Gramatyki (2)

Definiowanie języków programowania
Składnia i semantyka instrukcji

Zagadnienia

- Co to jest semantyka programu?
- Składnia podstawowych konstrukcji Pascala
- Semantyka podstawowych konstrukcji Pascala

Semantyka programu

- Programy coś robią. To coś, to jest właśnie semantyka. Jak to coś określić?
- Programy operują na zmiennych. Zmieniają ich wartości w taki sposób, aby końcowe wartości były rozwiązaniem jakiegoś zadania.
- Zazwyczaj na początku działania programu inicjalizujemy wartości zmiennych, wczytując je z pliku, albo wprowadzamy je za pomocą klawiatury, myszki czy innego urządzenia wejściowego.
- Pod koniec działania programu uzyskane wartości są przekazywane na wyjście (ekran, głośnik, plik).

Semantyka programu

- Stanem komputera nazwiemy zestaw wartości wszystkich jego zmiennych. Uwaga: niektóre zmienne mogą nie mieć wartości – to jest sytuacja normalna. Brak wartości jest legalną wartością!
- Działanie programu polega na przekształceniu jednego stanu komputera w drugi.
- Określenie semantyki instrukcji, to podanie, w jaki sposób instrukcje te zmieniają stan programu.
- Ponieważ instrukcje zdefiniujemy za pomocą gramatyki, więc w tym samym języku wyrazimy ich semantykę

Semantyka programu – sytuacje szczególne

- Nie zawsze program kończy działanie w sposób poprawny. Są dwie typowe sytuacje, gdy tak się nie dzieje:
  - błąd
  - zapętlenie
- W pierwszym przypadku mamy do czynienia z wykonaniem nieprawidłowej instrukcji, której semantyka jest nieokreślona, np. próbą dzielenia przez zero, próbą wczytania danych z pliku w sytuacji, gdy tych danych tam nie ma, próbą odwołania się do zakazanego obszaru pamięci i wiele innych.
- W drugim przypadku mamy do czynienia z nieskończonym wykonaniem: choć program nie robi nieczego błędnego, nie zakończy swojego działania i pożytek z niego będzie niewielki.

Semantyka programu - notacja

- Semantykę będziemy określać za pomocą notacji

  V [<instrukcja>] = V’

  co będzie oznaczało, że jeśli przy wartościowaniu V wykonamy instrukcję <instrukcja>, to końcowe wartościowanie będzie równe V’.

- Rezerwujemy sobie dwa symbole na oznaczenie nietypowych wartościowań: błąd oraz zapętlenie

Składnia instrukcji

- <instrukcja>::=
  - <instrukcja pusta> |
  - <instrukcja przypisania> |
  - <instrukcja złożona> |
  - <instrukcja warunkowa> |
  - <instrukcja pętli> |
  - <instrukcja czytania> |
  - <instrukcja pisania>

Składnia i semantyka instrukcji pustej

- <instrukcja pusta> ::= ε

Semantyka instrukcji pustej:

V [<instrukcja pusta>] = V

Czyli instrukcja pusta nie zmienia wartościowania.

Składnia i semantyka instrukcji przypisania

- <instrukcja przypisania> ::=

  <zmienna> := <wyrażenie>

- V [x := E] = V', gdzie dla każdej zmiennej z

$$V'(z) = \begin{cases} V(z) \text{ gdy } z \neq x \\ V(E) \text{ gdy } z = x \end{cases}$$

Przykład

- Załóżmy, że mamy trzy zmienne: x,y,z.
- V(x) = 3, V(y)=?, V(z)=7, co w skrócie zapiszemy jako x=3,y=?,z=7
- Dla tego wartościowania wykonujemy instrukcję przypisania

  y:=x+z-5

Nowe wartościowanie V' powstałe po wykonaniu tej instrukcji przypisania jast następujące:

