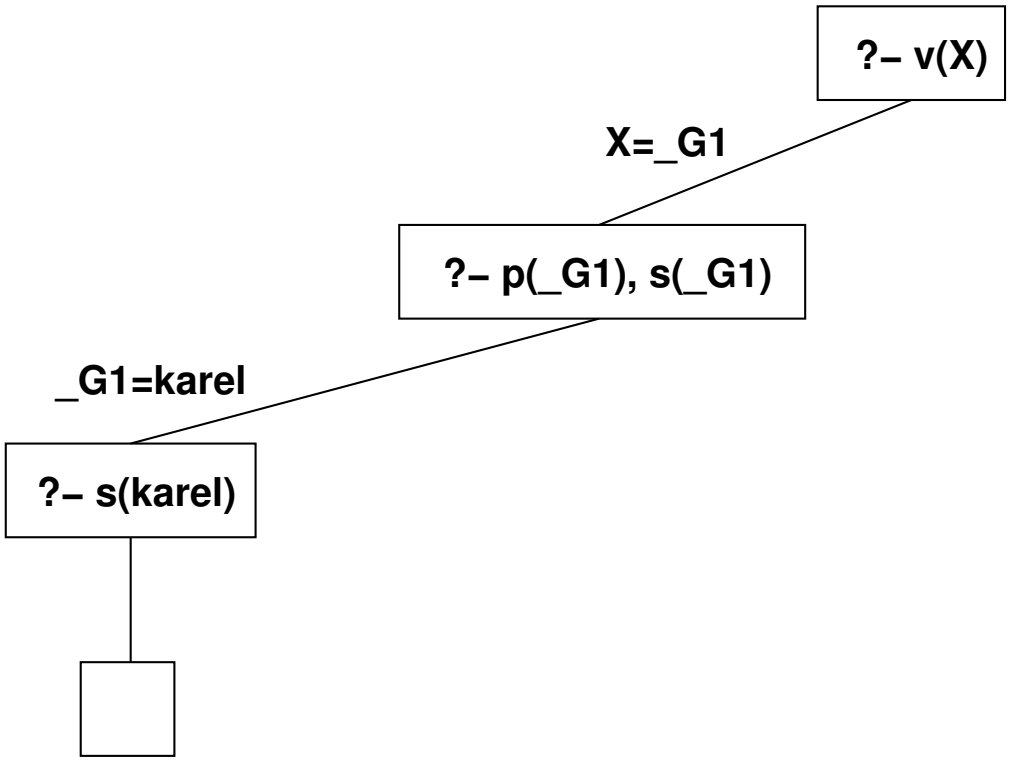
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
```



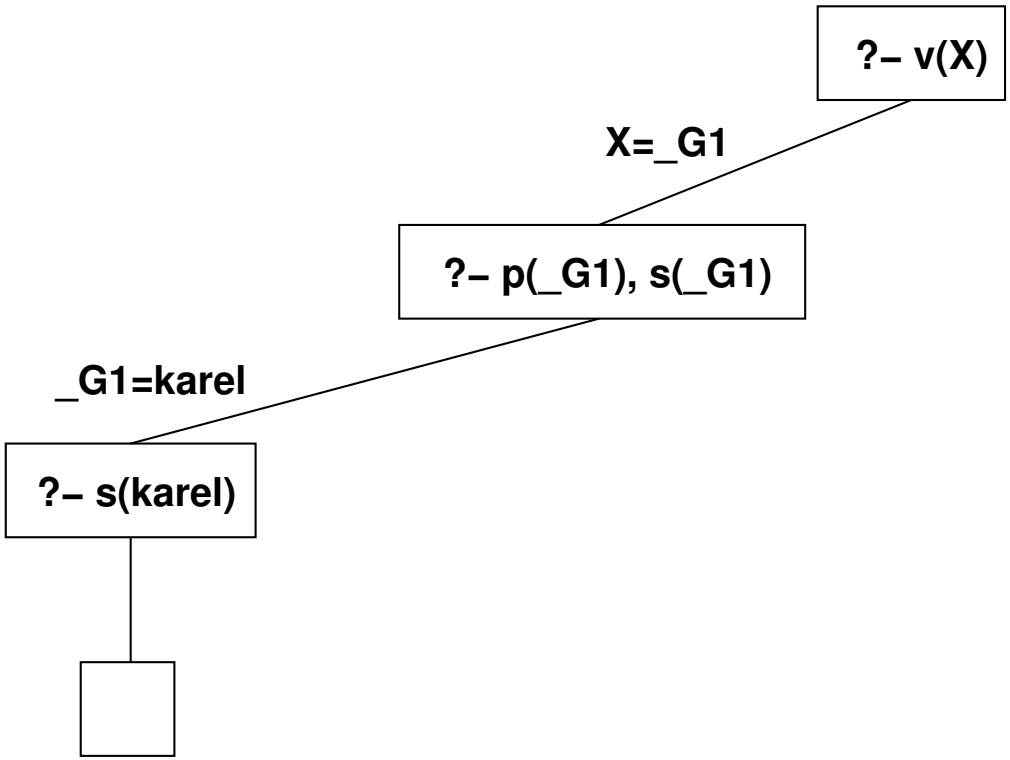
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
```



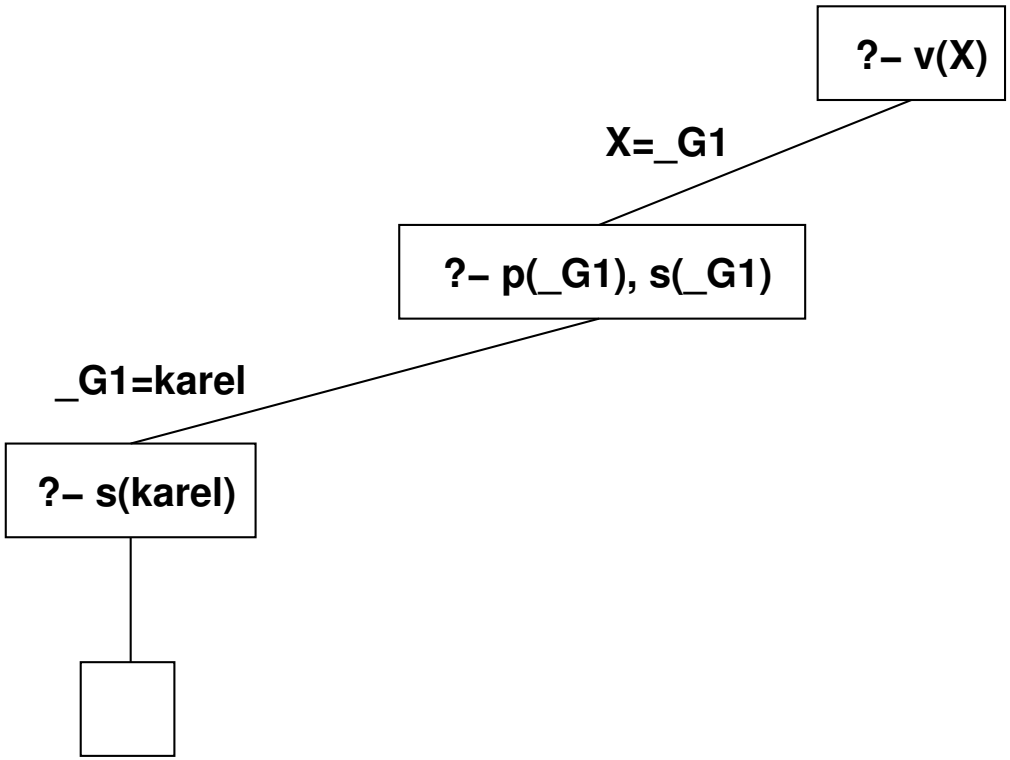
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
karel
```



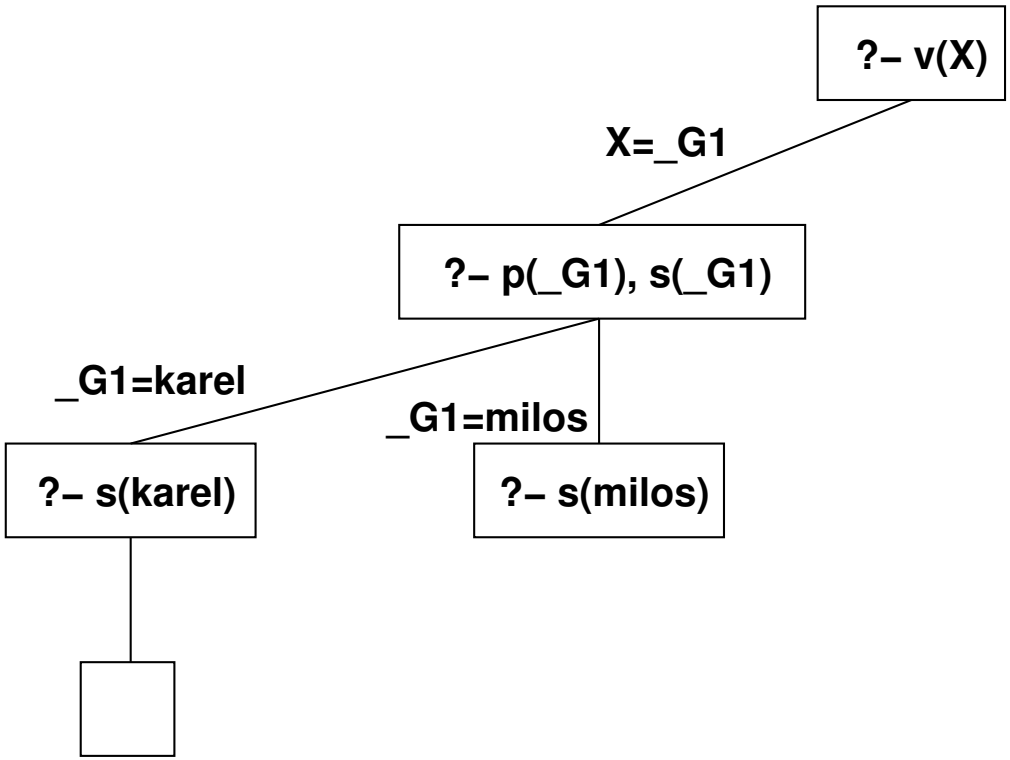
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
karel  ;     /* uživatel vložil ; */
```



