Olomoucký informatický korespondenční seminář – Olinx

Programovací jazyk PROLOG, 1. díl

Miroslav Kolařík

olinx.inf.upol.cz

Úvod

V tomto semináři se budeme věnovat programovacímu jazyku PROLOG, který je typickým zástupcem programovacího stylu založeného na principech logického programování. V prvním dílu si PROLOG zběžně představíme a naučíme se jeho základní syntaxi. Řekneme si, co jsou fakta a pravidla a funkčnost prvních příkladů otestujeme pomocí vhodně formulovaných dotazů. Několikrát použijeme princip rekurze. V příštích dílech se zaměříme na složitější datové struktury, na obtížnější úlohy a seznámíme se s některými pokročilejšími konstrukcemi, které lze v PROLOGu výhodně používat.

O PROLOGu

PROLOG je logický programovací jazyk. Vznikl ve Francii (v Marseille) začátkem 70. let minulého století. Byl vytvořen Alainem Colmeranerem ve spolupráci s Philippem Rousselem, na základě procedurálního výkladu Hornovy klauzule od Roberta Kowalskiho. Jméno „PROLOG“ vzniklo jako zkratka z „PROgrammation à LOGic“ (francouzsky „programování v logice“).

PROLOG je interpretační (neprocedurální) jazyk. Patří mezi deklarativní programovací jazyky – potlačuje imperativní[1] složku. PROLOG je využíván především v oboru umělé inteligence a v počítačové lingvistice (obzvláště pro zpracování přirozeného jazyka, pro nějž byl původně navržen).

PROLOG je založen na predikátové logice (prvního řádu); konkrétně se omezuje na takzvané Hornovy klauzule[2]. Základními využívanými přístupy jsou unifikace (speciální substituce), rekurze (využívá principu sebeopakování, viz dále) a backtracking (metoda prohledávání do hloubky).

O logickém programování

Základní koncepci logického programování vyjadřuje následující dvojice „rovností“:

program = množina axiomů,
výpočet = konstruktivní důkaz uživatelem zadaného cíle,

nebo volněji: program je souborem tvrzení, kterými programátor (uživatel, expert) popisuje určitou část okolního světa. Výpočet nad daným programem, který je iniciován zadáním dotazu, je hledání důkazu dotazu z daného souboru tvrzení (viz dále).

Logické programování je jedním z paradigmat programování (programovacích stylů). Uveďme nyní základní rysy, kterými se logické programování zásadně odlišuje od ostatních programovacích paradigmat:

- programátor specifikuje, co se má vypočítat, a ne jak se to má vypočítat a kam uložit mezivýsledky
- nejsou potřeba příkazy pro řízení běhu výpočtu ani pro řízení toku dat, nejsou potřeba příkazy cyklů, větvení, ani přiřazovací příkaz
- neexistuje rozdělení proměnných na vstupní a výstupní
- nerozlišuje se mezi daty a programem.

Dále je typické, že

- logický program je konečná množina formulí speciálního tvaru (v PROLOGu to jsou Hornovy klauzule)
- výpočet je zahájen zadáním dotazu, což je logická formule, kterou zadává uživatel
- cílem výpočtu je najít důkaz potvrzující, že dotaz logicky vyplývá (je dokazatelný) z logického programu (konstruktivnost)
- pokud je takto zjištěno, že dotaz z programu vyplývá, výpočet končí a uživateli je oznámeno `true` s hodnotami případných proměnných, které se v dotazu vyskytují
- pokud není zjištěno, že dotaz z programu vyplývá, výpočet končí a uživateli je oznámeno `false`
- může se stát, že výpočet neskončí (například uvázne v nekonečném cyklu).

Abychom si osvojili základní principy logického programování, budeme se učit programovací jazyk PROLOG.

Základem PROLOGu je databáze klauzulí, které lze dále rozdělit na fakta a pravidla, nad kterými je možno klást dotazy formou tvrzení, u kterých PROLOG zhodnocuje jejich pravdivost (dokazatelnost z údajů obsažených v databázi). Nejjednoduššími klauzulemi jsou fakta, která pouze vypovídají o vlastnostech objektu nebo vztazích mezi objekty. Složitějšími klauzulemi jsou pravidla, která umožňují (pomocí implikace) odvozovat nová fakta. Zapisují se ve tvaru

```
hlava :- telo.
```

kde `hlava` definuje odvozovaný fakt, `telo` podmínky, za nichž je pravdivý. Tělo pravidla obsahuje jeden či více cílů. Pokud se interpretu podaří odvodit, že je tělo pravdivé, ověří tím pravdivost hlavy.

