Bab III
Semantik

Konsep Semantik Bahasa Pemrograman (Semantik Analisis)

- Dari pembahasan bab-bab terdahulu maka kita ketahui bahwa proses ini merupakan proses kelanjutan dari proses kompilasi sebelumnya, yaitu analisa leksikal (scanning) dan analisa sintaks (parsing)
- Bagian terakhir dari tahapan analisis adalah analisis semantik
- Memanfaatkan pohon sintaks yang dihasilkan dari parsing
- Proses analisa sintaks dan analisa semantik merupakan 2 proses yang sangat erat kaitannya dan sulit untuk dipisahkan
- Contoh : A = (A + B) * (C + D)
- Parser hanya akan mengenali simbol-simbol ':=', '+', '*', parser tidak mengetahui makna dari simbol-simbol tersebut
- Untuk mengenali makna dari simbol-simbol tersebut, maka compiler memanggil routine semantics

Untuk mengetahui makna, maka routine ini akan memeriksa :

a. apakah variabel yang ada telah didefinisikan sebelumnya
b. apakah variabel-variabel tersebut tipenya sama
c. apakah operand yang akan dioperasikan tersebut ada nilainya, dst
d. menggunakan tabel simbol
e. pemeriksaan bisa dilakukan pada tabel identifier, tabel display, dan tabel block

Pengecekan yang dilakukan dapat berupa :

1. Memeriksa penggunaan nama-nama (keberlakuannya)

    a. Duplikasi
       Apakah sebuah nama terjadi pendefinisian lebih dari 2 kali. Pengecekan dilakukan pada bagian pengelolaan block
    b. Terdefinisi
       Apakah nama yang dipakai pada program sudah terdefinisi atau belum. Pengecekan dilakukan pada semua tempat kecuali block

2. Memeriksa Tipe
   Melakukan pemeriksaan terhadap kesesuaian tipe dalam statement yang ada, misalnya bila terdapat suatu operasi, diperiksa tipe operandnya.

   Contoh :

    - Ekspresi yang mengikuti If berarti tipenya Boolean, akan diperiksa tipe identifier dan tipe ekspresinya
    - Bila ada operasi antara 2 operand maka tipe operand yang pertama harus dioperasikan dengan operand yang kedua

Analisa semantik sering juga digolongkan dengan Intermediate Code yang akan menghasilkan output Intermediate Code.

Syntax-Directed Translation

Kode antara (Intermadiate Code) adalah sebuah representasi yang disiapkan untuk mesin abstrak tertentu. Dua sifat yang harus dipenuhi oleh kode antara adalah :

1. dapat dihasilkan dengan mudah
2. mudah ditranslasikan menjadi program sasaran (target program)

Representasi kode antara biasanya terbentuk tiga alamat (three-address code), baik berbentuk quadruples ataupun triples.

Kode antara (intermediate code) dibentuk dari sebuah kalimat x dalam bahasa context free. Kalimat x ini adalah keluaran dari parser. Kalimat ini tentu saja dapat dinyatakan dalam representasi pohon parsing (parse tree).

Syntax-directed translation adalah suatu urutan proses yang mentranslasikan parse tree menjadi kode antara. Tahap pertama dari pembentukan kode antara adalah evaluasi atribut setiap token dalam kalimat x. Yang dapat menjadi atribut setiap token adalah semua informasi yang dapat disimpan di dalam tabel simbol. Evaluasi dimulai dari parse tree.

Pandang sebuah node n yang ditandai sebuah token x pada parse tree. Kita tuliskan x.a untuk menyatakan atribut a untuk token x pada node n tersebut. Nilai x.a pada node n tersebut dievaluasi dengan menggunakan aturan semantik (semantic rule) untuk atribut a. Aturan semantik ini ditetapkan untuk setiap produksi dimana x adalah ruas kiri produksi. Sebuah parse tree yang menyertakan nilai-nilai atribut pada setiap nodenya dinamakan Annonated parse tree. Kumpulan aturan yang menetapkan aturan-aturan semantik untuk setiap produksinya dinamakan syntax-directed definition.

Untuk jelasnya berikut ini adalah sebuah syntax-directed translation yang mentransasikan ekspresi infix menjadi ekspresi postfix. Ekspresi infix ini dapat dipandang sebagai sebuah kalimat yang dihasilkan oleh parser.

Contoh :

Diketahui :

1. Kalimat x : 9 – 5 + 2
2. Grammar Q = { E → E + T | E – T | T, T → 0|1|2|…|9}
3. Syntax-directed definition :

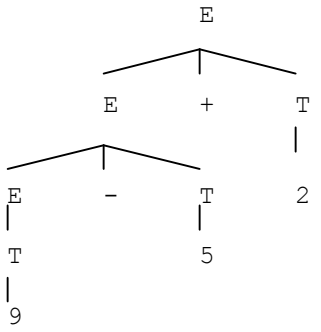
Produksi	Aturan Semantik
E → E + T	E.t := E.t || T.t || '+'
E → E - T	E.t := E.t || T.t || '-'
E → T	E.t := T.t
T → 0	T.t := '0'
T → 1	T.t := '1'
…	…
T → 9	T.t := '9'

Catatan :