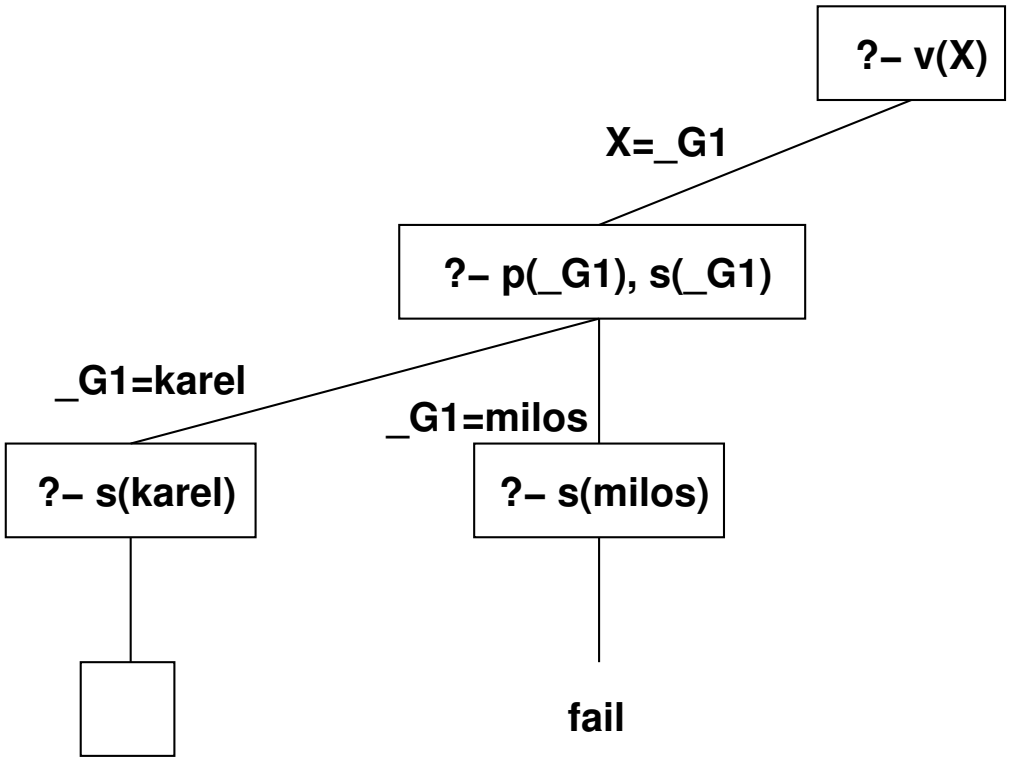
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
karel  ;     /* uživatel vložil ; */
```



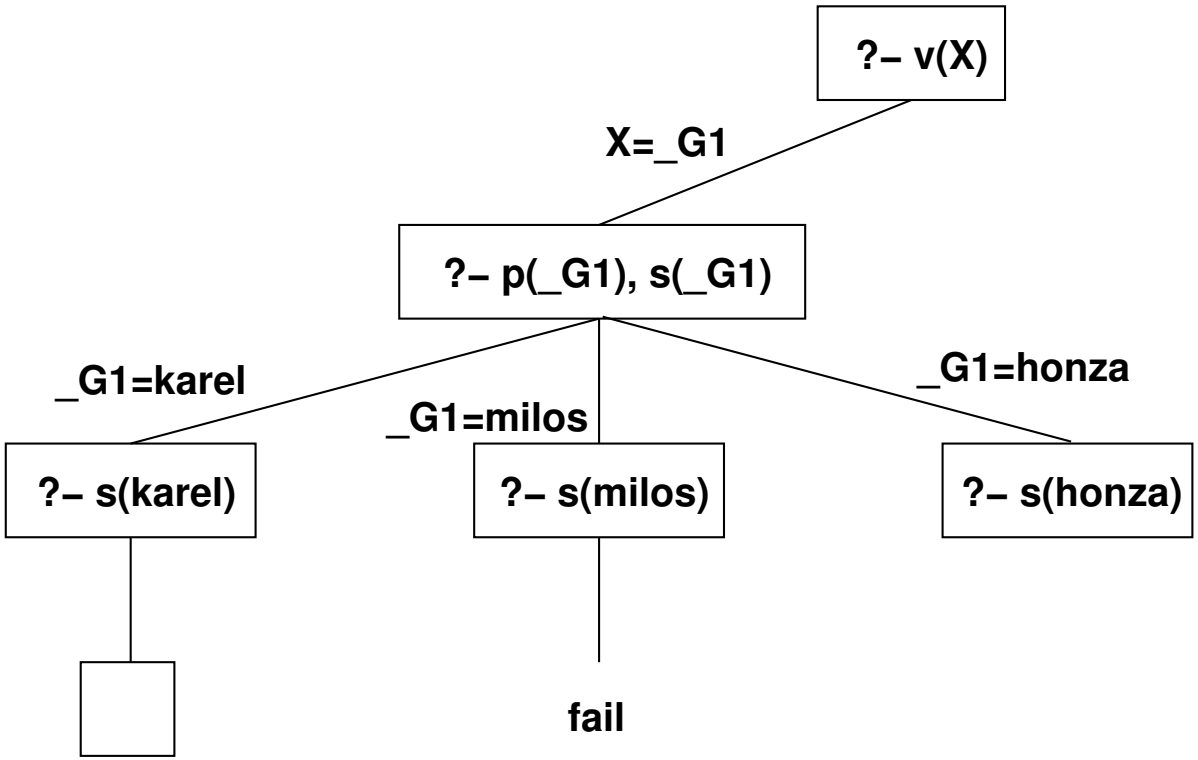
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
karel  ;     /* uživatel vložil ; */
```



