Organizační úvod

Poznámka (Organizační úvod)
Bude Moodle. DL1. Kódem je kód cvičení.

Prerekvizity nejsou formální, ale je vhodné mít za sebou ADS1 + logiku.

Zkouška má povinnou písemnou část s materiály z přednášky a překladači Prologu a Haskellu. Zadání i odevzdání prostřednictvím Moodle UK. Zápočet ke zkoušce je doporučený.

Zápočet: Zápočtový program v Prologu / Haskellu + cvičící si mohou volit další podmínky.

1 Úvod

Definice 1.1 (Neprocedurální programování)

Programování bez přiřazovacího příkazu. Lze tak programovat dvěma způsoby: Logické programování = popisujeme problém, který chceme řešit prostředky matematické logiky (Prolog). Funkcionální programování = program je definice funkcí, výpočet je pak aplikace funkce na argumenty (Haskell, Lisp).

Definice 1.2 (Prolog)

U zrodu stáli Robert Kowalski (Edinburgh) a Alain Colmerauer (Marseille).

Aplikace: výuka a výzkum, zpracování přirozeného jazyka, AI, automatické dokazování vět, expertní systémy, dotazovací systémy, systémy řízení, …

Existuje ohromné množství implementací. My budeme používat SWI Prolog.

Definice 1.3 (Syntaxe Prologu)

Před tečkou je term = funktor použitý na konstantu (např. zena(eva). nebo rodic(eva, kain).). Tím (několika řádky začínající vždy stejným funktorem) jsme vytvořili proceduru, která definuje predikát (funktor/počet parametrů) tak, že dává true právě tehdy, když je použitý na tuto konstantu.

Funktor použitý na proměnnou (standardně s velkým prvním písmenem) je definice proměnné.

Čárka je konjunkce. Disjunkce se píše jako středník (využívá se také, když chceme vrátit další možné splnění formule). Konjunkce má vyšší prioritu.

Definice pomocí predikátoru se dělá pomocí `:-`.

Nerovná se značí `\==`.

Kanonický tvar si můžeme vypsat pomocí predikátu `display/1`.

`\=` je operátor, který neuspěje (vrátí `false`), pokud se jeho strany dají unifikovat.

`_` je anonymní proměnná („může na jejím místě být cokoliv“).

`+-*` jsou binární operátory, kterým můžeme přiřadit význam.

Definice 1.4 (Unifikace)

Dva termy lze unifikovat, pokud jsou identické, nebo se stanou identickými po substituci vhodné hodnoty do proměnným v obou termech.

Prolog se vždy snaží najít nejobecnější substituci (tedy dvě proměnné položí rovné a nebude za ně nic dosazovat, pokud to stačí). Dělá to tak, že unifikuje právě tehdy, když

- oba termy jsou stejné konstanty,
- jeden term je proměnná a druhý konstanta (pak se dosadí),
- oba termy jsou proměnné (pak se položí rovny),
- oba termy mají stejnou hlavu (první funktor) a její argumenty se dají unifikovat (rekurze).

Definice 1.5 (Algoritmus splňování cíle)

Unifikační algoritmus + backtracking.

Pozor
Záleží na pořadí.

Platnost proměnné je omezena na pravidlo.

Poznámka
V komentářích se `+` označuje vstupní argument (musí to být základní term bez volných proměnných) a `-` argument výstupní (proměnná).

Definice 1.6 (Seznam)

`[]` je prázdný seznam, neprázdný sezname napíšeme jako `[a, b, …]`.

Se seznamy se pracuje hlavně za pomoci `|`, kterým se oddělí konečný počet prvních termů: `[HLAVA1, HLAVA2, … | ZBYTEK]`.

Asociované seznamy se vyrábí tak, že si definujeme nějaký binární operátor (ať už `+-*` nebo libovolný funktor s dvěma argumenty) a v seznamu pak bude tento operátor aplikovaný na dvojici klíč–proměnná.

První prvek se tedy bere pomocí `|`, poslední se musí získat rekurzí (tedy je to pomalé).