V'(x) = 3, V'(y) = 5, V'(z)=7, czyli x=3,y=5,z=7

- Uwaga: Sensowne byłoby przypisanie x:=x+1, które doprowadziłoby do wartościowania (x=4,y=?,z=7), ale przypisanie y:=x+y nie zmieniłoby wartościowania, bowiem wartość wyrażenia po prawej stronie byłaby nieokreślona. Natomiast przypisanie x:=x+y doprowadziłoby do wartościowania (x=?, y=?, z=7)

Składnia i semantyka instrukcji złożonej

- <instrukcja złożona> ::= begin <ciąg instrukcji>end
- <ciąg instrukcji> ::= <instrukcja> | <instrukcja>;< ciąg instrukcji>

Napisy, jakie uzyskujemy z tej produkcji są tego typu:

begin <instrukcja>;<instrukcja>;...;<instrukcja> end

Semantyka:

V [begin <instrukcja> end] = V[<instrukcja>]

V[begin<instrukcja>;<ciąg instrukcji>end] =

(V[<instrukcja>])[begin <ciąg instrukcji> end]

Zatem semantyka ciągu instrukcji jest opisana indukcyjnie za pomocą semantyki pierwszej instrukcji z ciągu i całej reszty. Jest równa semantyce tej reszty dla wartościowania, które powstaje po wykonaniu pierwszej instrukcji

Przykład instrukcji złożonej

```
{x=3, y=?, z=7}
begin
  y:=x + z – 5 ;   {x=3, y=5, z=7}
  z:=z – x + 1;     {x=3, y=5; z=5}
  y:= z* x
end
{x=3, y=15, z=5}
```

Przykład instrukcji złożonej (2)

Gdybyśmy jednak wykonali jeszcze jedno działanie

```
{x=3, y=?, z=7}
begin
  y:=x + z – 5 ;   {x=3, y=5, z=7}
  z:=z – x + 1;     {x=3, y=5; z=5}
  y:= z* x;         {x=3, y=15, z=5}
  z := z div (x*z-y)
end
```

{błąd}

Instrukcja warunkowa

- <instrukcja warunkowa> ::= <instrukcja warunkowa bez else'a> | <instrukcja warunkowa z else>
- <instrukcja warunkowa z else> ::=
  if <warunek> then <instrukcja1> else <instrukcja>
- <instrukcja warunkowa bez else'a> ::=
  if <warunek> then <instrukcja>
- <warunek> ::= <wyrażenie logiczne>
- <instrukcja1> ::= <instrukcja pusta> | <instrukcja przypisania> | <instrukcja złożona> |<instrukcja warunkowa z else> | <instrukcja pętli> | <instrukcja czytania> | <instrukcja pisania>

Uwaga: <instrukcja1> to są wszystkie instrukcje poza instrukcją warunkową bez else'a. Dopuszczenie tego rodzaju instrukcji doprowadziłoby do gramatyki niejednoznacznej!

Wyrażenia logiczne

- <wyrażenie logiczne>::= <składnik logiczny> | not <składnik logiczny> | <wyrażenie logiczne> or <składnik logiczny> | <wyrażenie relacyjne>
- <składnik logiczny> ::= <czynnik logiczny> | <składnik logiczny> and <czynnik logiczny>
- <czynnik logiczny> ::= <stała logiczna> | <zmienna logiczna> | (<wyrażenie logiczne>)
- <wyrażenie relacyjne> ::=

<wyrazenie arytmetyczne> <relacja> <wyrażenie arytmetyczne> | <wyrażenie logiczne> <relacja> <wyrażenie logiczne>

- <relacja> ::= = | <> | < | <= | > | >=
- <stała logiczna> ::= false | true

Przykłady wyrażeń logicznych

- true
- x>0
- (x>0) and (not koniec)
- (x>0) or (y>0)
- (x>0) = (y>0)

Przykłady niepoprawnych wyrażeń logicznych:

- x>0 and y>0 {and wiąże silniej, więc 0 and y jest bez sensu}
- x+y = true
- OK and x>0
- (x>0) and not koniec