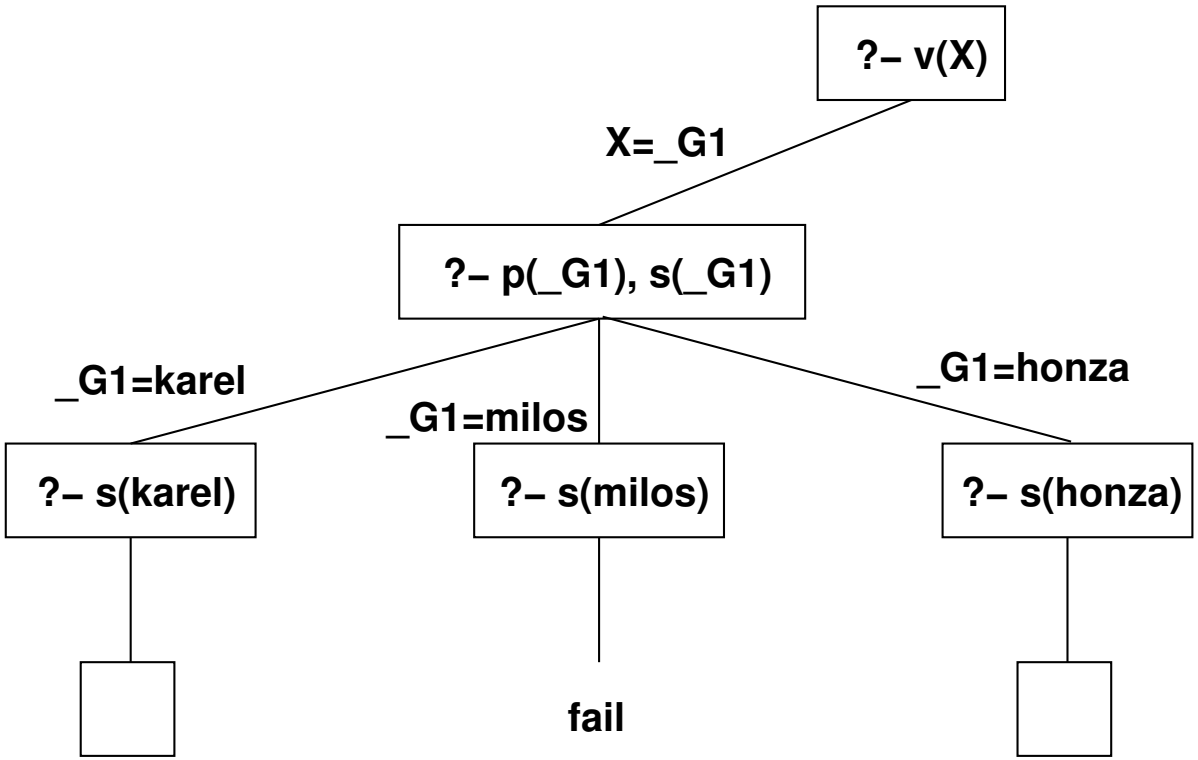
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
karel  ;     /* uživatel vložil ; */
```



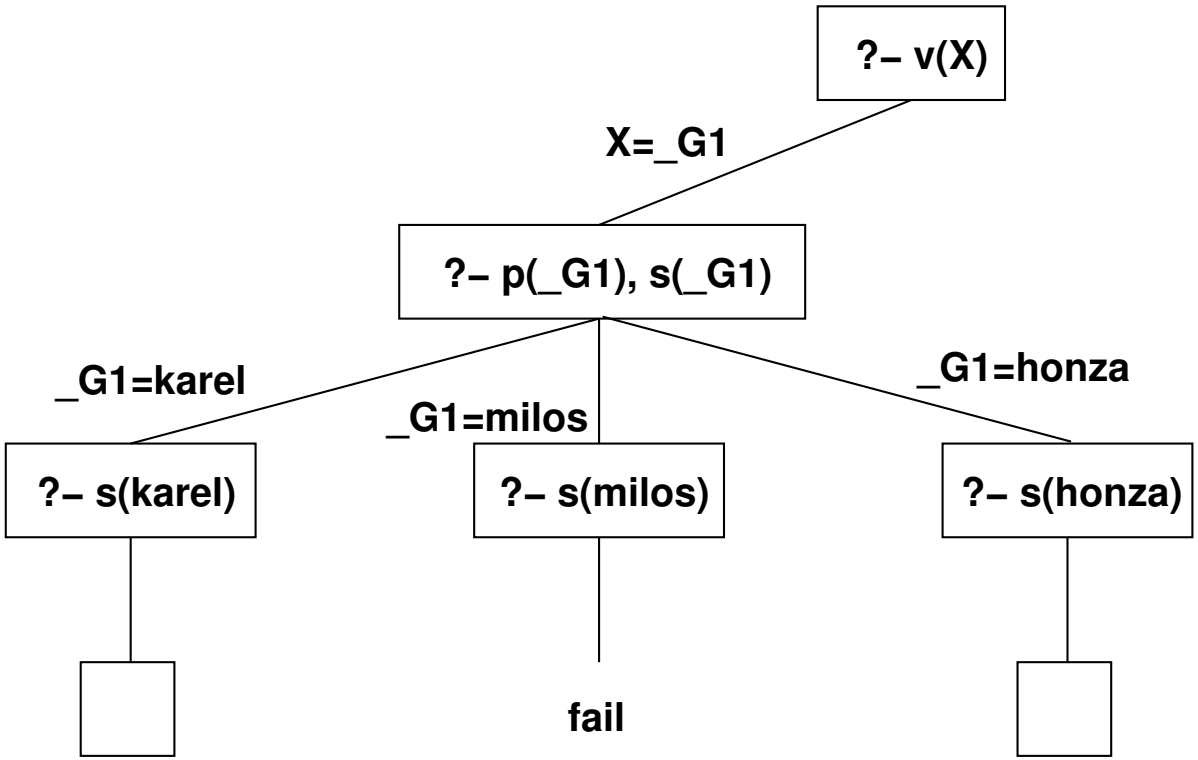
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
karel  ;     /* uživatel vložil ; */
```



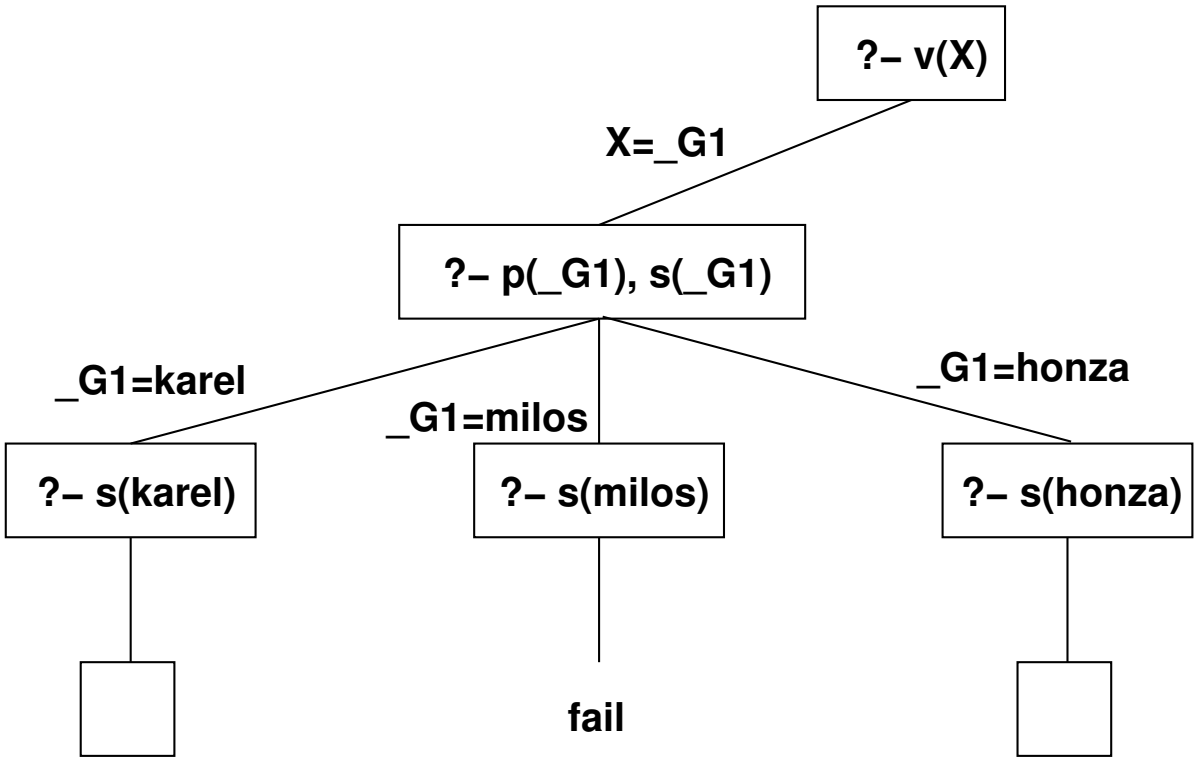
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
karel  ;     /* uživatel vložil ; */
honza
```



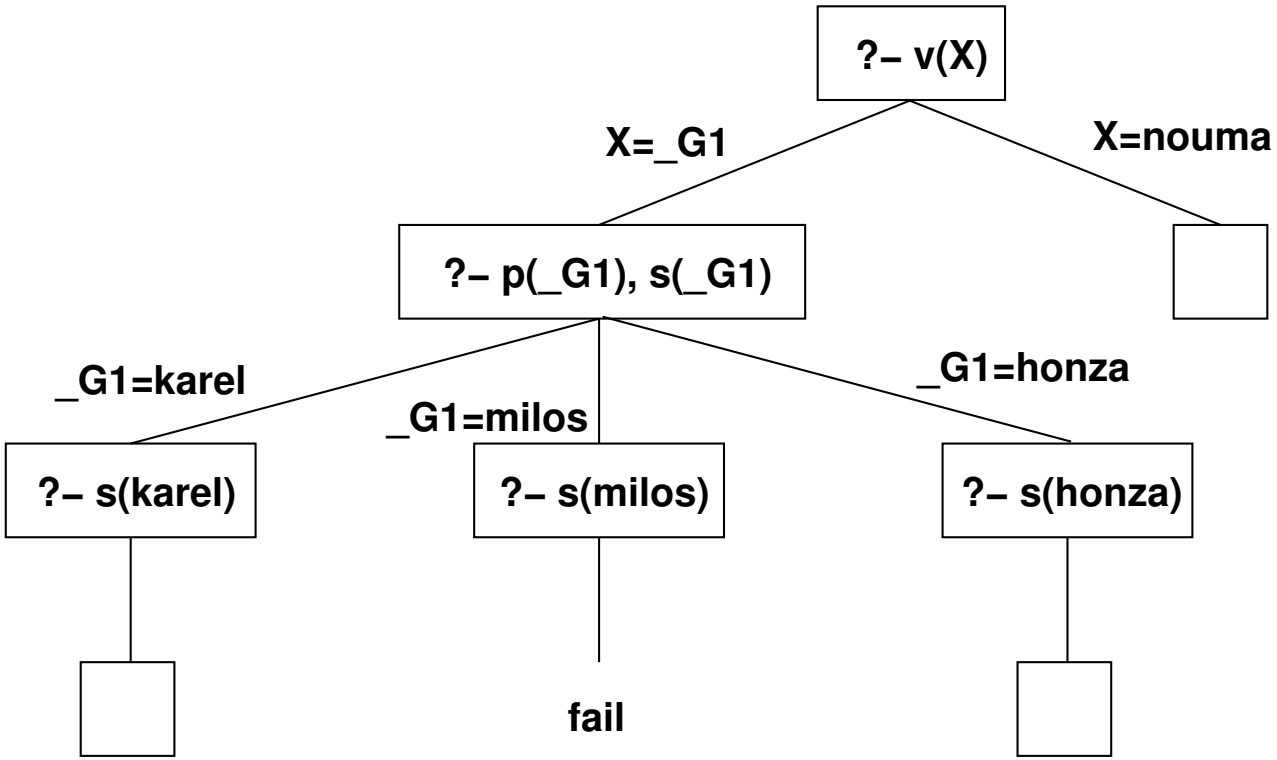
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
karel  ;     /* uživatel vložil ; */
honza  ;     /* uživatel vložil ; */
```



