IB015 Neimperativní programování

Ukázky použití Prologu a závěrečné zhodnocení

Jiří Barnat

Einsteinova hádanka

Zadání hádanky

Popis situace

- Je 5 domů, z nichž každý má jinou barvu.
- V každém domě žije jeden člověk, který pochází z jiného státu.
- Každý člověk pije nápoj, kouří jeden druh cigaret a chová jedno zvíře.
- Žádný z nich nepije stejný nápoj, nekouří stejný druh cigaret a nechová stejné zvíře.

Otázka

- Kdo chová rybičky?
- Za následujících předpokladů ...

Zadání hádanky – nápovědy

1. Brit bydlí v červeném domě.
2. Švéd chová psa.
3. Dán pije čaj.
4. Zelený dům stojí hned nalevo od bílého.
5. Majitel zeleného domu pije kávu.
6. Ten, kdo kouří PallMall, chová ptáka.
7. Majitel žlutého domu kouří Dunhill.
8. Ten, kdo bydlí uprostřed řady domů, pije mléko.
9. Nor bydlí v prvním domě.
10. Ten, kdo kouří Blend, bydlí vedle toho, kdo chová kočku.
11. Ten, kdo chová koně, bydlí vedle toho, kdo kouří Dunhill.
12. Ten, kdo kouří BlueMaster, pije pivo.
13. Němec kouří Prince.
14. Nor bydlí vedle modrého domu.
15. Ten, kdo kouří Blend, má souseda, který pije vodu.

Řešení v Prologu

Copy-paste, aneb programátorova smrt

- `einstein_0.pl`

Přeuspořádání, aneb optimalizace v praxi

- `einstein_1.pl`

Transformace na řešení absolventa FI

- `einstein_2.pl`
- `einstein_3.pl`
- `einstein_4.pl`
- `einstein_5.pl`

Programování s omezujícími podmínkami

Programování s omezujícími podmínkami

Vymezení pojmu

- Obecné neimperativní programovací paradigma.
- V množině možných řešení problému je hledané řešení popsáno pouze omezujícími podmínkami, které musí splňovat.
- Angl. „Constraint programming“.

Aplikace

- Problémy vedoucí na těžké kombinatorické řešení.
- Řízení, rozvrhování, plánování.
- DNA sequencing.
- ...

Obecný postup řešení a domény proměnných

Různé instance paradigmatu

- Podle typu proměnných, vystupujících v popisu problému.
- Pravdivostní hodnoty, Celočíselné hodnoty, Konečné množiny, Doména lineárních funkcí, ...

Postup řešení úloh

• Modelování problému v dané doméně.	Myšlenka
• Specifikace proměnných a jejich rozsahů.	Program
• Specifikace omezujících podmínek.	Program
• Vymezení cíle.	Program
• Zjednodušení zadání, propagace omezení.	Výpočet
• Systematické procházení možných valuací a hledání vyhovujícího řešení.	Výpočet

Constraint Programming a SWI-Prolog

Hostitelské jazyky

- Řešiče uvažovaných úloh jsou obvykle součástí jiného hostitelského programovacího jazyka nebo systému.
- Prvním výrazným hostitelem byly jazyky vycházející z logického programovacího paradigmatu.
- **Constraint Logic Programming** (CLP).

Knihovny ve SWI-Prologu

- **clpfd**: Constraint Logic Programming over Finite Domains

```
?- use_module(library(clpfd)).
```

- **clpqr**: Constraint Logic Programming over Rationals and Reals

```
?- use_module(library(clpqr)).
```

clpfd – výrazy, omezení

Výrazy v celočíselné doméně

- Celé číslo je výrazem v celočíselné doméně.
- Proměnná je výrazem s celočíselné doméně.
- Jsou-li `E1` a `E2` výrazy v celočíselné doméně, pak
  `-E1` (unární mínus)
  `E1+E2` (součet), `E1*E2` (součin), `E1-E2` (rozdíl),
  `E1^E2` (umocnění), `min(E1,E2)`, `max(E1,E2)`,
  `E1/E2` (celočíselné dělení ořezáním),
  `E1 rem E2` (zbytek po dělení /)
  jsou výrazy v celočíselné doméně.

