Backtracking, unifikace, aritmetika

Syntaxe logického programu

Term:

- univerzální datová struktura (slouží také pro příkazy jazyka)
- definovaný rekurzivně
- **konstanty:** číselné, alfanumerické (začínají malým písmenem), ze speciálních znaků (operátory)
- **proměnné**: pojmenované (alfanumerické řetězce začínající velkým písmenem), anonymní (začínají podtržítkem)
- **složený term**: funktor, arita, argumenty struktury jsou opět termy

Anatomie a sémantika logického programu

- **Program**: množina predikátů (v jednom nebo více souborech).
- **Predikát** (procedura) je seznam klauzulí s hlavou stejného jména a arity
- **Klauzule**: věty ukončené tečkou, se skládají z hlavy a těla.
  Prázdné tělo mají **fakta**, neprázdné pak **pravidla**, existují také klauzule bez hlavy – direktivy.
  Hlavu tvoří **literál (složený term)**, tělo seznam literálů.
  Literálům v těle nebo v dotazu říkáme **cíle**.
  Dotazem v prostředí interpretu se spouští programy či procedury.

Sémantika logického programu:
procedury ≡ databáze faktů a pravidel ≡ logické formule

Sicstus Prolog minimum I

- **Spuštění interpretu:** V unixu přidáme modul

  ```
  module add sicstus
  ```

  a spustíme příkazem

  ```
  sicstus
  ```

  Pracovním adresářem je aktuální (tam kde byl spuštěn).
  V MS Windows standardně z nabídky Start/Programs nebo pomocí ikony, nastavíme pracovní adresář pomocí File/Working directory, v případě potřeby nastavíme font Settings/Font a uložíme nastavení Settings/Save settings.

Sicstus Prolog minimum II

- **Načtení programu:** tzv. konzultace
  Editor není integrován, takže program editujeme externě ve svém oblíbeném editoru. Pak ho načteme z příkazové řádky v interpretu příkazem

  ```
  ?- consult(jmeno).
  ```

  nebo pomocí zkrácené syntaxe

  ```
  ?- [jmeno].  % (předpokládá se přípona .pl)
  ```

  pokud uvádíme celé jméno případně cestu, dáváme jej do apostrofů

  ```
  ?- ['D:\prolog\moje\programy\jmeno.pl'].
  ```

  V MS Windows lze také pomocí nabídky File/Consult

Sicstus Prolog minimum III

- **Spouštění programů/procedur/predikátů** je zápis dotazů, př.

  ```
  ?- muj_predikat(X,Y).
  ?- suma(1,2,Y), vypis('Vysledek je ',Y).
  ```

  Každý příkaz ukončujeme tečkou.
- **Přerušení a zastavení cyklícího programu**: Ctrl-C
- **Ukončení interpretu příkazem**

  ```
  ?- halt.
  ```


Příklad rodokmen

```
rodic(petr, filip).          muz(petr).
rodic(petr, lenka).          muz(filip).
rodic(pavel, jan).           muz(pavel).
rodic(adam, petr).           muz(jan).
rodic(tomas, michal).        muz(adam).
rodic(michal, radek).        muz(tomas).
rodic(eva, filip).           muz(michal).
rodic(jana, lenka).          muz(radek).
rodic(pavla, petr).
rodic(pavla, tomas).         zena(eva).
rodic(lenka, vera).          zena(lenka).
                             zena(pavla).
                             zena(jana).
                             zena(vera).
otec(Otec,Dite) :- rodic(Otec,Dite), muz(Otec).
```


Backtracking: příklady

V pracovním adresáři vytvořte program rodokmen.pl.
Načtěte program v interpretu (konzultujte).
V interpretu Sicstus Prologu pokládejte dotazy:

- Je Petr otcem Lenky?
- Je Petr otcem Jana?
- Kdo je otcem Petra?
- Jaké děti má Pavla?
- Ma Petr dceru?
- Které dvojice otec-syn známe?