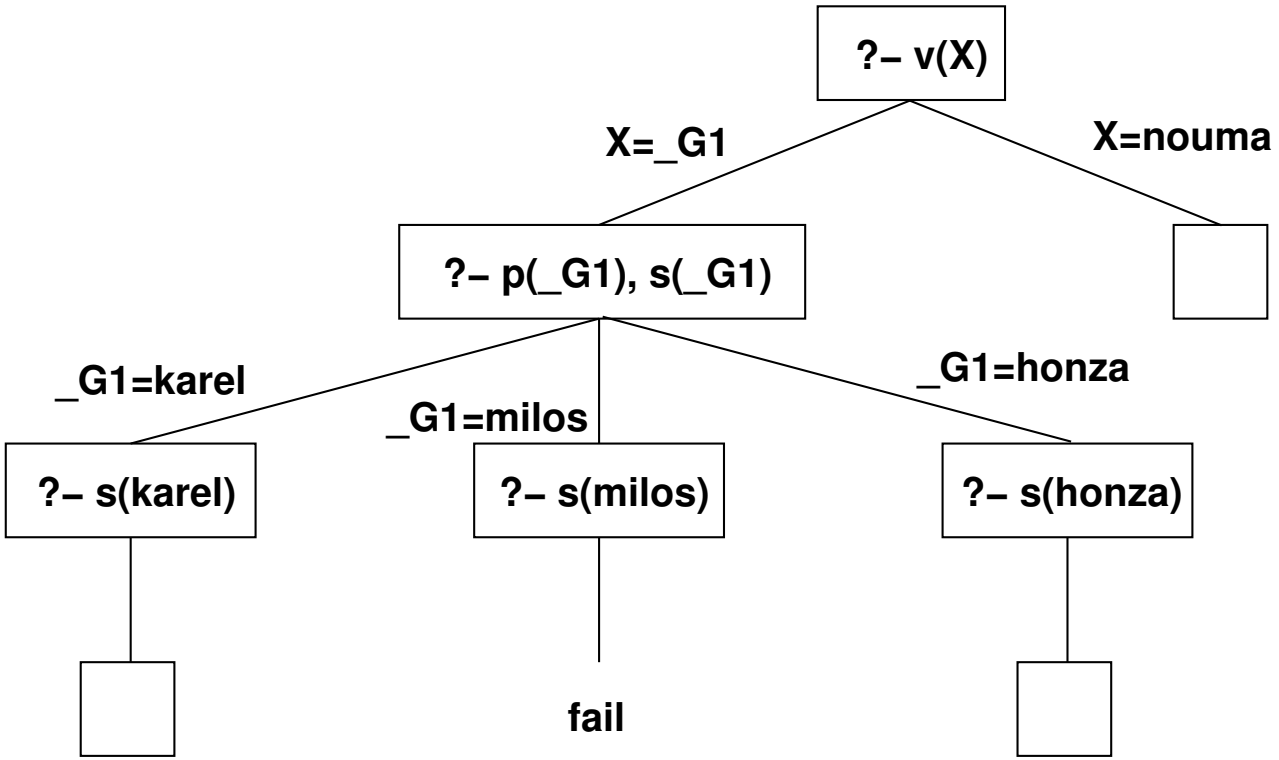
Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
karel  ;     /* uživatel vložil ; */
honza  ;     /* uživatel vložil ; */
```



Příběh slečny Prology

```
v(X) :- p(X), s(X).
v(nouma).

p(karel).
p(milos).
p(honza).

s(tomas).
s(honza).
s(karel).

?-v(X).
karel  ;     /* uživatel vložil ; */
honza  ;     /* uživatel vložil ; */
nouma.
```



Konstrukce výpočetního stromu

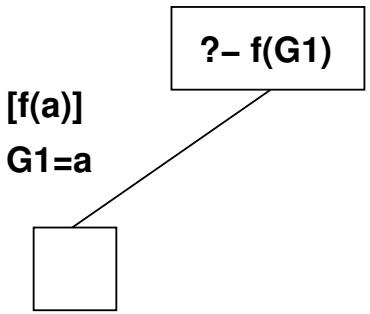
- Pro první podcíl v daném vrcholu se prohledává databáze faktů a pravidel vždy od začátku. Pro každou nalezenou vyhovující položku je vytvořen nový vrchol.
- Vrchol je vytvořen tak, že první podcíl se unifikuje s hlavou nalezené položky, a v nově vzniklém vrcholu je nahrazen tělem nalezené položky (fakta mají prázdné tělo, cíl je "vynechán").
- Hrana vedoucí do nového vrcholu je anotována unifikačním přiřazením. Proměnné vyskytující se v těle pravidla, které nejsou dotčeny unifikací, jsou označeny čerstvou proměnnou.
- Prázdné vrcholy (listy) značí úspěch, hodnoty hledaných proměnných vyplývají z unifikačních přiřazenních na cestě od listu ke kořeni stromu.
- Vrcholy, pro které se nepodařilo nalézt položku v databázi vyhovující prvnímu podcíli jsou označeny podvrcholem **fail**.

Výpočet s rekurzí

?– f(G1)

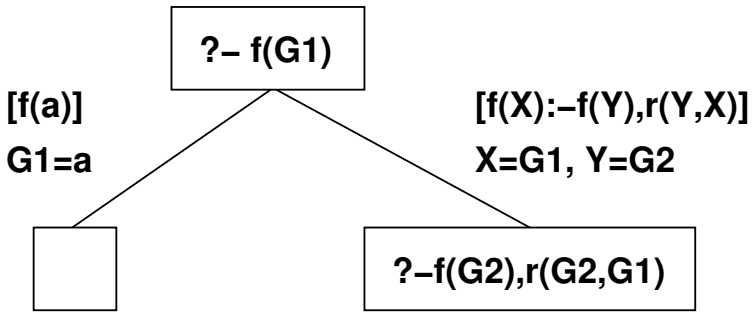
```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