Omezující podmínky

- Relační operátory předřazené znakem `#`.
- `E1 #>= E2`, `E1 #=< E2`,
  `E1 #= E2`, `E1 #\= E2`,
  `E1 #> E2`, `E1 #< E2`,

clpfd – logické spojky

Logické spojky

- `#\ Q` – Negace
- `P #\/ Q` – Disjunkce
- `P #/\ Q` – Konjunkce
- `P #<==> Q` – Ekvivalence
- `P #==> Q` – Implikace
- `P #<== Q` – Implikace

Číselná reprezentace logických hodnot

- Pravda/Nepravda jsou realizovány hodnotami `1` a `0`.
- Relační operátory jsou aplikovatelné na tyto celočíselné hodnoty.

Domény volných proměnných

`?Var` **in** `+Domain`

- Proměnná `Var` má hodnotu z domény `Domain`.

`+Vars` **ins** `+Domain`

- Proměnné v seznamu `Vars` mají hodnotu z domény `Domain`.

all_different(`Vars`**)**

- Každá proměnná ze seznamu `Vars` má jinou hodnotu.

Specifikace domény

- `N` — jednoprvková množina obsahující celé číslo `N`.
- `Lower..Upper` — všechna celá čísla `I` taková, že `Lower =< I =< Upper`, `Lower` musí být celé číslo, nebo term `inf` označující záporné nekonečno, podobně `Upper` musí být celé číslo, nebo term `sup` označující kladné nekonečno.
- `Domain1 \/ Domain2` — sjednocení domén `Domain1` a `Domain2`.

clpfd – jednoduché dotazy

Pozorování

- Následující dotazy jsou řešeny pouze fází propagace omezujících podmínek (neprochází se systematicky prostor všech možných přiřazení hodnot volným proměnným).

Příklady dotazů na `clpfd`

- ```
  ?- X #\= 20.
  X in inf..19\/21..sup.
  ```

- ```
  ?- X*X #= 144.
  X in -12\/12.
  ```

- ```
  ?- 4*X + 2*Y #= 24, X + Y #= 9, X #>= 0, Y #>= 0.
  X = 3, Y = 6.
  ```

- ```
  ?- X #= Y #<==> B, X in 0..3, Y in 4..5.
  B = 0, X in 0..3, Y in 4..5.
  ```

SEND + MORE = MONEY

Popis

- Kryptoaritmetické puzzle, každé písmeno představuje jednu cifru, žádná dvě různá písmena nepředstavují tutéž cifru. Jaké je mapování písmen na číslice?

Zadání pro `clpfd`

```
puzzle([S,E,N,D]+ [M,O,R,E] = [M,O,N,E,Y]) :-
        Vars = [S,E,N,D,M,O,R,Y],
        Vars ins 0..9,
        all_different(Vars),
                  S*1000 + E*100 + N*10 + D +
                  M*1000 + O*100 + R*10 + E #=
        M*10000 + O*1000 + N*100 + E*10 + Y,
        M #\= 0, S #\= 0.
```

`clpfd` – hledání řešení

label(+Vars)

- Zahájí hledání vyhovujících hodnot proměnných `Vars`.
- Totéž, co `labeling([],Vars)`.

labeling(+Options,+Vars)

- Zahájí hledání vyhovujících hodnot proměnných `Vars`.
- Parametry uvedené v seznamu `Options` ovlivňují způsob enumerace hledaných hodnot.

Parametry hledání

- Pořadí fixace proměnných.
- Směr prohledávání domén.
- Strategie větvení prohledávaného stromu.

clpfd – parametry hledání řešení

Pořadí fixace proměnných

- `leftmost` — přiřazuje hodnoty proměnným v tom pořadí, ve kterém jsou uvedeny.
- `ff` — preferuje proměnné s menšími doménami.
- `ffc` — preferuje proměnné, které participují v největším počtu omezujících podmínek.
- `min` — preferuje proměnná s nejmenší spodní závorou.
- `max` — preferuje proměnná s největší horní závorou.

Směr prohledávání domén

- `up` — zkouší prvky domény od nejmenších k největším.
- `down` — zkouší prvky domény od největších k nejmenším.