Semantyka wyrażeń logicznych

- Analogiczna do semantyki wyrażeń arytmetycznych. Przy wyrażeniach relacyjnych wylicza się wartości obu stron dla danego wartościowania i sprawdza się, czy zachodzi odpowiednia relacja.
- Spójniki not, and, or oznaczają standardowe operatory logiczne negacji, koniunkcji i alternatywy

Semantyka instrukcji warunkowej

- Napisy zgodne z semantyką to:

if B then P lub

if B then P1 else P2

$$V[\text{ if B then P }] = \begin{cases} V[P] & \text{gdy Iv(B)=true} \\ V & \text{gdy Iv(B)=false} \end{cases}$$

$$V[\text{ if B then P1 else P2 }] = \begin{cases} V[P1] & \text{gdy Iv(B)=true} \\ V[P2] & \text{gdy Iv(B)=false} \end{cases}$$

Przykład

```
delta := b*b - 4*a*c;

if delta<0 then Write('Nie ma rozwiązań')
else if delta=0 then Write('Jest jedno rozwiązanie')
     else Write('Są dwa rozwiązania')
-----------------------------------------------
if LiczbaStudentów > 100 then Alarm := true
                         else ALarm := false
```

... co jest równoważne

```
Alarm := LiczbaStudentów > 100
```

Instrukcja warunkowa – przykład

```
{a=1,b=-5,c=6,x1=?,x2=?, delta=?}
delta := b*b - 4*a*c;
{a=1,b=-5,c=6,x1=?,x2=?, delta=1}
if delta<0 then Write('Nie ma rozwiązań')
else if delta=0 then
     begin
       x1:= -b/(2*a);
       x2:= x1
     end
     else
     begin
       x1:= (-b-sqrt(delta))/(2*a);
       x2:= (-b+sqrt(delta))/(2*a);
     end
{a=1, b=-5, c=6, x1=2, x2=3, delta=1}
```

Uwaga

- Z gramatyki dla instrukcji warunkowej wynika, że przed else nie może być średnika!

Instrukcja pętli

<instrukcja pętli> := while <warunek > do <instrukcja>

Instrukcję pętli rozumiemy w taki sposób: dopóki warunek jest prawdziwy wykonujemy instrukcję, która występuje po warunku. W pierwszym momencie, w którym po kolejnym (być może zerowym) obrocie pętli warunek przestanie być spełniony, wykonanie pętli kończy się („wychodzimy z pętli”)

Semantyka instrukcji pętli

$$V\,[\text{ while B do P }] = \begin{cases} V & \text{gdy } V(B)=\text{false} \\ (V[P])[\text{while B do P}] & \text{gdy } V(B)=\text{true} \end{cases}$$

Dodatkowo przyjmujemy, że gdy po każdej iteracji pętli V(B)=true, to znaczeniem tej instrukcji jest zapętlenie.

Niestety nie ma możliwości dla każdej pętli zawczasu z góry stwierdzić, czy taka sytuacja wystąpi. Zostało to udowodnione przez Alana Turinga i znane jest jako problem stopu.

Przykład (1)

```
{n=11, x=?, l=?}
x:=1; l:=0;
{n=11, x=1, l=0}
while x<n do
  begin
    x:=x*2;
    l:=l+1;
  end
{x=16, l=4, n=11}
```

Przykład 2

```
{n>0, x=?, l=?}
x:=1; l:=0;
{n>0, x=1, l=0}
while x<n do
  begin
    x:=x*2;
    l:=l+1;
  end

{n>0, x=2^l, l= ⌈log n⌉ }
```

Przykład 3

```
{x0>0}
x:=x0; y:=y0; z:=0;
while x<>0 do
  begin
    if x mod 2 = 1 then z:=z+y;
    x:=x div 2;
    y:=y+y
  end
{x0>0, z=x0*y0}
```

Przykład 3a

```
{x0>0}
x:=x0; y:=y0; z:=1;
while x<>0 do
  begin
    if x mod 2 = 1 then z:=z*y;
    x:=x div 2;
    y:=y*y
  end
{x0>0, z=y0^x0}
```

Przykład 4

```
{x>0}
while x>1 do
   if x mod 2 = 0 then x:= x div 2
                  else x:= 3*x+1
```

Problem Collatza: czy dla każdej wartości początkowej x ten program się zatrzyma?