Backtracking: řešení příkladů

Středníkem si vyžádáme další řešení

```
| ?- otec(petr,lenka).
yes
| ?- otec(petr,jan).
no
| ?- otec(Kdo,petr).
Kdo = adam ? ;
no
| ?- rodic(pavla,Dite).
Dite = petr ? ;
Dite = tomas ? ;
no
| ?- otec(petr,Dcera),zena(Dcera).
Dcera = lenka ? ;
no
```

```
| ?- otec(Otec,Syn),muz(Syn).
Syn = filip,
Otec = petr ? ;
Syn = jan,
Otec = pavel ? ;
Syn = petr,
Otec = adam ? ;
Syn = michal,
Otec = tomas ? ;
Syn = radek,
Otec = michal ? ;
no
| ?-
```


Backtracking: příklady II

Predikát potomek/2:

```
potomek(Potomek,Predek) :- rodic(Predek,Potomek).
potomek(Potomek,Predek) :- rodic(Predek,X), potomek(Potomek,X).
```

Naprogramujte predikáty

- prababicka(Prababicka,Pravnouce)
- nevlastni_bratr(Nevlastni_bratr,Nevlastni_sourozenec)

Řešení:

```
prababicka(Prababicka,Pravnouce):-
        rodic(Prababicka,Prarodic),
        zena(Prababicka),
        rodic(Prarodic,Rodic),
        rodic(Rodic,Pravnouce).
```


Backtracking: řešení příkladů II

```
nevlastni_bratr(Bratr,Sourozenec):-
        rodic_v(X,Bratr),
        muz(Bratr),
        rodic_v(X,Sourozenec),
        /* tento test neni nutny,
           ale zvysuje efektivitu */
        Bratr \== Sourozenec,
        rodic_v(Y,Bratr),
        Y \== X,
        rodic_v(Z,Sourozenec),
        Z \== X,
        Z \== Y.
```

```
/* nevhodne umisteni testu -
vypocet "bloudi" v neuspesnych vetvich */
nevlastni_bratr2(Bratr,Sourozenec):-
        rodic_v(X,Bratr),
        rodic_v(X,Sourozenec),
        rodic_v(Y,Bratr),
        rodic_v(Z,Sourozenec),
        Y \== X,
        Z \== X,
        Z \== Y,
        muz(Bratr).
```


Backtracking: prohledávání stavového prostoru

- Zkuste předem odhadnout (odvodit) pořadí, v jakem budou nalezeni potomci Pavly?
- Jaký vliv má pořadí klauzulí a cílu v predikátu potomek/2 na jeho funkci?
- Nahraďte ve svých programech volání predikátu rodic/2 následujícím predikátem rodic_v/2

  ```
  rodic_v(X,Y):-rodic(X,Y),print(X),print('? ').
  ```

  Pozorujte rozdíly v délce výpočtu dotazu nevlastni_bratr(filip,X) při změně pořadí testů v definici predikátu nevlastni_bratr/2