Výpočet s rekurzí



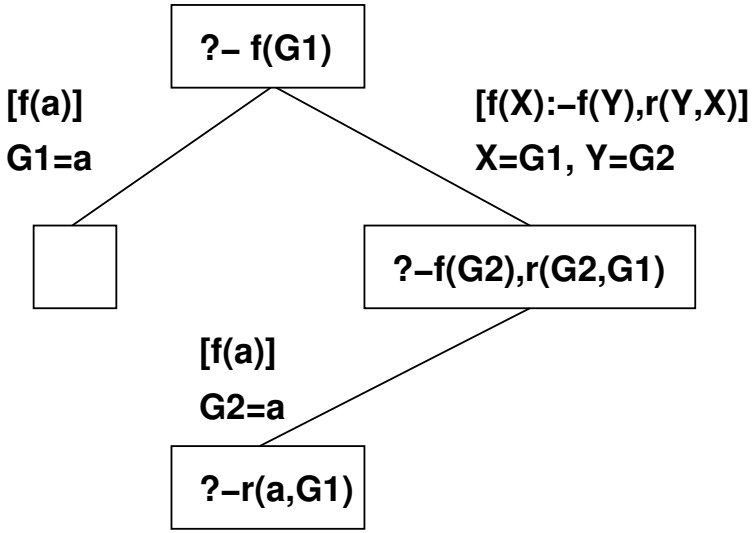

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

Výpočet s rekurzí



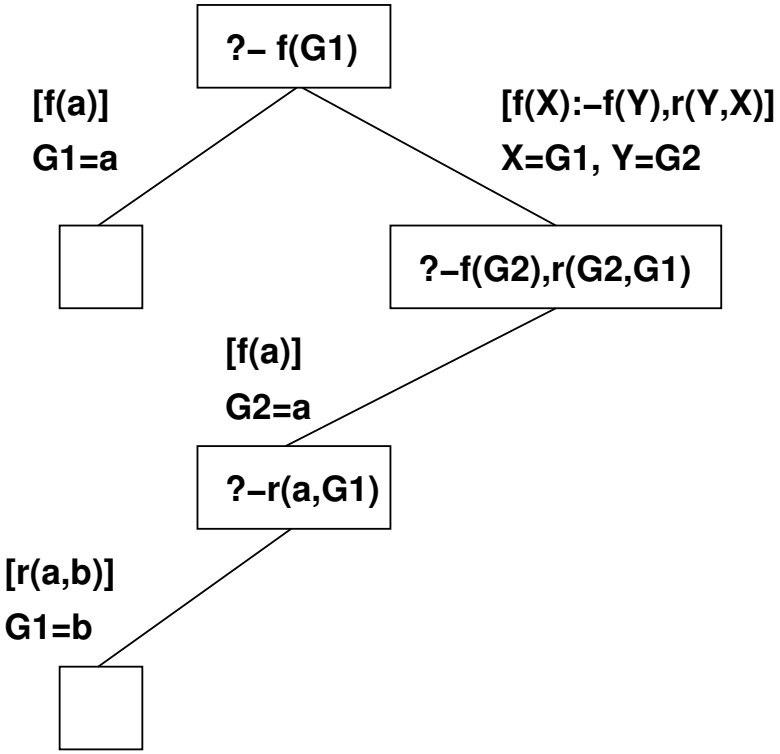

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

Výpočet s rekurzí



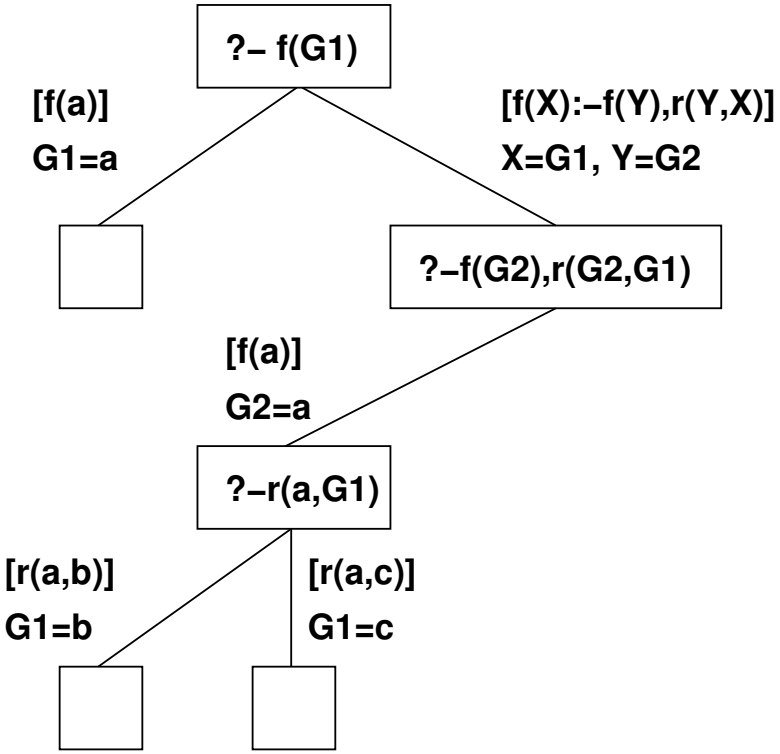

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

Výpočet s rekurzí



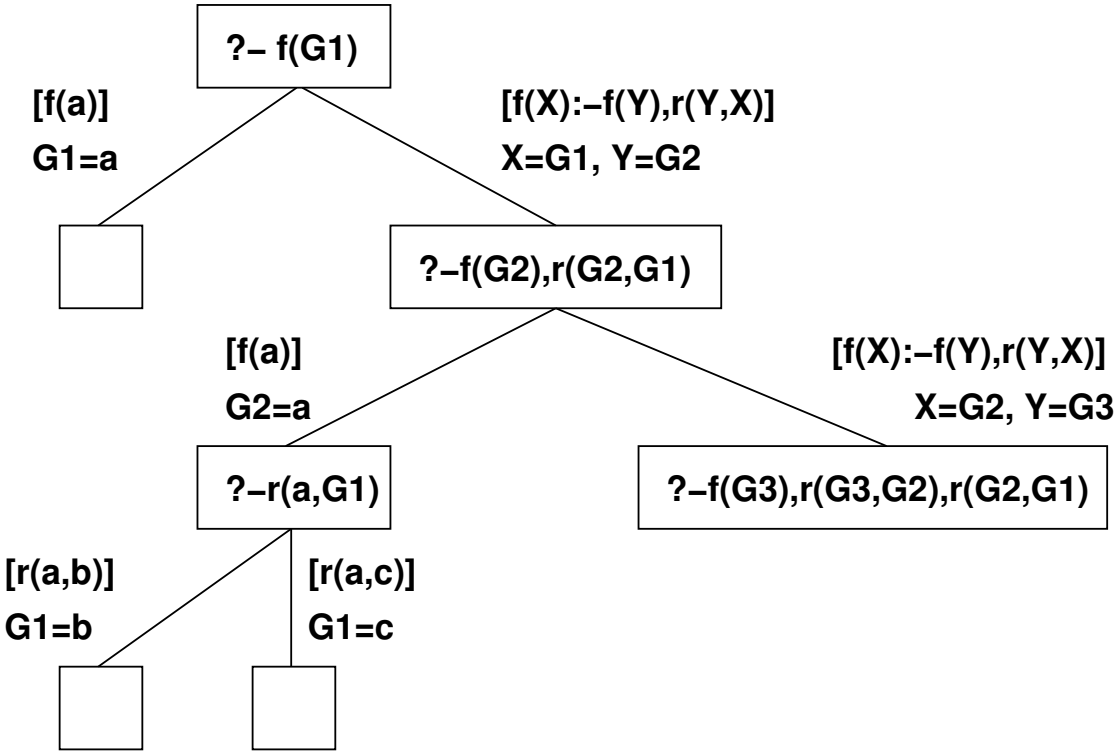