Kolejne wartościowania dla początkowego x=54

54 27 82 41 124 62 31 94 47 142 71 214 107 322 161
484 242 121 364 182 91 274 137 412 206 103 310
155 466 233 700 350 175 526 263 790 395 1186
593 1780 890 445 1336 668 334 167 502 251 754
377 1132 566 283 850 425 1276 638 319 958 479
1438 719 2158 1079 3238 1619 4858 2429 7288
3644 1822 911 2734 1367 4102 2051 6154 3077
9232 4616 2308 1154 577 1732 866 433 1300 650
325 976 488 244 122 61 184 92 46 23 70 35 106
53 160 80 40 20 10 5 16 8 4 2 1

Instrukcja czytania

- <instrukcja czytania> ::= Read(<zmienna>)

Semantyka instrukcji czytania musi odnieść się do jakiegoś urządzenia wejściowego. Przedstawimy je w postaci listy wartości, które czekają na to, aby być wczytane. Znaczenie instrukcji czytania polegać będzie na pobraniu pierwszej wartości z wejścia i nadaniu tej wartości zmiennej. Instrukcja czytania zmienia więc wartościowanie zmiennych.

Instrukcja pisania

- Instrukcja pisania ::= Write(<wyrażenie>)

Analogicznie, jak dla instrukcji czytania, instrukcja pisania odnosi się do listy wyrażeń wypisanych na wyjście (ekran, plik, drukarkę,...). Po wykonaniu Write(E) lista wyjściowa zostaje wydłużona o wartość Iv(E), a wartościowanie zmiennych nie ulega zmianie.

Formalnie...

- Instrukcja czytania i pisania komplikuje nam nieco formalny opis działania programu, bo odnosi się do świata zewnętrznego. Nie tylko wartościowanie opisuje stan programu, ale i zawartości dwóch list: wejściowej i wyjściowej.
- My skupimy się na wartościowaniach pamiętając o istnieniu tych dwóch list.

Przykłady

```
{x=?, lista wejściowa= 2,3,4}
  Read(x)
{x=2, lista wejściowa = 3,4}
--------------------------------
{x=2, lista wyjściowa = 2,7,6}
Write(x+3);
{x=2, lista wyjściowa = 2,7,6,5}
```

Skróty

- Aby ułatwić pisanie i czytanie, wprowadzamy skróty notacyjne pozwalające wczytywać i wypisywać na raz parę obiektów:

Read(x1,...,xn)

Write(E1,...,En)

- Typy obiektów, które można wczytywać i wypisywać:
  - liczby całkowite
  - liczby rzeczywiste
  - znaki
  - napisy

Przykłady

```
{x=3,y=4,lista wyjściowa=ε}
Write('Wynikiem obliczeń jest',x+y)
{x=3,y=4, lista wyjściowa =
                'Wynikiem obliczeń jest 7'}
-----------------------------------
{x1=3,y1=4,x2=2,y2=4)
Write('Robot przeszedł z pola (', x1,y1,
  ') na pole (',x2,y2,')');
{x1=3,y1=4,x2=2,y2=4,
 lista wyjściowa=
'Robot przeszedl z pola (3,4) na pole (2,4)'}
```

Formatowanie wydruku

- Write(n:5) spowoduje wypisanie liczby całkowitej n na dokładnie 5 pozycjach.
- Write(x:10:3) dla liczby rzeczywistej spowoduje wypisanie wartości x na łącznie 10 pozycjach (z kropka dziesiętną), z czego 3 cyfry będą po kropce.