První příklad

Víme, že se dá z Prahy letět přímo do Paříže a do Lyonu. Dále víme, že se dá přímo letět z Paříže do Marseille a z Marseille přímo do Vídně. Letecká spojení mezi městy jsou buď přímá nebo přes nějaká další města.

Uvedené informace nyní přepíšeme tak, aby syntaxe odpovídala PROLOGu. Dostaneme tzv. formalizovanou znalostní bázi (jednoduchý logický program):

```
direct(praha,pariz).
direct(praha,lyon).
direct(pariz,marseille).
direct(marseille,viden).
connection(X,Y) :- direct(X,Y).
connection(X,Z) :- direct(X,Y), connection(Y,Z).
```

[1] Imperativní paradigma popisuje, jak vyřešit problém. Deklarativní paradigma popisuje, co je problém.

[2] Hornova klauzule je formule (ve tvaru disjunkce atomických formulí), která obsahuje nejvýše jednu nenegovanou atomickou formuli (ostatní jsou právě jednou negované).

První čtyři řádky jsou fakta, předposlední a poslední řádek jsou pravidla. Všimněme si, že jak fakta, tak pravidla jsou zakončena tečkou, a že před levou závorkou nikdy není mezera. Jelikož jsou velká počáteční písmena vyhrazena pro proměnné, píšeme zde města s malými počátečními písmeny.

Nainstalujme si nyní SWI-PROLOG. Je to jedna z mnoha aplikací, ve které můžeme vytvářet, překládat a testovat programy psané v PROLOGu. Autorem SWI-PROLOGu je Jan Wielemaker z University of Amsterdam. Zdarma lze stáhnout (verze pro Unix, MS-Windows) ze stránky[3]:

http://www.swi-prolog.org

Do právě staženého programu můžeme načítat programy v PROLOGu (a to přes nabídku: `File` → `Consult ...`)[4]. Ihned po načtení konkrétního souboru lze zadávat konkrétní dotazy, na které nám interpret PROLOGu obratem odpoví. Zkopírujte (nebo přepište) výše uvedených šest řádků do (obyčejného) textového souboru a po uložení (typicky s příponou `.pl`) soubor načtěte do aplikace SWI-PROLOG. Nyní je PROLOG připraven na dotazy, které se zapisují za symboly „`?-`“[5]. Nyní vyzkoušíme zadat některé jednoduché dotazy.[6] Níže uvedené dotazy vyzkoušejte na počítači. Nebojte se vyzkoušet i chybně položené dotazy[7], ať víte, jak na ně interpret PROLOGu zareaguje.

```
?- direct(praha,lyon).
```

Jelikož je tento fakt součástí našeho programu, odpoví PROLOG:

```
true.
```

Nyní položíme podobný dotaz:

```
?- direct(lyon,praha).
```

Tento dotaz z našeho programu nijak nevyplývá, proto na něj PROLOG odpoví:

```
false.
```

Nyní zadáme dotaz s proměnnou `X`:

```
?- direct(praha,X).
```

Na tento dotaz můžeme v souladu s naším programem dostat dvě různé (pravdivé odpovědi):

```
X=pariz;
X=lyon.
```

Pomocí středníku požadujeme další odpovědi, které můžeme na náš dotaz obdržet. Pokud bychom místo středníku stiskli klávesu „`Enter`“, seznam případných dalších odpovědí nedostaneme; můžeme však položit nový dotaz. Všimněme si, že při zadání dotazu s proměnnou `X` dostáváme jako odpověď možné hodnoty této proměnné.

Zkuste odhadnout a poté si sami ověřte, jaké odpovědi dá PROLOG na dotaz obsahující dvě různé proměnné `X` a `Y`.

