SEMANTIK

Sintak mendifinisikan suatu bentuk program yang benar dari suatu bahasa.

Semantic mendefinisikan arti dari program yang benar secara sintak dari bahasa tersebut.

Semantic suatu bahasa membutuhkan semacam ekspresi untuk mengirimkan suatu nilai kebenaran (TRUE, FALSE, NOT atau nilai integer).

Dalam banyak kasus, program hanya dapat dieksekusi jika benar, serta mengikuti aturan sintak dan semantic.

Semantic suatu bahasa pemrograman mempunyai banyak potensial / keunggulan, beberapa diantara nya adalah :

a. Standarisasi bahasa pemrograman.
   Banyak usaha yang dilakukan untuk menstandarisasi bahasa pemrograman seperti FORTRAN, COBOL, dan PL/1, untuk lebih memudahkan programmer menggunakannya.

b. Referensi untuk user.
   Programmer membutuhkan suatu dokumentasi yang pasti supaya user dapat mengoperasikan program yang dibuat dengan baik.

c. Pembuktian dari program yang benar.
   Secara matematis, program tidak mungkin bekerja dan berjalan jika tidak ada semantic.

d. Referensi untuk implementor.
   Semantic akan mencegah suatu gaya bahasa yang tidak kompetibel yang diwujudkan dalam suatu implementasi berbeda walaupun dengan bahasa yang sama.

e. Implementasi otomatis.
   Suatu tool/alat dapat secara otomatis membuatu translasi bahasa yang melebihi parsing. Hal ini dapat dilakukan jika semantic sudah dirumuskan.

f. Pemahaman yang lebih baik dari desain bahasa.
   Jika suatu rumusan semantic sulit untuk di deskripsikan secara formal maka rumusan semantic tersebut juga akan sulit digunakan oleh programmer.

Dua alasan mengapa perlu memahami suatu desain bahasa pemrograman dengan lebih baik, yaitu :

1. Memahami dengan lebih baik suatu desain bahasa pemrograman, berarti mambantu menguasai dan menggunakan bahasa tersebut.

2. Memahami dengan lebih baik suatu disain bahasa pemrograman secara detail, membantu programmer untuk memperbaiki proses pengembangan program menjadi lebih baik.

Teknik semantic :

a. Operational semantic
   Pendekatan ini mendefinisikan suatu mesin buatan (abstrak) dengan instruksi-instruksi primitive, tidak perlu realistic, tetapi cukup sederhana supaya tidak muncul kesalahpahaman. Deskripsi semantic dari bahasa pemrograman menentukan suatu translasi ke kode.

b. Detonational semantic.
   Pada pendekatan ini, diberikan suatu fungsi yang memetakan program-program computer yang di tunjuk ke dalam bentuk nilai-nilai abstrak secara matematika (angka, nikai kebenaran, fungsi matematika, dsb).

c. Axiomatic semantic.
   Pada pendekatan ini di definisikan suatu tindakan program yang di bangun dengan property logika yang menyimpan status computer sebelum dan sesudah dieksekusi.

d. Algebraic semantic.
   Pada pendekatan ini dipertimbangkan suatu objek komputasi yang menjadi syarat-syarat dalam aljabar multi stored. Program mengimplementasikan fungsi yang dapa di wujudkan dengan suatu persamaan di antara syarat-syarat tersebut.

e. Structured operational semantic atau natural semantic.
   Seperti dalam pengambilan keputusan secara alamiah dengan logika, program di beri suatu arti dari aturan yang diturunkan yang menggambarkan penilaian gagasan suatu bahasa.

Proses analisa sintak dan analisa semantic merupakan 2 proses yang sangat erat kaitannya dan sulit untuk dipisahkan.

Contoh :

A:= (A+B) * (C+D)

Parser hanya akan mengenali symbol-simbol ‘:=”, “+”, “*”, parser tidak mengetahui makna dari symbol-simbol tersebut. Untuk mengenali makna dari symbol-simbol terserbut maka compiler memanggil routin semantics.

Untuk mengetahui makna, maka routin ini akan memeriksa :

- Apakah variable yang ada telah didefinisikan sebelumnya.
- Apakah variable-variabel terserbut tipenya sama.
- Apakah operand yang akan di operasikan terserbut ada nilainya, dst.

- Menggunakan table symbol.
- Pemeriksaan bisa dilakukan pada table identifier, table display, dan table block.

Analisa semantic sering juga digunakan dengan intermeadiate code yang akan menghasilkan output intermediate code.

Sintax-Directed Translator

Kode Antara (Intermediate Code) adalah : sebuah representasi yang disiapkan untuk mesin abstrak tertentu.

Dua sifat yang harus dipenuhi oleh kode antara adalah :

1. dapat di hasilkan dengan mudah.
2. Mudah ditranslasikan menjadi program sasaran (target program).

