Praktikum z logického programování (kombi) - Cvičení č. 2

KOMBINOVANÉ: OPAKOVÁNÍ A REKURZE

Opakování: Aneb co Vám mělo utkvět z minula

Klauzule v Prologu

- Tři typy
  - **Fakta**
    - Tvrzení bez předpokladu, pomocí predikátu definujeme základní informace o modelovaném světě (vlastnosti objektů daného světa).
    - **predikat(parametr).**
  - **Pravidla**
    - Obecná tvrzení ve tvaru „závěr platí pokud platí všechny jeho předpoklady“.
    - **zaver :- predpoklad1, predpoklad2.**
  - **Dotaz**
    - Ptáme se, jestli něco platí, případně se ptáme co platí. Spouštíme program.

Klauzule v Prologu

- Ukončujeme tečkou.
- Klauzule ve tvaru: **hlava :- tělo.**
- Hlava obsahuje vždy a pouze jeden atom!
- V těle může být více atomů
  - Vztah konjunkce **(AND)** – zapisuje se čárkou **(,)**
    - Musí platit všechny předpoklady jinak FALSE
  - Vztah disjunkce **(OR)** – zapisuje se středníkem **(;)**
    - Platí alespoň jeden předpoklad jinak FALSE
    - Disjunkci lze rozdělit na dvě klauzule

Elementy jazyka Prolog

- **Predikát**
  - barva(trava, zelena).
  - Jeho **argumenty musí být v relaci**, jinak FALSE
- **Konstanta**
  - jan, lev, 8, "Petr", 'Alice‘
  - Definuje **jeden konkrétní prvek** daného světa
- **Proměnná**
  - Vždy **jedinečné v rámci dané klauzule** (a to i když se jmenují stejně)!
- **Funktor**
  - Provádění operace a vrácení její hodnoty – typicky aritmetické operace (+, -, *, /, **, //, rem, div, mod, max, min, …)

Obecná proměnná

- Ekvivalent pro **všeobecný kvantifikátor**
  - Může nabývat **různých hodnot**
- Zapisujeme vždy **počátečním velkým písmenem** (X, Osoba, Zvire, …)
- Může být volná nebo vázaná
  - **Vázaná** (inicializovaná)
    - Obsahuje nějakou hodnotu, kterou předáváme. Jakmile je proměnná vázaná, nelze její hodnotu již měnit.
  - **Volná** (neinicializovaná)
    - Neobsahuje hodnotu, používáme pokud chceme, získat nějakou hodnotu z volaného predikátu → stane se vázanou

Anonymní proměnná

- Ekvivalent **existenčního kvantifikátoru**
  - Existuje nějaká hodnota
- Zapisujeme pomocí podtržítka **(_)**, případně názvem začínajícím podtržítkem **(_nekdo)**
- Pokud nás nezajímá konkrétní hodnota, ale víme, že **nějaká hodnota existuje**
  - V dotazu – platí daný predikát pro nějaký prvek?
  - V hlavě pravidla – neklademe omezení na danou hodnotu, ale nepotřebujeme s ní pracovat

Znalostní báze

- Množina klauzulí **(fakta a pravidla)**, mezi nimiž je vztah **konjunkce (AND)**
- Definují námi modelovaný svět, respektive program
- Při provádění programu se prochází od shora dolů
- Pokud se proměnná v dané klauzuli vyskytuje pouze jednou, použijeme anonymní proměnnou
- Prvně uvádíme fakta a až poté pravidla – důležité, pokud jsou stejně pojmenované!

Nápověda a typy argumentů

- **apropos/1** – pro hledání vestavěných predikátů, kde přesně nevíme jak se jmenuje, ale známe nějaké klíčové slovo
- **help/1** – nápověda k vestavěnému predikátu
- Typy argumentů v nápovědě:
  - **+argument** musí být instanciovaný, tedy mít už předem přiřazenu nějakou hodnotu, používáme pro předání hodnoty **do** predikátu
  - **-argument** zde má být volná proměnná, používáme pro potřeby získání hodnoty proměnné **z** predikátu
  - **?argument** instanciovaný parametr nebo proměnná, obousměrný argument tedy pro předání i pro zisk hodnoty

Přiřazování, porovnávání, unifikace

Predikát	Význam	Provádí unifikaci?	Provádí výpočet?
`=`	shoduje se	ano, obě strany	ne
`\=`	neshoduje se, totéž jako not(...=...)	ne	ne
`is`	přiřazení	ano, levá strana	ano, pravá strana
`=:=`	vyhodnotí, ověří shodu	ne	ano, obě strany
`=\=`	vyhodnotí, ověří neshodu	ne	ano, obě strany
`==`	pouze porovná, ověří shodu	ne	ne
`\==`	pouze porovná, ověří neshodu	ne	ne

Všechno je **rekurze**

Rekurze

- Základní výpočetní prvek Prologu
- **Přímá rekurze**
  - **V těle** klauzule se nachází **stejný predikát** jaký je **v hlavě klauzule**
  - citac(X) :- Dalsi is X + 1, citac(Další).
- **Nepřímá rekurze**
  - **V těle** klauzule se **nenachází stejný predikát**, jaký je v hlavě klauzule, ale nachází se zde predikát po jehož volání a volání dalších dospějeme **po konečném počtu zanoření k volání sebe sama**, tedy predikátu, který je v hlavě klauzule
  - zacatek(X) :- nejaka_akce(X, Y), dalsi_krok(Y).
  - dalsi_krok(Y) :- proved_neco(Y), ziskej_dalsi(X), zacatek(X).