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

Složitější a zajímavější je binární relace `connection`, která je definována pomocí dvou pravidel. Pravidlo

```
connection(X,Y) :- direct(X,Y).
```

nám říká, že spojení mezi místy `X` a `Y` může být přímé a pravidlo

```
connection(X,Z) :- direct(X,Y), connection(Y,Z).
```

nám říká, že spojení mezi místy `X` a `Z` může vést také přes místo `Y`, případně přes další místa, mezi kterými je přímé spojení.

Na dotaz

```
?- connection(praha,viden).
```

tak dostaneme odpověď

```
true.
```

neboť z Prahy vedou přímá spojení (přes jednotlivá města) až do Vídně, konkrétně přes Paříž a Marseille. V případě relace `connection` používáme princip rekurze (silný a v PROLOGu často používaný nástroj), který ještě několikrát výhodně použijeme.

2. PŘÍKLAD

PROLOG je programovací jazyk pro symbolické, nenumerické výpočty. Zejména je vhodný pro řešení problémů, které obsahují objekty a vztahy (relace) mezi objekty. V následujícím příkladu budeme popisovat rodinné vztahy (mezi jistými osobami). Fakt, že Tom je rodičem Roberta zapíšeme v PROLOGu takto:

```
rodic(tom,robert).
```

Zde jsme zvolili `rodic` jako jméno relace; `tom` a `robert` jsou jejími argumenty. Jména píšeme s malými počátečními písmeny, aby je překladač PROLOGu nepovažoval za proměnné. Podobně můžeme pomocí faktů definovat další vztahy.

V následujícím příkladu budeme uvažovat jednoduchou databázi faktů zachycující hierarchii příbuzenských vztahů. Tato databáze obsahuje fakta zapsaná pomocí binární relace `rodic`, definující vztah: člověk `X` je rodičem člověka `Y`. V databázi jsou dále použity dvě unární relace `muz` a `zena` definující pohlaví dané osoby.[8]

```
rodic(pavla,robert).
rodic(tom,robert).
rodic(tom,liza).
```

[3] Nebo se podívejte do repozitářů své distribuce.

[4] Pokud máte swipl puštěný v konzoli (a ne v okenním prostředí), tak lze načtení souboru „file.pl“ provézt pomocí „[file]“.

[5] Každý dotaz musí být ukončen tečkou. Klávesou „Enter“ se aktivuje vyvolání odpovědi.

[6] Po vložení dotazu PROLOG spustí inferenční mechanismus a snaží se najít odpověď na dotaz. Odpověď je buďto nalezena (a zodpovězena) nebo nikoliv (například proto, že se interpret dostal do nekonečného cyklu).

[7] Například dotaz se špatným počtem argumentů, nebo s chybějící závorkou, nebo s neznámým relačním symbolem a tak podobně.

[8] Unární relací rozumíme relaci s jedním argumentem, tedy konkrétní vlastnost nějakého objektu. Binární relací rozumíme relaci popisující vztah mezi dvěma objekty, tedy relaci se dvěma argumenty. Podobně ternární relací rozumíme relaci se třemi argumenty. Například ternární relace `stred(A,S,B)` může být interpretována tak, že bod `S` je středem úsečky určené body `A` a `B` . Obecně n-ární relací rozumíme relaci s n argumenty, tedy relaci, která popisuje vztah n objektů, kde n je přirozené číslo. Dále doplňme, že každá relace se skládá z identifikátoru, za nímž následuje výraz v kulatých závorkách, který obsahuje příslušný počet argumentů.

```
rodic(tom,vilem).
rodic(robert,anna).
rodic(robert,patricie).
rodic(patricie,jan).

zena(pavla).
zena(liza).
zena(anna).
zena(patricie).
muz(tom).
muz(robert).
muz(jan).
```

Lze vyčíst, že Pavla je rodičem Roberta. Dále, že Tom je rodičem Roberta, Lízy a Viléma a tak dále. Všimněme si, že databáze například nedefinuje, kdo je druhým rodičem Jana, ačkoliv bychom jeho existenci mohli intuitivně předpokládat.