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

Výpočet s rekurzí



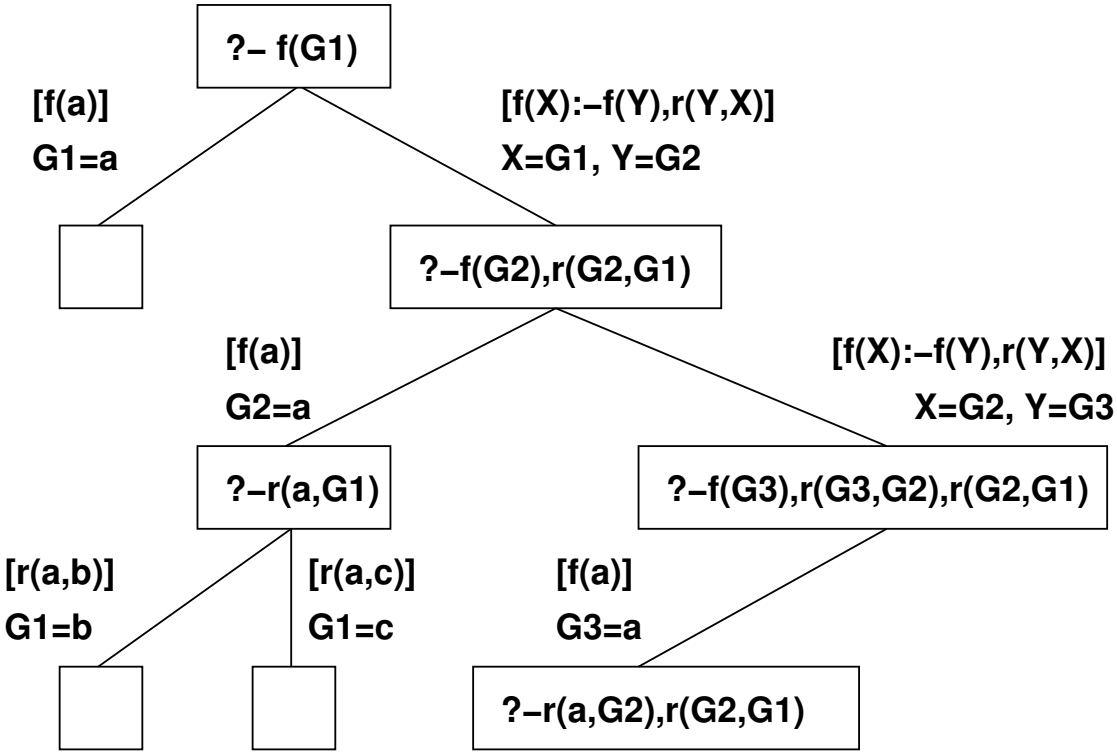

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

Výpočet s rekurzí



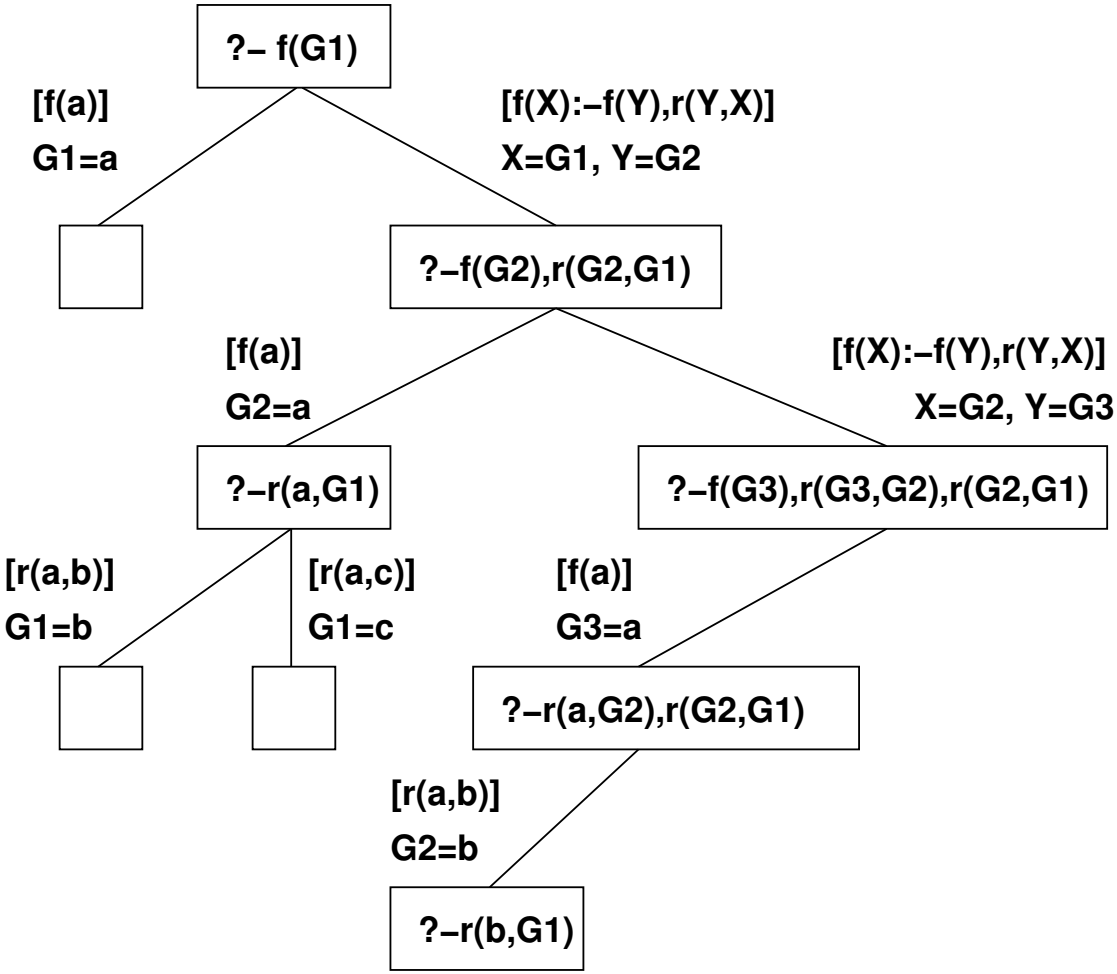

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

Výpočet s rekurzí



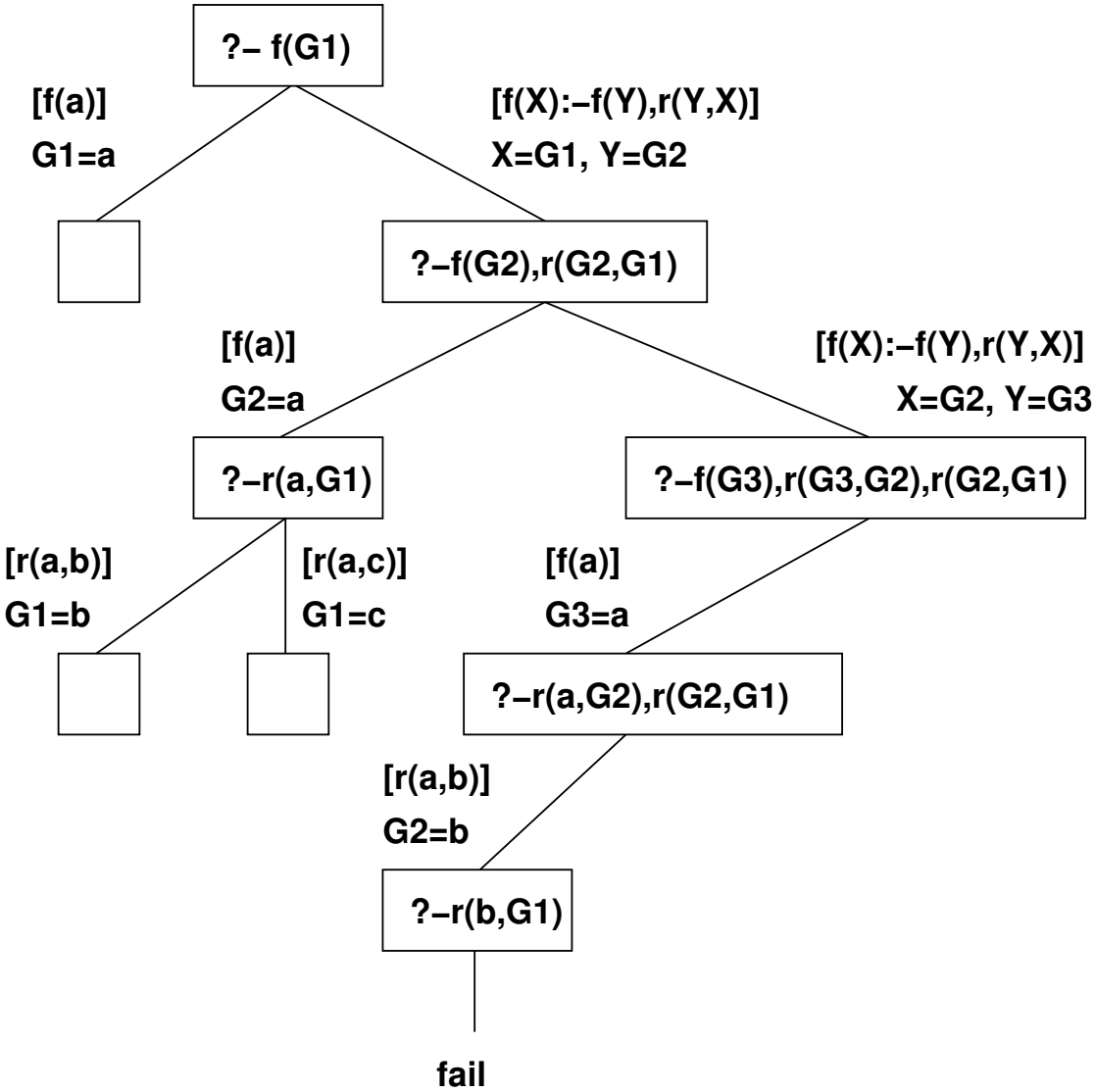

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