Existuje `member/2`, `select/3` (aplikuje se na prvek, seznam a seznam, prostřední seznam se rozpadne na jeden z jeho prvků a zbytek), `delete/3` (jako select, ale vypustí všechny „instance“ prvku, pozor, má jiné pořadí argumentů), `last/2`, `append/3` (argumenty jsou 3 seznamy, první dva se řetězí na třetí), `\permutation/2`.

Definice 1.7 (Tail recursion (koncová rekurze))

Rekurze, ve které je volání funkce až na konci, takže se dá přeložit na cyklus (a interpreti prologu to udělají).

Definice 1.8 (Deklarativní význam programu, deklarativní správnost)

Na program v Prologu se můžeme dívat jako na množinu formulí.

Program je správný deklarativně, když odpověď na dotaz existuje. (Tj. nemůže dát chybný výsledek.)

Definice 1.9 (Procedurální význam programu, procedurální správnost)

Ale také se na něj můžeme dívat jako na popis cílů, které se mají postupně splnit.

TODO procedurální správnost?

Definice 1.10 (Aritmetika)

`S is T`, kde term `T` je aritmetický výraz. V aritmetických výrazech můžeme používat `+, -, *, /, //, ^, mod, >, <, >=, =<, =:=, =\=`, z nichž poslední dva jsou rovnost a nerovnost.

Lze použít i `max, min, abs, sin, cos, tan, sqrt, log` nebo bitové operace `\/, TODO, <<, >>`.

Poznámka (Programování s omezujícími podmínkami)
`use_module(library(clpfd))`

Operátory `#=, #\=, #<, #>, #=<, #>=`

Definice 1.11 (Ladění)

Pro ladění se hodí `trace/0`, `notrace/0`, `guitracer/0` a `noguitracer/0`.

Nebo můžeme ladit selektivně pomocí `spy(predikát)`, např. `spy()`. Odstranit je můžeme pomocí `nospyall/0` a `nospy(predikát)`. Pomocí `nodebug/0` a `debug/0` můžeme vypnout a zapnout ladění bez ztráty sledovaných predikátů.

Definice 1.12 (Řez)

Predikát `!/0` vždy uspěje, ale když se algoritmus vrací z backtrackingu, tak ho rovnou hodí zpět (místo, aby se šlo na další definici daného predikátu).

Použití řezu lze rozdělit na dvě možnosti: červený řez (mění deklarativní význam programu – přidává negaci, např. deterministický `prvek/2`) a zelený řez (nemění deklarativní význam programu, jen odřezává neperspektivní větve výpočtu, např. `max/2`).

Definice 1.13 (Negace)

`\+`

TODO matice?

2 Neúplně definované datové struktury

Definice 2.1 (Rozdílový seznam)

Nemá nic společného s rozdílem, krom znaku mínus. Je to seznam prvků, za kterými následuje jako zbytek seznamu proměnná. Za tímto seznamem je mínus a znovu ta proměnná, tedy když unifikujeme `[a, b, …, | Y]-Y = L-[c, d, …]`, tak `L` bude seznam `[a, b, …, c, d, …]`.

Zřetězení probíhá v konstantním čase, přidávání na konec v konstantním čase.

Definice 2.2 (Vestavěné predikáty)

Dotazy na to, zda je něco: `atom/1` (atom), `atomic/1` (konstanta), `number/1`, `integer/1`, `float/1`, `var/1` (volná proměnná), `nonvar/1`, `ground/1` (term bez volných proměnných, tzv. základní term), `compound/1` (složený term).

Definice 2.3 (univ)

Predikát `=..` rozebere term (jednu „hladinu“) na `[HlavníFunktor | SeznamArgumentů]`.

Definice 2.4 (arg, functor)

`arg(pořadovécislo, term, hodnota)` a `functor(term, funktor, početargumentů)` nám naopak umožňují term poskládat.

Definice 2.5 (bagof, setof, findall)

Shromažďují výsledky. bagof vrací výsledky opakovaně, pokud se do nich lze dostat více způsoby. findall vždy uspěje a automaticky nebere ohled na proměnné nezadané v cíli.

Dále se probírali grafové algoritmy a vstup a výstup.

Dokončili jsme povídání o programu Elisa. Následně byly predikáty upravující program z něho samého.

3 Funkcionální programování

Definice 3.1 (LISP)

List Processing. My se podíváme na dialekt Scheme.

3.1 Haskell