Backtracking: řešení III

```
/* varianta 1a */
potomek(Potomek,Predek):-rodic(Predek,Potomek).
potomek(Potomek,Predek):-rodic(Predek,X),potomek(Potomek,X).

/* varianta 1b - jine poradi odpovedi, neprimi potomci maji prednost */
potomek(Potomek,Predek):-rodic(Predek,X),potomek(Potomek,X).
potomek(Potomek,Predek):-rodic(Predek,Potomek).

/* varianta 2a - leva rekurze ve druhe klauzuli,
na dotaz potomek(X,pavla) vypise odpovedi, pak cykli */
potomek(Potomek,Predek):-rodic(Predek,Potomek).
potomek(Potomek,Predek):-potomek(Potomek,X),rodic(Predek,X).

/* varianta 2b - leva rekurze v prvni klauzuli,
na dotaz potomek(X,pavla) hned cykli */
potomek(Potomek,Predek):-potomek(Potomek,X),rodic(Predek,X).
potomek(Potomek,Predek):-rodic(Predek,Potomek).
```



```
| ?- nevlastni_bratr(X,Y).
petr? petr? petr? petr? eva? petr? jana?
X = filip,
Y = lenka ? ;
petr? pavel? pavel? adam? adam? tomas? tomas? michal? michal? eva? eva? jana?
pavla? pavla? pavla? adam? pavla? pavla? pavla? pavla? pavla? pavla? lenka?
no
| ?- nevlastni_bratr2(X,Y).
petr? petr? petr? petr? eva? eva? petr? eva? petr? petr? petr? jana? eva? petr?
X = filip,
Y = lenka ? ;
petr? petr? petr? petr? eva? jana? petr? eva? petr? petr? petr? jana? jana? pet
jana? pavel? pavel? pavel? pavel? adam? adam? adam? adam?  pavla? pavla? adam?
pavla? tomas? tomas? tomas? tomas? michal? michal? michal? michal? eva? eva? pe
petr? eva? eva? petr? eva? jana? jana? petr? petr? jana? jana? petr? jana? pavl
pavla? adam? adam? pavla? pavla? adam? pavla? pavla? adam? pavla? pavla? pavla?
pavla? pavla? pavla? adam? pavla? pavla? pavla? pavla? lenka? lenka? lenka? ler
no
```


Unifikace:příklady

Které unifikace jsou korektní, které ne a proč?

Co je výsledkem provedených unifikací?

1. a(X)=b(X)
2. X=a(Y)
3. a(X)=a(X,X)
4. X=a(X)
5. jmeno(X,X)=jmeno(Petr,plus)
6. s(1,a(X,q(w)))=s(Y,a(2,Z))
7. s(1,a(X,q(X)))=s(W,a(Z,Z))
8. X=Y,P=R,s(1,a(P,q(R)))=s(Z,a(X,Y))

Neuspěje volání 1) a 3), ostatní ano, cyklické struktury vzniknou v případech 4),7) a 8) přestože u posledních dvou mají levá a pravá strana unifikace disjunktní množiny jmen proměnných.

Mechanismus unifikace I

Unifikace v průběhu dokazování predikátu odpovídá předávání parametrů při provádění procedury, ale je důležité uvědomit si rozdíly. Celý proces si ukážeme na příkladu predikátu suma/3.

```
suma(0,X,X).                        /*klauzule A*/
suma(s(X),Y,s(Z)):-suma(X,Y,Z).     /*klauzule B*/
```

pomocí substitučních rovnic při odvozování odpovědi na dotaz

```
?- suma(s(0),s(0),X0).
```

Mechanismus unifikace II

```
suma(0,X,X). /*A*/                suma(s(X),Y,s(Z)):-suma(X,Y,Z). /*B*/
?- suma(s(0),s(0),X0).
```

1. dotaz unifikujeme s hlavou klauzule B, s A nejde unifikovat (1. argument)

```
suma(s(0),s(0),X0) = suma(s(X1),Y1,s(Z1))
==> X1 = 0, Y1 = s(0), s(Z1) = X0
==> suma(0,s(0),Z1)
```

2. dotaz (nový podcíl) unifikujeme s hlavou klauzule A, klauzuli B si poznačíme jako další možnost

```
suma(0,s(0),Z1) = suma(0,X2,X2)
X2 = s(0), Z1 = s(0)
==> X0 = s(s(0))
X0 = s(s(0)) ;
```

2’ dotaz z kroku 1. nejde unifikovat s hlavou klauzule B (1. argument)

no

Vícesměrnost predikátů

Logický program lze využít vícesměrně, například jako

- výpočet kdo je otcem Petra? `?- otec(X,petr).`
  kolik je 1+1? `?- suma(s(0),s(0),X).`
- test je Jan otcem Petra? `?- otec(jan,petr).`
  Je 1+1 2? `?- suma(s(0),s(0),s( (0))).`
- generátor které dvojice otec-dítě známe? `?-otec(X,Y).`
  Které X a Y dávají v součtu 2? `?- suma(X,Y,s(s(0))).`

... ale pozor na levou rekurzi, volné proměnné, asymetrii, a jiné záležitosti

Následující dotazy

`?-suma(X,s(0),Z).`   `?-suma(s(0),X,Z).`

nedávají stejné výsledky. Zkuste si je odvodit pomocí substitučních rovnic.