Výpočet s rekurzí



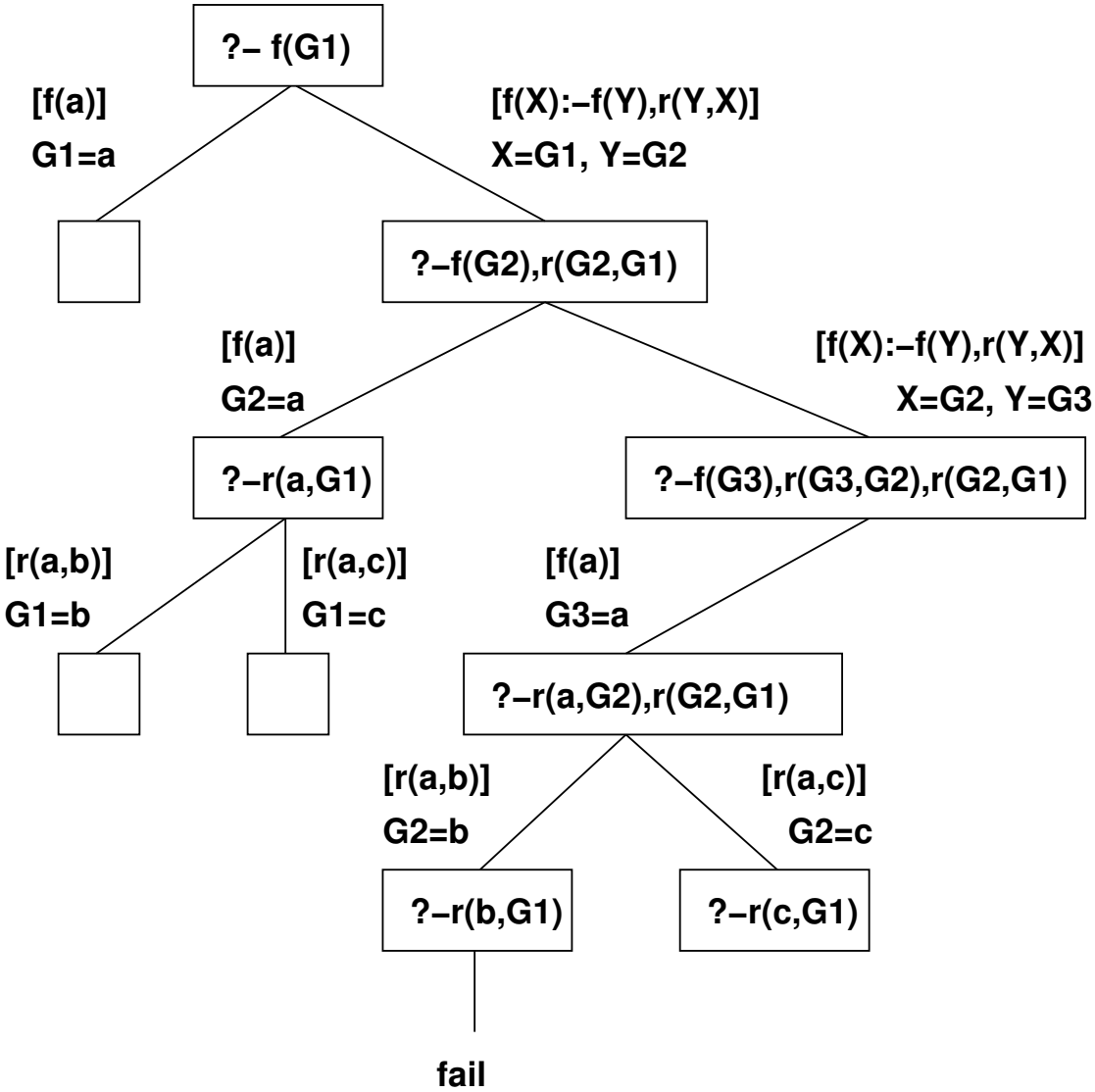

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

Výpočet s rekurzí



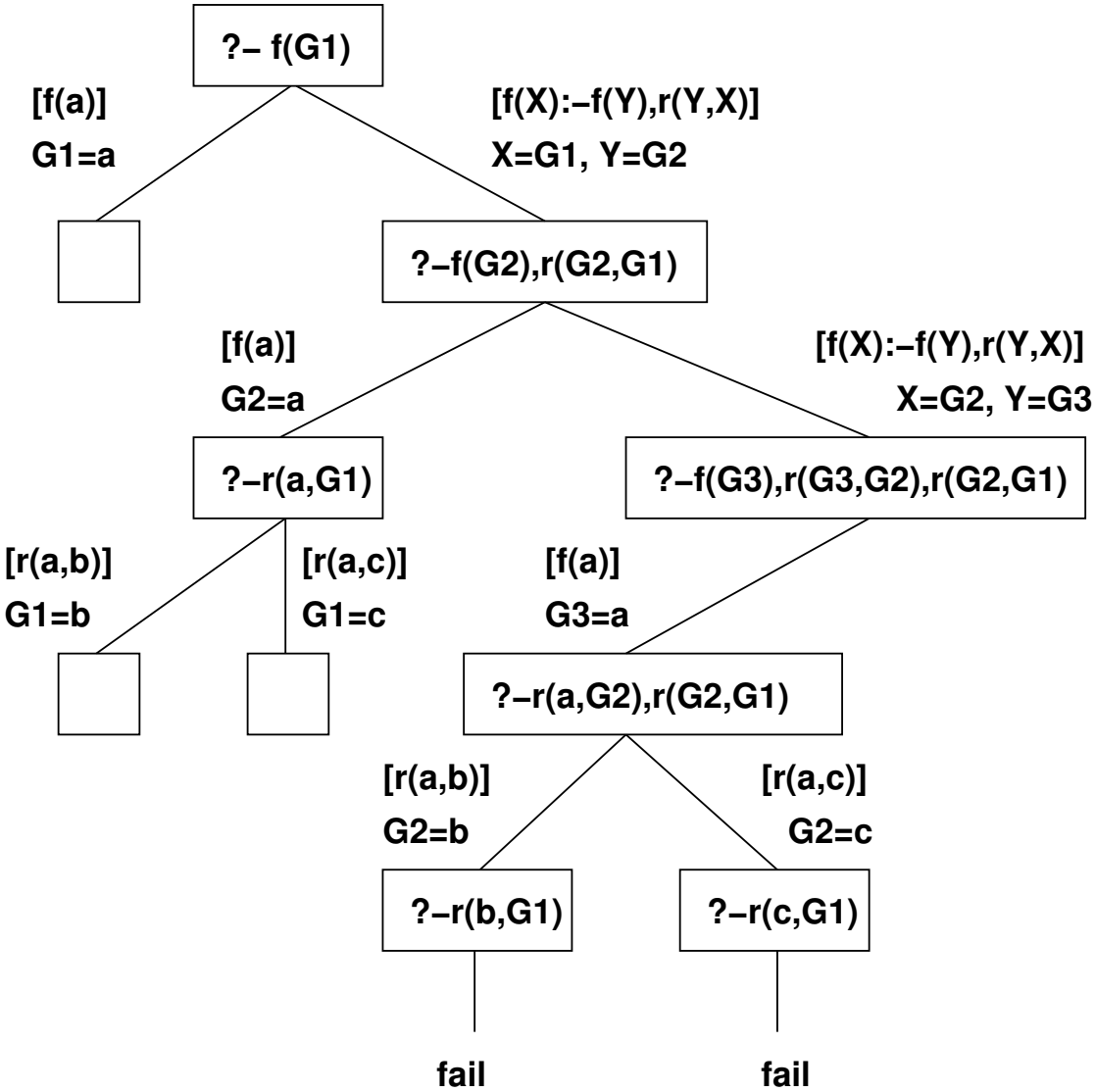

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

Výpočet s rekurzí




```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

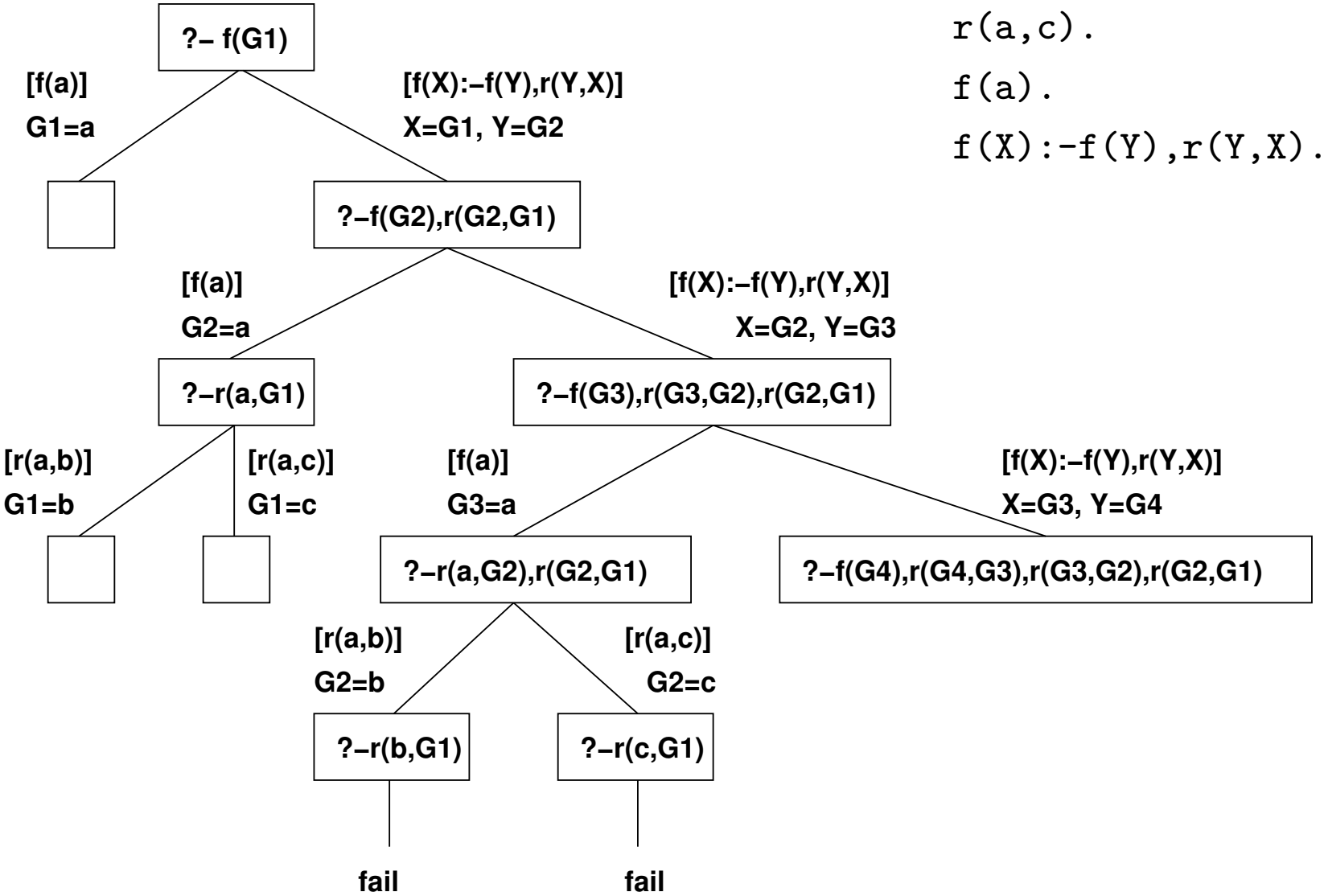
Výpočet s rekurzí




