Aritmetické operace, vícesměrnost predikátů, práce se seznamy

IB015 Neimperativní programování

Kolektiv cvičících IB015

Fakulta informatiky, Masarykova univerzita

podzim 2018

Opakování

Napište predikát `parentOfTwo(Parent)`, který uspěje, pokud má `Parent` (v souboru `09_pedigree.pl`) alespoň dva potomky.

Aritmetické operátory

- při unifikaci nemají aritmetické operátory speciální význam
  - `1 + 1` odpovídá termu `+(1, 1)`
- predikát `is/2` aritmeticky vyhodnotí pravou stranu, výsledek unifikuje s levou
  - pravá strana nesmí obsahovat neinstanciované proměnné

Aritmetické operátory

- při unifikaci nemají aritmetické operátory speciální význam
  - `1 + 1` odpovídá termu `+(1, 1)`
- predikát `is`/`2` aritmeticky vyhodnotí pravou stranu, výsledek unifikuje s levou
  - pravá strana nesmí obsahovat neinstanciované proměnné

Příklad 10.1.1: Vyhodnoťte následující výrazy a vysvětlete chování jednotlivých operátorů.

a) `2 = 1 + 1`
b) `1 + 1 = 1 + 1`
c) `2 is 1 + 1`
d) `1 + 1 is 1 + 1`
e) `X = 1 + 1`
f) `X is 1 + 1`
g) `2 is 1 + X`
h) `X = 1, 2 is 1 + X`

Aritmetické operátory

- `==` je test na identitu termů (bez unifikace)
- `=:=` je test na aritmetickou shodu (vyhodnocuje obě strany)

Aritmetické operátory

- `==` je test na identitu termů (bez unifikace)
- `=:=` je test na aritmetickou shodu (vyhodnocuje obě strany)

Příklad 10.1.2: Vyhodnoťte následující výrazy a vysvětlete chování jednotlivých operátorů.

a) `X = 2`
b) `X == 2`
c) `X = 2, X == 2`
d) `X =:= 2`
e) `1 + 1 == 2`
f) `1 + 1 =:= 2`
g) `2 =:= 1 + 1`

Aritmetické operátory

- relační operátory <, >, =<, >= aritmeticky vyhodnocují obě strany
- \== je test na neidentitu
- =\= je aritmetická nerovnost (vyhodnotí obě strany)

Aritmetické operátory

- relační operátory `<`, `>`, `=<`, `>=` aritmeticky vyhodnocují obě strany
- `\==` je test na neidentitu
- `=\=` je aritmetická nerovnost (vyhodnotí obě strany)

Příklad 10.1.3: Vyhodnoťte následující výrazy a vysvětlete chování jednotlivých operátorů.

a) `2 < 2 + 1`
b) `1 + 2 =< 2 + 1`
c) `1 + 2 >= 1`
d) `2 * 3 > 3 * 1.5`
e) `1 + 1 \== 2`
f) `1 + 1 =\= 2`
g) `1 + 2 =\= 2`

Aritmetika

Příklad 10.1.7: Napište predikát `firstnums/2`, který spočítá součet prvních `N` přirozených čísel. Například dotaz `?- firstnums(5, X).` uspěje s unifikací `X = 15`.

Aritmetika

Příklad 10.1.7: Napište predikát `firstnums/2`, který spočítá součet prvních `N` přirozených čísel. Například dotaz `?- firstnums(5, X).` uspěje s unifikací `X = 15`.

Predikát má každý argument v jednom z módů:

- `+` (instanciovaný),
- `-` (proměnná),
- `?` (cokoliv – instanciovaný nebo proměnná).

Aritmetika

Příklad 10.1.9: Napište predikát `powertwo/1`, který uspěje, pouze když číslo zadané jako argument je mocninou dvou. Využijte fakt, že mocniny dvou lze opakovaně beze zbytku dělit dvěma, dokud nedostaneme jedna. Můžete využít binární operátory `mod` pro modulo a `div` pro celočíselné dělení.

Příklad 10.1.8: Implementujte predikát `fact/2`, který při dotazu `fact(`m`, `n`)` uspěje, pokud $m > 0$ a $n = m!$. V ostatních případech může interpret cyklit.
Predikát by měl rovněž fungovat při volání ve tvaru `fact(`n`, Res)`, kde `Res` je volná proměnná, a unifikovat tuto proměnnou s $n!$.