SEND + MORE = MONEY

Odpověď `clpfd` bez prohledávání

- ```
  Vars = [9, E, N, D, 1, 0, R, Y],
  S = 9, M = 1, O = 0,
  E in 4..7, N in 5..8, D in 2..8, R in 2..8, Y in 2..8,
  all_different([9, E, N, D, 1, 0, R, Y]),
  1000*9+91*E+ -90*N+D+ -9000*1+ -900*0+10*R+ -1*Y#=0.
  ```

Požadavek na prohledávání

- Uvedením podcíle `label([S,E,N,D]).`

Odpověď `clpfd` s vyhledáním valuací proměnných S,E,N a D

- ```
  Vars = [9, 5, 6, 7, 1, 0, 8, 2],
  S = 9, E = 5, N = 6, D = 7,
  M = 1, O = 0, R = 8, Y = 2 ;
  false.
  ```

`clpfd` – vybrané predikáty omezujících podmínek

sum`(+Vars,+Rel,?Expr)`

- Součet hodnot proměnných v seznamu `Vars` je v relaci `Rel` s hodnotou výrazu `Expr`.

scalar_product`(+Cs,+Vs,+Rel,?Expr)`

- Skalární součin seznamu čísel `Cs` s čísly, nebo proměnnými v seznamu `Vs`, je v relaci `Rel` s hodnotou výrazu `Expr`.

serialized`(+Starts,+Durations)`

- Pro hodnoty `Starts=[S1,...,SN]` a `Durations=[D1,...,DN]`, platí, že úlohy začínající v čase `SI` a trvající dobu `DI` se nepřekrývají, tj. `SI+DI =< SJ` nebo `SJ+DJ =< SI`.

Jiné použití clpfd v Prologu

- Aritmetické vyhodnocování v celých číslech bez nutnosti instanciace argumentů aritmetických operací (propagace hodnot všemi směry).

Příklad

- ```
  n_factorial(0,1).
  n_factorial(N,F) :-
    N #> 0, N1 #= N - 1, F #= N * F1,
    n_factorial(N1,F1).
  ```
- ```
  ?- n_factorial(N,1).
  N = 0 ;
  N = 1 ;
  false.
  ```

Deklarativní versus imperativní

Deklarativní programovací paradigma

Princip

- Programem je především formulace cíle a vztahu požadovaného výsledku výpočtu k daným vstupům.
- Popis postupu výpočtu není požadován, nebo je druhotným vstupem zadávaným kvůli zvýšení efektivity výpočtu.

Výhody a nevýhody

+ Kratší a srozumitelnější kód.
+ Méně skrytých chyb.
– Náročnější tvorba kódu, požaduje schopnost abstrakce.
– Riziko neefektivního řešení.
– Obtížná přímá kontrola výpočetního HW.

Imperativní programovací paradigma

Princip

- Programem je popis transformace zadaných vstupů na požadovaný výsledek.
- Popis vztahů výsledku vzhledem ke vstupům není požadován, nebo je do programu vkládán za účelem kontroly korektnosti popisované transformace.

Výhody a nevýhody

+ Detailní kontrola nad postupem výpočtu.
+ Efektivní využití dostupného HW .
+ Snazší tvorba kódu.
– Více prostoru pro zanesení chyb.
– Skryté a dlouho neodhalené chyby.
– Nečitelnost významu programu.

Průnik deklarativních a imperativního světa

Jazykové konstrukce

- Nepojmenované funkce (lambda funkce).
- Parametrický polymorfismus / generické programování.
- Silná typová kontrola.
- Sémantika jazyka oddělená od výpočetního HW.

Programátorský styl

- Přenos kontroly typů z doby za běhu programu do doby kompilace.
- Deklarace vzájemných vztahů vnitřních dat v imperativním programu.
- Programování bez pomocných přepisovatelných proměnných.

Příklad použití deklarativního stylu

Původně imperativním stylem

- ```
  int vysledek=1;
  for (int i=1; i<=N; i++)
  {
    vysledek=vysledek*i;
  }
  print vysledek;
  ```

Nově deklarativním stylem

- ```
  int fact(int n)
  {
    if (n==0) return 1;
    else return n*fact(n-1);
  }
  print fact(N);
  ```

Příklad použití deklarativního stylu

Původně imperativním stylem

- ```
  int vysledek=1;
  for (int i=1; i<=N; i++)
  {
    vysledek=vysledek*i;
  }
  print vysledek;
  ```

- Co to vlastně počítá?

Nově deklarativním stylem

- ```
  int fact(int n)
  {
    if (n==0) return 1;
    else return n*fact(n-1);
  }
  print fact(N);
  ```

Příklad použití deklarativního stylu

Původně imperativním stylem

- ```
  int vysledek=1;
  for (int i=1; i<=N; i++)
  {
    vysledek=vysledek*i;
  }
  print vysledek;
  ```

- Co to vlastně počítá?
- Přepisovatelná proměnná navíc, těžší optimalizace.