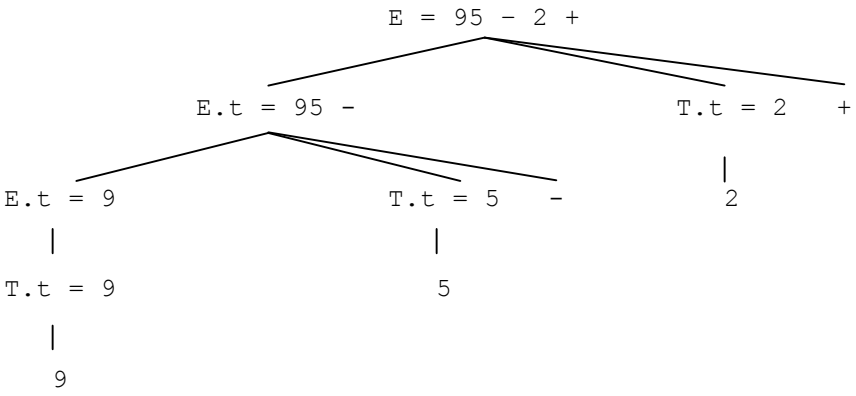
- lambang '||' menyatakan concatenation
- aturan semantik kedus produksi pertama adalah concate dua operand diikuti sebuah operator

Langkah-langkah translasi

1. Pembentukan Parse Tree



2. Pembentukan Annonated Parse Tree



Teknik-teknik Pendeskripsian Semantik Bahasa Pemrograman

a. Operational Semantic

Pendekatan ini mendefinisikan suatu mesin buatan (Abstract) dengan instruksi-instruksi primitif, tidak perlu realistik, tetapi cukup sederhana supaya tidak muncul kesalahpahaman. Deskripsi semantik dari bahasa pemrograman menentukan suatu translasi ke kode.

Operational semantics for Peano arithmetic

Abstarct Syntax :

N in Nat (the natural numbers)
N ::=0 | S(N) | (N + N) | (N X N)

Interpreter :

I : N **->** N
I[(n + 0)] ==> n
I[(n x 0)] == > 0
…
where m,n in nat

Operational Examples

- Stack
- List
- Queue
- Binary Search tree
- Graph
- Complex numbers
- Rational numbers
- Floating point numbers

b. Denotational Semantic

Pada pendekatan ini, diberikan suatu fungsi yang memetakan program-program komputer yang ditunjuk ke dalam bentuk nilai-nilai abstrak secara matematika (angka, nilai, kebenaran, fungsi matematika, dsb).

Denotational definitioan of Peano arithmetic

Abstract Syntax :

N in Nat (the natural numbers)

N ::=0 | S(N) | (N + N) | (N X N)

Semantic Algebra :

Nat (the Natural Numbers (0, 1, …))

+ : Nat -> Nat -> Nat

Valuation Function :

D : Nat -> **Nat**

D [(n + 0)] = **D**[n]

D [(n x 0)] = **0**

…

where m,n in Nat

Denotational

- Show that the following code denotes the same following

```
int f (int n) {
      If n > 1 then n*f(n-1)
      else 1
}
int f (int n) {
      int t = 1;
      while n > 1 do {
            t := t*n
            n := n-1
      }
}
```

Denotational Examples

- Stack
- List
- Queue
- Binary Search tree
- Graph
- Complex numbers
- Rational numbers
- Floating point numbers

c. Axiomatic Semantic

Pada pendekatan ini didefinisikan suatu tindakan program yang dibangun dengan properti logika yang menyimpan status komputer sebelum dan sesudah eksekusi.

```
s,I := 0,0
while I < n do    S,I := S+A[I+1],I+1
end
-------------------------
S=0
i=0
DO WHILE i<=n
     i=i+1
     S=S+A[i]
LOOP
-------------------------
S=0
i=0
DO WHILE i<n
     S=S+A[i]
     i=i+1
LOOP
```

Axiomatic Examples

- Linear Search
- Integer divison implemented by repeated subtraction
- Factorial function
- Fn the n-th Fibonacci number where
  F0 = 0, F1 = 1, and Fi+2 = Fi+1 + Fi for i >= 0.
- Binary Search
- Quick Sort

d. Algebraic Semantic

Pada pendekatan ini dipertimbangkan suatu objek komputasi yang menjadi syarat-syarat dalam aljabar multi-sorted. Program mengimplementasikan fungsi yang dapat diwujudkan dengan suatu persamaan diantara syarat-syarat tersebut.

Domains:

```
     Bool = {true, false} (Boolean values)
     N in Nat (the natural numbers)
     N ::= 0 | S(N)
```

Functions:

```
     = : (Nat, Nat) -> Bool
     + : (Nat, Nat) -> Nat
     × : (Nat, Nat) -> Nat
```

Axioms and equations:

```
     not S(N) = 0
     if S(M) = S(N) then M = N
     ( n + 0 ) = n
     ( n × 0 ) = 0
```

Algebraic Examples

- Stack
- List
- Queue
- Binary Search tree
- Graph
- Complex numbers
- Rational numbers
- Floating point numbers

e. Structured Operational atau Natural Semantic

Seperti dalam pengambilan keputusan secara alamiah dengan logika. Program diberi suatu arti dari aturan yang diturunkan yang menggambarkan penilaian gagasan suatu bahasa.