Následujícím dotazem s proměnnou `X` snadno zjistíme, kdo všechno je potomkem Toma.

```
?- rodic(tom,X).
```

```
X=robert;
X=liza;
X=vilem.
```

Vyzkoušíme si nyní dotaz[9] složený ze dvou částí, které mají být splněny současně:

```
?- rodic(tom,X), zena(X).
```

Ptáme se opět na potomky Toma, ale jen takové, které mají ženské pohlaví (ptáme se tedy na dcery Toma). Tentokrát dostaneme jedinou odpověď:

```
X=liza.
```

K programu postupně doplníme pravidla. Nejprve program rozšíříme zavedením binární relace `potomek`, jako inverzi k relaci `rodic`. Mohli bychom postupně dodefinovat nová fakta, například

```
potomek(robert,pavla).
```

Mnohem elegantnějším řešením je však využít již zavedené relace `rodic` a definovat k ní inverzní relaci `potomek`. V PROLOGu toto pravidlo zapíšeme následovně:

```
potomek(Y,X) :- rodic(X,Y).
```

V pravidlu figurují dvě proměnné; můžeme ho číst takto: pro každé `X` a `Y`, jestliže `X` je rodičem `Y`, pak `Y` je potomkem `X`. Mezi fakty a pravidly si můžeme všimnout podstatného rozdílu. Každý fakt, například

```
rodic(tom,liza).
```

je něco, co je vždy, nepodmíněně, pravdivé. Na druhou stranu, pravidla specifikují věci, které jsou pravdivé, jestliže je splněna nějaká podmínka. Říkáme, že pravidla mají

- podmínkovou část (pravá strana pravidla), tzv. tělo pravidla
- důsledkovou část (levá strana pravidla), tzv. hlava pravidla.

Například, v pravidle

```
potomek(Y,X) :- rodic(X,Y).
```

je hlavou `potomek(Y,X)` a tělem `rodic(X,Y)`. Pokud je podmínka `rodic(X,Y)` pravdivá, pak je jako logický důsledek pravdivá i relace `potomek(Y,X)`.

S přidaným pravidlem pro potomky nyní na dotaz

```
?- potomek(jan,patricie).
```

obdržíme odpověď `true`, ačkoliv relaci `potomek` mezi fakty nenajdeme.

Dále definujeme jednoduchým pravidlem unární relaci `mitdite(X)`, která bude pravdivá v případě, že proměnná `X` je rodičem libovolného dítěte. Jelikož při definici těla pravidla na daném dítěti nezáleží, použijeme takzvanou anonymní proměnnou, kterou značíme podtržítkem[10], tedy symbolem „_“.

```
mitdite(X) :- rodic(X,_).
```

Unární relaci `mitdite` si sami otestujte.

Definujme další tři pravidla:

```
matka(X,Y) :- rodic(X,Y), zena(X).
prarodic(X,Z) :- rodic(X,Y), rodic(Y,Z).
sestra(X,Y) :-
  rodic(Z,X), rodic(Z,Y), zena(X), not(X=Y).
```

Čárka mezi dvěma podmínkami v těle pravidla znamená konjunkci obou podmínek, tedy to, že obě podmínky musí být současně pravdivé.

Jak vidíme relace `matka` je založena na následujícím tvrzení se dvěma proměnnými: pro každé[11] `X` a `Y`, `X` je matkou `Y`, jestliže `X` je rodičem `Y` a `X` je žena.

Relace `prarodic` je založena na následujícím tvrzení se třemi proměnnými: pro každé `X`, `Y` a `Z`, `X` je prarodičem `Z`, jestliže `X` je rodičem `Y` a zároveň `Y` je rodičem `Z`.

Na dotaz, kdo jsou prarodiče Patricie, obdržíme (v souladu s naším programem) dvě odpovědi:

```
?- prarodic(X,patricie).

X=pavla;
X=tom;
false.
```

Pokud by nás zajímalo, kdo všechno jsou prarodiče a nepotřebovali bychom znát jména vnoučat, využijeme dotaz s anonymní proměnnou:

```
?- prarodic(X,_).

X=pavla;
X=tom;
X=robert;
false.
```

Pravidlo pro relaci `sestra` je trochu delší, a tak jsme jej, pro lepší čitelnost rozdělili na dva řádky (hlavu na prvním a

[9] Pokud chcete vyvolat předchozí dotazy, stiskněte několikrát klávesu „šipka nahoru“.