```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```

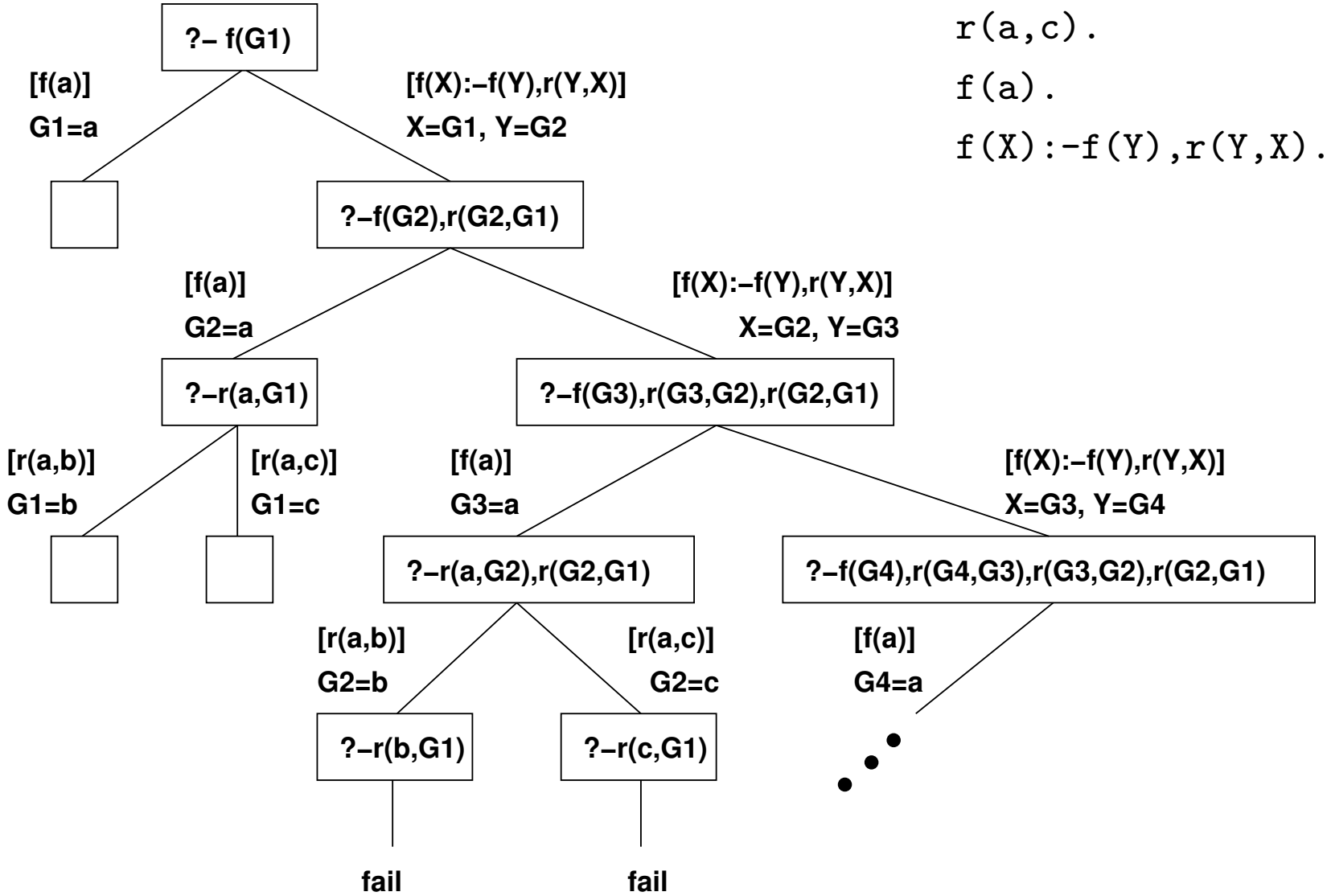
Výpočet s rekurzí

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```



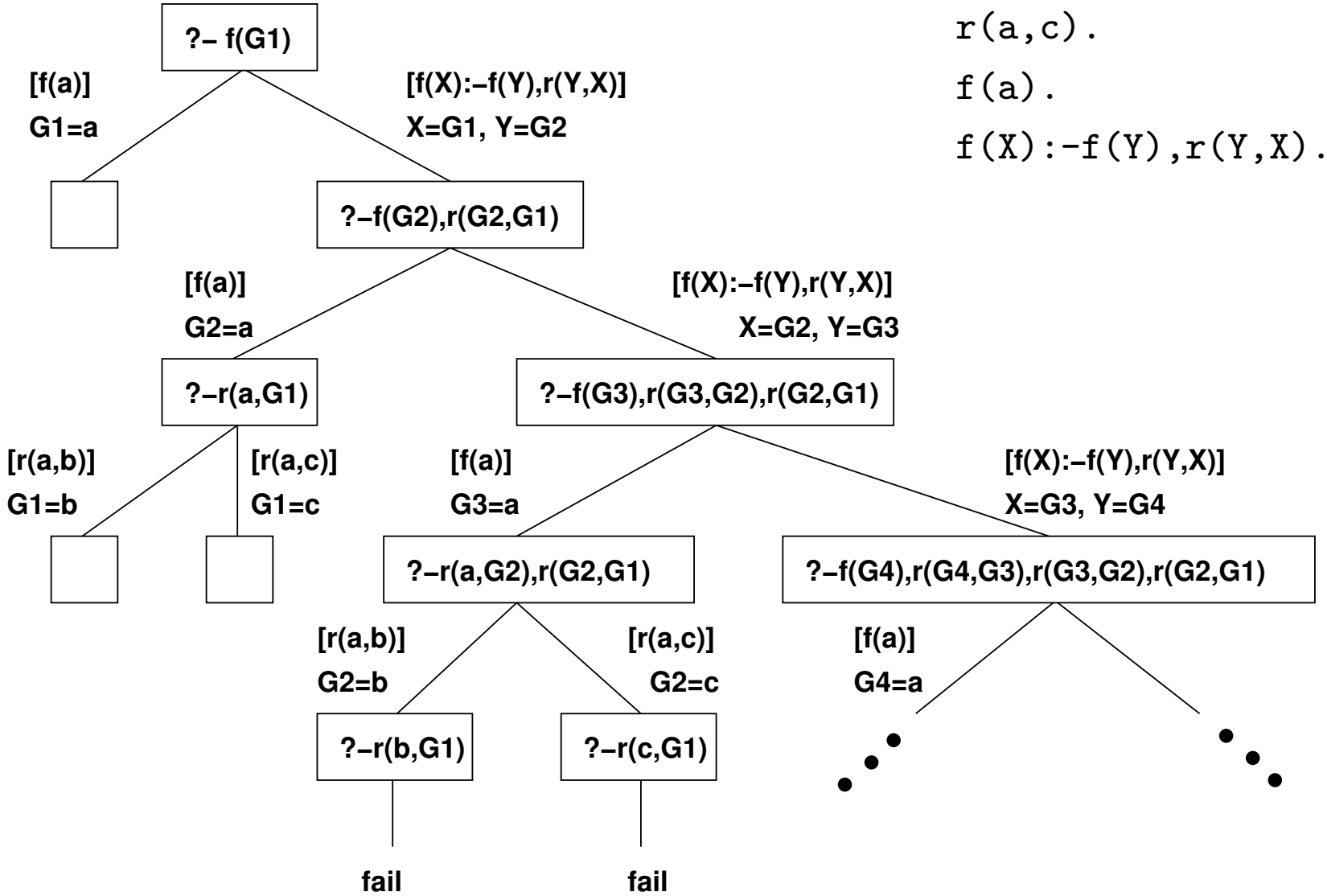
Výpočet s rekurzí

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```



Výpočet s rekurzí

```
r(a,b).
r(a,c).
f(a).
f(X):-f(Y),r(Y,X).
```



Rizika výpočtu a doporučení

Pozorování

- Strom je procházen algoritmem prohledávání do hloubky.
- Výpočet nad nekonečnou větví stromu prakticky končí chybovou hláškou o nedostatečné velikosti paměti zásobníku.
- Použití levorekurzivních pravidel (první podcíl v těla pravidla je rekurzivní použití hlavy pravidla) často vede k nekonečné větvi, a to i v případě, kdy počet vyhovujících přiřazení na původní dotaz je konečný.
- Pořadí faktů a pravidel v databázi neovlivní počet úspěšných listů ve výpočetním stromu, ale ovlivní jejich umístění (tj. pořadí, ve kterém budou nalezeny a prezentovány uživateli.)

Doporučení

- Používají se pravorekurzivní definice pravidel.
- Fakta se uvádějí před pravidly.

Checkpoint

Příklad

- S využitím znalostí prezentovaných v této přednášce naprogramujte nad databází měst:

```
mesto(prague).
mesto(viena).
mesto(warsaw).
mesto(roma).
mesto(paris).
mesto(madrid).
```

  predikát `triMesta/3`, který je pravdivý pokud jako argumenty dostane tři různá města uvedené v databázi.

Nekonečný výpočet

- Narhněte program v jazyce Prolog takový, aby ze zadání bylo zřejmé, že odpověď na určený dotaz je pravdivá, ovšem výpočet Prologu k tomuto výsledků nikdy nedospěje.