Rekurze – pravidla pro znalostní bázi

- Nutnost definice **ukončovací podmínky** abychom nevytvořili nekonečnou rekurzi (limit zanoření → chyba)
  - Ukončující klauzule jako první v bázi pro daný predikát
- Klauzule obsahující rekurzi musí být uvedena až za všemi pravidly se stejným názvem a stejným počtem argumentů (stejný predikát)

Rekurze – jak na to?

- V závislosti na typu výpočtu můžeme používat stejný predikát nebo dva odlišné (jinak pojmenované nebo s jiným počtem argumentů)
- Dva predikáty
  - **Rozhraní** (košilka) – slouží k „uživatelské“ interakci (volání funkce vracející hodnotu na základě poskytnutých parametrů)
    - vypocet(argument_1, argument_2, vysledek)
  - **Pracovní predikát** (výpočetní) – interně „voláno“ rozhraním, provádí skutečný výpočet, používá pomocné proměnné, které nemají být viditelné z „uživatelského“ pohledu
    - tmp_vypocet(argument_1, argument_2, tmp_argument_1, mezivysledek)

Příklad: faktorial

- Rekurzivní definice pro $n \geq 0,\ \mathrm{n!} = \mathrm{n} * (\mathrm{n} - 1)!$
- Rozhraní – faktorial/2
  - **faktorial(+Cislo, -Vysledek)**
    - Cislo – číslo ze kterého chceme faktoriál počítat (N, N-1, …)
    - Vysledek – výsledek výpočtu
- Pracovní – fakt_pom/3
  - **fakt_pom(+Citac, +Akumulator, -Vysledek)**
    - Cislo – číslo ze kterého v daném kroku počítáme faktoriál (N, N-1, …)
    - Akumulator – mezivýsledek pro přinásobování v každém kroku
    - Vysledek – výsledek výpočtu ze zanořeného volání pro Citac === 0

Příklad: faktorial

- **faktorial(N, Vysl) :- fakt_pom(N, 1, Vysl).**
  - Do akumulátoru je vložen neutrální prvek pro násobení důležitý v první kroku výpočtu (první zanoření)
- **fakt_pom(0, Vysl, Vysl).**
  - Klauzule zastavující rekurzi
  - Pokud je Citac roven nule, Vysledek je stejný jako Akumulator ← unifikace třetího parametru s druhým ← relace (0, X, X)
- **fakt_pom(Citac, Akumulator, Vysl) :-**

```
        Citac > 0,                  % test Citace, proč důležité?
        C is Citac - 1,             % dekrementace Citace pro další
                                    % zanoření
        A is Akumulator * Citac,    % přinásobení čitače do
                                    % akumulátoru pro další zanoření
        fakt_pom(C, A, Vysl).       % rekurzivní volání sebe sama s
                                    % novými hodnotami
```

Predikát řezu !

- **!/0** – predikát řezu
- Vyhodnocen **vždy jako true**
- Zabraňuje navracení (backtrackingu)
- Odřízne další možná generovaná řešení pokud jich může být více
- Dva druhy
  - **Zelený** – ovlivňuje pouze **efektivitu provádění** – odřízne generování zbytečných větví
  - **Červený** – ovlivňuje **funkčnost provádění** programu – bez jeho použití dostaneme špatný výsledek

Příklad: faktorial – další varianta

- **faktorial(0, 1).**
- **faktorial(Citac, Vysl) :-**
  Citac > 0,
  C is Citac - 1,
  faktorial(C, V),
  Vysl is Citac * V
  - Jaký je zde rozdíl oproti předchozímu řešení?
  - Co se stane, když do ukončovací klauzule přidáme predikát řezu **!**?
    - **faktorial(0,1) :- !.**
- Jak zjistíme, zdali dané číslo odpovídá faktoriálu jiného?

Vyzkoušejte si!

1. Výpočet Fibonacchiho posloupnosti
   - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …
   - $$F_n = \begin{cases} 0, & pro\ n = 0; \\ 1, & pro\ n = 1; \\ F_{n-1} + F_{n-2} & pro\ ostatní \end{cases}$$
2. Je zadané číslo prvkem Fibonnaciho posloupnosti?
   - isFib(5). → true, isFib(4). → false
3. Vypište posloupnost do zadaného n-tého prvku
   - pro n = 5 je výstup: „0, 1, 1, 2, 3, 5“