Aritmetika

Zavádíme z praktických důvodů, ale aritmetické predikáty již nejsou vícesměrné, protože v každém aritmetickém výrazu musí být všechny proměnné instaciovány číselnou konstantou.

Důležitý rozdíl ve vestavěných predikátech is/2 vs. =/2 vs. =:=/2

is/2: < konstanta nebo proměnná > is < aritmetický výraz >

výraz na pravé straně je nejdříve aritmeticky vyhodnocen a pak unifkován s levou stranou

=/2: < libovolný term > = < libovolný term >

levá a pravá strana jsou unifikovány

"=:="/2 "=\="/2 ">="/2 "=<"/2

< aritmetický výraz > =:= < aritmetický výraz >

< aritmetický výraz > =\= < aritmetický výraz >

< aritmetický výraz > =< < aritmetický výraz >

< aritmetický výraz > >= < aritmetický výraz >

levá i pravá strana jsou nejdříve aritmeticky vyhodnoceny a pak porovnány

Aritmetika: příklady

Jak se liší následující dotazy (na co se kdy ptáme)? Které uspějí (kladná odpověď), které neuspějí (záporná odpověď), a které jsou špatně (dojde k chybě)? Za jakých předpokladů by ty neúspěšné případně špatné uspěly?

1. X = Y + 1
2. X is Y + 1
3. X = Y
4. X == Y
5. 1 + 1 = 2
6. 2 = 1 + 1
7. 1 + 1 = 1 + 1
8. 1 + 1 is 1 + 1
9. 1 + 2 =:= 2 + 1
10. X \== Y
11. X =\= Y
12. 1 + 2 =\= 1 - 2
13. 1 <= 2
14. 1 =< 2
15. sin(X) is sin(2)
16. sin(X) = sin(2+Y)
17. sin(X) =:= sin(2+Y)

Nápověda: ’=’/2 unifikace, ’==’/2 test na identitu, ’=:=’/2 aritmetická rovnost, ’\==’/2 negace testu na identitu, ’=\=’/2 aritmetická nerovnost

Aritmetika: příklady II

Jak se liší predikáty s1/3 a s2/3? Co umí s1/3 navíc oproti s2/3 a naopak?

```
s1(0,X,X).
s1(s(X),Y,s(Z)):-s1(X,Y,Z).
```

```
s2(X,Y,Z):- Z is X + Y.
```

s1/3 je vícesměrný - umí sčítat, odečítat, generovat součty, ale pracuje jen s nezápornými celými čísly
s2/3 umí pouze sčítat, ale také záporná a reálná čísla

Operátory

Definice operátorů umožňuje přehlednější infixový zápis binárních a unárních predikátů, příklad: definice op(1200,Y,':-') umožňuje zápis

```
a:-print(s(s(0))),b,c).
```

pro výraz

```
:-(a,,(print(s(s(0))),,(b,c))).
```

Prefixovou notaci lze získat predikátem display/1. Vyzkoušejte

```
display((a:-print(s(s(0))),b,c)).
display(a+b+c-d-e*f*g-h+i).
display([1,2,3,4,5]).
```

Definice standardních operátorů najdete na konci manuálu.

Závěr

Dnešní látku jste pochopili dobře, pokud víte

- jaký vliv má pořadí klauzulí a cílu v predikátu potomek/2 na jeho funkci,
- jak umisťovat testy, aby byl prohledávaný prostor co nejmenší (příklad nevlastni_bratr/2),
- k čemu dojde po unifikaci X=a(X),
- proč neuspěje dotaz ?- X=2, sin(X) is sin(2).
- za jakých předpokladů uspějí tyto cíle X=Y, X==Y, X=:=Y,
- a umíte odvodit pomocí substitučních rovnic odpovedi na dotazy suma(X,s(0),Z) a suma(s(0),X,Z).