Prolog aneb programování netradičně

Eva Špačková

Chtěla bych předeslat, že doporučuji číst tento článek až po přednášce — nevím, zda je úplně srozumitelný i začátečníkům. Byl psán s tím úmyslem, že až za pár let o Prologu někde uslyšíte, dokážete si po jeho přečtení vzpomenout, co to ten Prolog vlastně byl zač.

Většina dnes běžně užívaných programovacích jazyků má jeden rys společný – popisujeme v nich (víceméně přesně), *jak* se má daná úloha vyřešit. Říkáme jim *procedurální jazyky*. Základem, na kterých tyto jazyky stojí, je pojem proměnné jako označení místa v paměti a přiřazovacího příkazu, který umožní nějak získanou hodnotu uložit do vhodné proměnné. V tom, co je nad těmito základy vystavěno, se *Basic, assembler, Fortran, Algol, Pascal, Ada, C* a *C++* (abychom vyjmenovali jen ty nejznámější) liší velmi podstatně, daný rámec však nepřekračují.

Existují však i programovací jazyky založené na jiných principech. V *neprocedurálním* programovacím jazyce specifikujeme *problém*, který chceme řešit – ne nutně přesně *způsob*, jakým se má řešit. Historicky nacházejí takové jazyky uplatnění především v oblasti tzv. *umělé inteligence*, která se mimo jiné vyznačuje převahou úloh nenumerického charakteru.

Chtěla bych představit jeden z neprocedurálních jazyků, nejznámějšího představitele tzv. *logického programování* – Prolog. V tomto článku si na příkladě ukážeme, jak se Prolog liší od procedurálního jazyka Pascal.

```
permutace([],[]).
permutace(Seznam,[X|Zb_permutace])
   :- member(X,Seznam),
      del(X,Seznam,Zb_seznamu),
      permutace(Zb_seznamu,Zb_permutace).
```

Predikát `member` říká, že X je prvkem Seznamu. Predikát `del`, že Zb_seznamu vznikne tak, že ze Seznamu vypustíme prvek X. Celý program si lze tedy vyložit takto: Milý Prologu, věz, že permutace prázdného seznamu je prázdný seznam. Permutace neprázdného seznamu vypadá takto: její první prvek je libovolný prvek ze Seznamu a zbytek permutace je permutace zbytku seznamu.

Ještě bych měla říci, jak se s takovým „programem“ zachází. Dáme ho „přechroustat“ Prologu, který se pravidla výrobě permutací naučí, a vyzve nás, abychom mu zadali dotaz. Tak se zeptáme: řekni nám, Prologu, jaké permutace můžeme dostat ze seznamu 1,2,3,4 – jinými slovy napíšeme

```
permutace([1,2,3,4],P).
```

A tak nám tedy Prolog odpoví:

```
P=[1,2,3,4]
```

To je sice pravda, ale asi nám to nestačí. Poprosíme tedy Prolog, jestli by neměl v záloze něco dalšího (to se dělá tak, že napíšeme středník). A Prolog na to:

```
P=[1,2,4,3]
```

Budeme-li nadále požadovat další možnosti, bude Prolog generovat další permutace – 1,3,2,4 1,3,4,2 ... 4,3,2,1. Řekneme-li si nyní o další permutaci, Prolog odpoví:

```
no
```

Jestli jste tomu rozuměli tak, že Prolog podle námi zadaných pravidel žádné permutace neumí vyrobit, tak to je správně.

Co myslíte, že se stane, když se zeptám

```
permutace([cervena, bila, modra],P).
```

Bude to fungovat? Bude. Když jsme přece Prolog učili, co je to permutace, nikde jsme neřekli, ze to, co se permutuje, musí být zrovna přirozená čísla.

A co se stane teď?

```
permutace([a,b,c,d,e],[b,e,c,a,d]).
```

Nic překvapivého - Prolog prostě odpoví

```
yes
```

Ptali jsme se přece, jestli je b,e,c,a,d permutací a,b,c,d,e. Stejně tak na

```
permutace([b,f,l,m,p,s,v,z],[h,ch,k,r,d,t,n]).
```

odpoví `no`. Tuto odpověď je třeba chápat tak, že žádná taková pravidla, z kterých by se dalo dokázat, že b,f,l... je permutace h,ch,k... prostě Prolog nezná.

Teď se podívejte, jak vypadá program na vygenerování všech permutací v Pascalu:

```
program permutace;
uses crt;

const max = 10;

type pole = array [1..max] of boolean;
     vypis = array[1..max] of byte;
     rozsah = 1..max;
var n,p : rozsah;
    pouzite : pole;
```

```
    vysledek : vypis;
procedure permutuj(index:rozsah;kde:pole;kolikate:rozsah;vypsat:vypis);

var i : rozsah;
    hotovo :boolean;
     begin

if not kde[index] then
   begin kde[index]:=true;
       vypsat[index]:=kolikate;
       if kolikate=n then begin for i:=1 to n do
                       write(vypsat[i]:3);
                       writeln;
                       end
       else begin
       inc(kolikate);
       for i:=1 to n do permutuj(i,kde,kolikate,vypsat);
   end; end;
end;
BEGIN
  clrscr;
  write('Zadej kolik cisel permutovat '); Readln(n);
  for p:=1 to n do pouzite[p]:=false;
  for p:=1 to n do permutuj(p,pouzite,1,vysledek);
  repeat until keypressed;
END.
```

Tak, a teď řekněte – co se vám líbí víc? :-) Jestli ten prologovský způsob, tak jste mi udělali radost. Pochopitelně (přiznám se bez mučení) jsem vybrala takový příklad, který lze v Prologu napsat velmi přirozeně. Ne všechny jsou samozřejmě takové. Obecně se v Prologu velmi dobře programuje rekurze a procházení stavovým prostorem pomocí backtrackingu (jestli nevíte co to je, tak to klidně pusťte z hlavy).

Doufám že si z přednášky (a také z tohoto článku) odnesete alespoň pocit, že Prolog je sympatický jazyk. A jestli ne, tak se také nic neděje, třeba vás o tom někdy v budoucnu přesvědčí někdo jiný :-)

Poděkování.

- *Rudolfu Krylovi*, že napsal taková pěkná skripta o Prologu
- *Šárce Štěpánové*, která mi ušetřila práci a napsala program v Pascalu