[10] Přesněji řečeno, která začíná podtržítkem.

[11] Proměnné jsou vždy brány s obecnými kvantifikátory, tedy v nejširším možném rozsahu.

tělo s malým odsazením na druhém řádku). Toto rozdělení se běžně používá. Na konci těla pravidla používáme předdefinovanou relaci `not(X=Y)`, která je splněna, právě když proměnná `X` se nerovná proměnné `Y`. Definovat tuto relaci není těžké (a lze to provést vícero způsoby), my se k tomu v některém z dalších dílů semináře vrátíme. Pro tuto chvíli však předpokládejme, že je tato binární relace PROLOGu známa.

Relace `sestra` je založena na následujícím tvrzení: pro každé `X` a `Y`: `X` je sestrou `Y`, jestliže `X` a `Y` mají stejného rodiče, `X` je žena a `X` je různé od `Y`. Jak si můžeme všimnout, v pravidle je stejný rodič označen proměnnou `Z`. Položením dotazu zjistěme, kdo všechno je sestrou Patricie:

```
?- sestra(X,patricie).

X=anna;
false.
```

Vyzkoušejte si, že kdybychom v pravidle neměli binární relaci `not(X=Y)`, dostali bychom k dotazu následující odpověď:

```
?- sestra(X,patricie).

X=anna;
X=patricie;
false.
```

což by bylo očividně špatně, neboť člověk nemůže být sám sobě sestrou. Promyslete si, proč tento příklad zdůvodňuje potřebu relace `not(X=Y)`.

Přidejme si nyní k našemu programu ještě dvě další pravidla, která definují binární relaci `predek`. První pravidlo definuje přímé (bezprostřední) předky a můžeme ho popsat takto:
Pro každé `X` a `Z`: `X` je předkem `Z`, jestliže `X` je rodičem `Z`.

Druhé pravidlo definuje nepřímé předky. Řekneme, že `X` je nepřímým předkem některého `Z`, jestliže existuje rodičovský řetězec lidí mezi `X` a `Z`. Toto pravidlo definujeme pomocí sebe sama (využijeme tak rekurzi):
Pro každé `X` a `Z`: `X` je předkem `Z`, jestliže existuje `Y` takové, že

(1) `X` je rodičem `Y`
(2) `Y` je předkem `Z`.

Následují obě pravidla přepsaná do PROLOGu:

```
predek(X,Z) :- rodic(X,Z).
predek(X,Z) :- rodic(X,Y), predek(Y,Z).
```

Promyslete si, proč je relace `predek` korektně zavedena. Nakreslete si k tomu odpovídající obrázek. Funkčnost pravidla si vyzkoušíme na dotazu, který má za cíl zjistit, komu je Pavla předkem.

```
?- predek(pavla,X).

X=robert;
X=anna;
X=patricie;
X=jan;
false.
```

Rekurze znamená sebeopakování. Velmi často se používá v matematice a informatice. V programování rekurze představuje opakované vnořené volání stejné funkce (podprogramu); hovoříme o tzv. rekurzivní funkci. Nedílnou součástí rekurzivní funkce musí být ukončující podmínka určující, kdy se má rekurze zastavit. Po každém kroku volání sebe sama musí dojít ke zjednodušení problému. Pokud nenastane koncová situace, provede se rekurzivní krok. Rekurze je jedním ze základních principů programování v PROLOGu a je v něm často využívána.

Závěrem

PROLOG je programovací jazyk zvláště vhodný pro problémy, které zahrnují objekty (zejména strukturované objekty) a vztahy mezi nimi. Silným nástrojem pro vytváření programů v PROLOGu je rekurze. Díky těmto vlastnostem je PROLOG výkonným jazykem pro umělou inteligenci a nečíselné programování obecně.

Programy v jazyku PROLOG se skládají z výrazů, které tvoří fakta a pravidla. Zvláštní výrazy, které nejsou přímou součástí programů, jsou dotazy, někdy nazývané cíle. Programy v PROLOGu slouží k vyjádření (popisu) naší znalostní báze. Programy píšeme v roli „programátorů“, pomocí cílů aktivujeme výpočet, přičemž cíle zadáváme v roli „uživatelů“ vytvořeného programu.