Seznamy

- výčtem prvků: `[1,2,3]`, `[a, 1, [3, b]]`
- ve tvaru `[ {vycet zacatku} | {seznam} ]`:
  `[1,2,3 | [5] ]`, `[ a, [1], 1 | [[]] ]`
- predikát `is_list/1` testuje, zda je argument seznam

Seznamy

- výčtem prvků: `[1,2,3]`, `[a, 1, [3, b]]`
- ve tvaru `[ {vycet zacatku} | {seznam} ]`:
  `[1,2,3 | [5] ]`, `[ a, [1], 1 | [[]] ]`
- predikát `is_list/1` testuje, zda je argument seznam

Příklad 10.2.1: Které z následujících zápisů představují korektní zápis seznamu? Pokud je zápis korektní, určete počet prvků v daném seznamu.

a) `[1|[2,3,4]]`
b) `[1,2,3|[]]`
c) `[1|2,3,4]`
d) `[1|[2|[3|[4]]]]`
e) `[[]|[]]`
f) `[[1,2]|[4]]`
g) `[[1,2],[3,4]|[5,6,7]]`

Predikáty na seznamech

V hlavě můžeme používat unifikaci v seznamu

- `split([X|XS], X, XS).`

Některé zabudované seznamové predikáty:

- `length(?List, ?Length)` zjistí délku seznamu,
- `member(?Elem, ?List)` zjistí, zda je `Elem` obsažen v seznamu `List`,
- `append(?List1, ?List2, ?list1AndList2)` spojí seznamy `List1` a `List2` do seznamu `List1AndList2`.

Vícesměrnost predikátů

Popište slovy, co počítají (kdy uspějí) následující predikáty:

```
?- member(2, [1,2,3]).
```

Vícesměrnost predikátů

Popište slovy, co počítají (kdy uspějí) následující predikáty:

```
?- member(2, [1,2,3]).
```

- Uspěje, pokud `2` je prvkem `[1,2,3]`.

```
?- member(X, [1,2,3]).
```

Vícesměrnost predikátů

Popište slovy, co počítají (kdy uspějí) následující predikáty:

```
?- member(2, [1,2,3]).
```

- Uspěje, pokud `2` je prvkem `[1,2,3]`.

```
?- member(X, [1,2,3]).
```

- Unifikuje do `X` všechny prvky `[1,2,3]`.

```
?- member(2, X).
```

Vícesměrnost predikátů

Popište slovy, co počítají (kdy uspějí) následující predikáty:

```
?- member(2, [1,2,3]).
```

- Uspěje, pokud `2` je prvkem `[1,2,3]`.

```
?- member(X, [1,2,3]).
```

- Unifikuje do `X` všechny prvky `[1,2,3]`.

```
?- member(2, X).
```

- Unifikuje do `X` všechny seznamy obsahující `2`.

```
?- member(X, Y).
```

Vícesměrnost predikátů

Popište slovy, co počítají (kdy uspějí) následující predikáty:

```
?- member(2, [1,2,3]).
```

- Uspěje, pokud `2` je prvkem `[1,2,3]`.

```
?- member(X, [1,2,3]).
```

- Unifikuje do `X` všechny prvky `[1,2,3]`.

```
?- member(2, X).
```

- Unifikuje do `X` všechny seznamy obsahující `2`.

```
?- member(X, Y).
```

- Unifikuje do `Y` všechny seznamy obsahující `X`.

Predikáty na seznamech

Příklad 10.2.2: Implementujte vlastní verze predikátů, které pro seznamy počítají standardní seznamové funkce, které znáte z Haskellu. Konkrétně imitujte funkce `head`, `tail`, `last` a `init`.

Příklad 10.2.11: Napište predikát `nth/3`, který vrátí *n*-tý prvek seznamu. Například dotaz `?- nth(4, [5,2,7,8,0], X).` uspěje se substitucí `X = 8`.

Predikáty na seznamech

Definice `append/3` z knihovny:

```
1 append([], YS, YS).
2 append([X|XS], YS, [X|ZS]) :- append(XS, YS, ZS).
```

Příklad 10.2.3: Napište následující predikáty pro práci se seznamy za pomoci vestavěného predikátu `append/3`.

a) `prefix/2` uspěje, jestliže je první argument prefixem seznamu ve druhém argumentu.
b) `suffix/2` uspěje, jestliže je první argument sufixem seznamu ve druhém argumentu.
c) `element/2` uspěje, jestliže je první argument členem seznamu ve druhém argumentu.
d) `adjacent/3` uspěje, jestliže jsou první dva prvky členy seznamu ve třetím argumentu a jsou vedle sebe (v daném pořadí).
e) `sublist/2` uspěje, jestliže je seznam v prvním argumentu podseznamem seznamu v druhém argumentu.