Nově deklarativním stylem

- ```
  int fact(int n)
  {
    if (n==0) return 1;
    else return n*fact(n-1);
  }
  print fact(N);
  ```

Příklad použití deklarativního stylu

Původně imperativním stylem

```
int vysledek=1;
for (int i=1; i<=N; i++)
{
  vysledek=vysledek*i;
}
print vysledek;
```

- Co to vlastně počítá?
- Přepisovatelná proměnná navíc, těžší optimalizace.
- Větší prostor pro zanesení chyb (`i=1`, `i<=N`).

Nově deklarativním stylem

```
int fact(int n)
{
  if (n==0) return 1;
  else return n*fact(n-1);
}
print fact(N);
```

Příklad použití deklarativního stylu

Původně imperativním stylem

```
int vysledek=1;
for (int i=1; i<=N; i++)
{
  vysledek=vysledek*i;
}
print vysledek;
```

- Co to vlastně počítá?
- Přepisovatelná proměnná navíc, těžší optimalizace.
- Větší prostor pro zanesení chyb (`i=1`, `i<=N`).
- „Skryté“ chování pro `N=0`.

Nově deklarativním stylem

```
int fact(int n)
{
  if (n==0) return 1;
  else return n*fact(n-1);
}
print fact(N);
```

Příklad použití deklarativního stylu

Původně imperativním stylem

```
int vysledek=1;
for (int i=1; i<=N; i++)
{
  vysledek=vysledek*i;
}
print vysledek;
```

- Co to vlastně počítá?
- Přepisovatelná proměnná navíc, těžší optimalizace.
- Větší prostor pro zanesení chyb (`i=1`, `i<=N`).
- „Skryté“ chování pro `N=0`.

Nově deklarativním stylem

```
int fact(int n)
{
  if (n==0) return 1;
  else return n*fact(n-1);
}
print fact(N);
```

- Jasné chování pro `N=0`.

Příklad použití deklarativního stylu

Původně imperativním stylem

```
int vysledek=1;
for (int i=1; i<=N; i++)
{
  vysledek=vysledek*i;
}
print vysledek;
```

- Co to vlastně počítá?
- Přepisovatelná proměnná navíc, těžší optimalizace.
- Větší prostor pro zanesení chyb (`i=1`, `i<=N`).
- „Skryté“ chování pro `N=0`.

Nově deklarativním stylem

```
int fact(int n)
{
  if (n==0) return 1;
  else return n*fact(n-1);
}
print fact(N);
```

- Jasné chování pro `N=0`.
- Pojmenovaná funkce, syntaktická indicie pro sémantický význam.

A to je konec ...

Závěrem

Co si odneseme do života ...

- Funkcionální výpočetní paradigma.
- Solidní základy programovacího jazyka Haskell.
- Solidní základy programování v Prologu.

Čím ještě nám byl kurz prospěšný ...

- **Deklarativní návyky při návrhu programů a algoritmů mnohokrát využijeme v naší (převážně imperativní) informatické praxi.**
- Mentální posilovna.

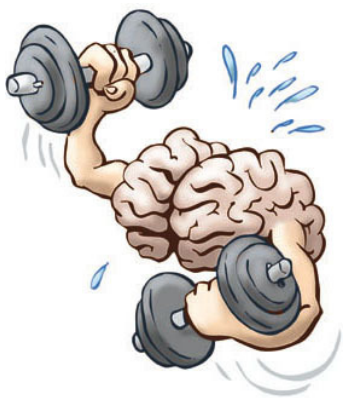
Závěrem

Co si odneseme do života ...

- Funkcionální výpočetní paradigma.
- Solidní základy programovacího jazyka Haskell.
- Solidní základy programování v Prologu.

Čím ještě nám byl kurz prospěšný ...

- **Deklarativní návyky při návrhu programů a algoritmů mnohokrát využijeme v naší (převážně imperativní) informatické praxi.**
- Mentální posilovna.



Poděkování

Přednášejícího studentům

- Za vzornou docházku a přípravu jak na přednášky, tak i na cvičení, vnitrosemestrální a zkouškové písemky, a za celkově poctivý přístup ke studiu.

Studentů přednášejícímu

- Formou zpětné vazby například vyplněním studentské ankety a upozorněním na zásadní, ale i okrajové nedostatky jak přednášejícího, tak i jím připravených studijních materiálů.