Programátor není zbaven zodpovědnosti za to, jak bude výpočet probíhat. Výpočet (tj. logické dokazování) je řízen PROLOGovským překladačem a programátor musí znát pravidla (alespoň chce-li psát efektivní programy), kterými se výpočet řídí a v souladu s nimi program v PROLOGu vytvářet. To, jak PROLOG nachází odpovědi na zadané dotazy, se dozvíme v příštích dílech tohoto semináře.

Úkoly k odevzdávání

Následují úkoly, které se odevzdávají přes web semináře Olinx,

https://olinx.inf.upol.cz/

Úkoly stačí odevzdat ve formě jednoho jednoduchého textového souboru, ve kterém jsou napsány řešení ke všem úkolům. Pro tvorbu řešení se předpokládá používání aplikace SWI-PROLOG. Pokud jsou vyžadovány dotazy, nezapomeňte je k řešení přidat.

Můžete využít komentáře. Řádkový komentář začíná symbolem %. Tedy vše, co je za znakem „%“ je až do konce řádku považováno za komentář. Blokový (víceřádkový) komentář začíná dvěma symboly /* a končí dvěma symboly */. Tedy vše, co je mezi symboly „/*“ a „*/“ je bráno jako komentář a při překladu programu je tato část ignorována. Následuje malá ukázka použití komentářů:

```
/* 1. ukol
   Zvirata */

   velky(medved).          %Medved je velky.
   ...
   seda(mys).              %Mys je seda.
   ...
```

Úkol 1 *6 bodů*

Formalizujte následující slovní popis a přepište jej do PROLOGu:
„Medvěd je velký, slon je velký. Kočka je malá, myš je malá. Medvěd je hnědý, kočka je černá, slon je šedý, myš je šedá. Vše černé a vše hnědé je tmavé.“
Poté (v PROLOGu) napište dotaz, kterým zjistíte, kdo všechno je šedý a druhý dotaz, kterým zjistíte, kdo je tmavý a současně velký.

Úkol 2 *13 bodů*

Uvažme následující PROLOGovský program:

```
rodic(pavla,robert).
rodic(tom,robert).
rodic(tom,liza).
rodic(robert,anna).
rodic(robert,patricie).
rodic(patricie,jan).

zena(pavla).
zena(liza).
zena(anna).
zena(patricie).
muz(tom).
muz(robert).
muz(jan).

potomek(Y,X) :- rodic(X,Y).
matka(X,Y) :- rodic(X,Y), zena(X).
prarodic(X,Z) :- rodic(X,Y), rodic(Y,Z).
sestra(X,Y) :-
  rodic(Z,X), rodic(Z,Y), zena(X), not(X=Y).
predek(X,Z) :- rodic(X,Z).
predek(X,Z) :- rodic(X,Y), predek(Y,Z).
```

K právě uvedenému programu doplňte čtyři fakta obsažená ve větě „Jan je rodičem chlapce Šimon a dívky Ester.“ a definujte tři pravidla:

- binární relaci `deda`,
- binární relaci `bratr`,
- binární relaci `vnouce`.

Poté položte čtyři dotazy:

- Koho je Šimon vnouče?
- Kdo je bratr Lízy?
- Kdo všechno je dědou?
- Kdo všechno jsou předci Ester?

Úkol 3 *6 bodů*

Uvažme následující program v PROLOGu:

```
funkceF(0,1).
funkceF(N,X) :-
  N >= 1,
  M is N-1,
  funkceF(M,R),
  X is R*N.
```

Otestujte několika dotazy, jaké hodnoty vrací funkceF definovaná výše. Pro testování používejte pouze nezáporná celá čísla, pro která je funkceF definována (jedním faktem a jedním pravidlem). Uveďte o jakou funkci se jedná. Význam neznámých symbolů intuitivně odhadněte (nebo dohledejte). Zjistěte, zda si SWI-PROLOG poradí i se vstupní hodnotou rovnou deseti tisícům, tedy, zda dokáže odpovědět na dotaz

```
funkceF(10000,X).
```

Literatura

[1] I. Bratko, 2011. *PROLOG Programming for Artificial Intelligence (4th Edition)*. Pearson Education Canada. ISBN 9780321417466.