Representasi kode antara biasanya berbentuk 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 iniadalah 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 disimpan didalam table symbol. Evaluasi dinilai 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 di evaluasi dengan menggunakan aturan semantic (semantic rule) untuk atribut a. Aturan semantic ini di tetapkan untuk setiap produksi dimana x adalah ruas kiri produksi sebuah parse tree yang menyertakan nilai-nilai atribut pada setiap node nya, dinamakan annonated parse tree. Kumpulan aturan yang menetapkan aturan-aturan semantic untuk setiap tahap produksinya dinamakan syntax directed definition.

Untuk jelasnya berikut ini adalah sebuah syntax directed translation yang menstranslasikan ekspresi infix menjadi ekspresi posfix. 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 | I, T → 0|1|2|…|9}
3. syntax directed definition.

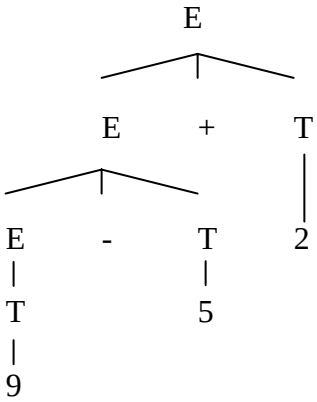
Produksi	Aturan Semantik
E → E1 + T	E := E1.+ || T.+ || '+'
E → E1 – T	E := E1.+ || T.+ ||'-'
E → T	E := T.+
E → 0	E := '0'
E → 1	E := '1'
…	…
E → 9	E := '9'

Catatan :

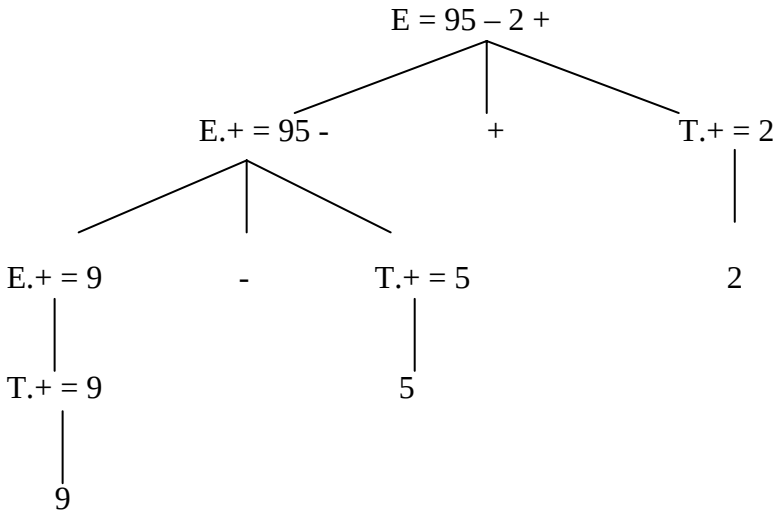
- Lambang ' || ' menyatakan concatenation
- Aturan semantic kedua produksi pertama adalah concate dua operand diikuti sebuah operator.

Langkah-langkah translasi :

1. Pembentukan parse tree



2. Pembentukan annonates parse tree



KODE ANTARA

Kode antara / Intermediate code

Merupakan : hasil dari tahapan analisis yang dibuat oleh kompilator pada saat mentranslasikan program dari bahasa tingkat tinggi.

Kegunaannya :

- untuk memperkecil usaha dalam membangun kompilator dari sejumlah bahasa ke sejumlah mesin. Dengan adanya kode antara yang lebih machine independent maka kode antara yang dihasilkan dapat digunakan lagi pada mesin lainnya.
- Proses optimasi masih lebih mudah. Beberapa strategi optimasi lebih mudah dilakukan pada kode antara dari pada program sumber atau pada kode assembly dan kode mesin.
- Bisa melihat program internal yang gampang dimengerti. Kode antara ini akan lebih mudah dipahami dari pada kode assembly atau kode mesin.

Kerugiannya :

- Bila melakukan dua kali translasi, membutuhkan waktu yang relative lama.

Dua macam kode antara :

1. Notasi posfix

Pada suatu operasi : Notasi infix → letak operator berada di tengah
Notasi postfix → operator diletakkan paling akhir. Disebut juga notasi suffix atau reverse polish.

Sintax notasi postfix : **<Operand> <Operand> < Operator>**

Contoh :

1. (a+b) * (c+d) dinyatakan dalam postfix → ab+cd+*
2. If <exp> THEN <stmt1> ELSE <stmt2>
   Dinyatakan dalam postfix → <exp> <label> BZ <stmt1> <label> BR <stmt2>

Label1 label2

Arti dari notasi postfix nya :

"jika kondisi ekspresi slah maka instruksi akan meloncat ke label1 dan menjalankan statemet2. Bila kondisi benar, maka statement1 akan dijalankan lalu meloncat ke label2. label1 dan label2 sendiri menunjukkan posisi tujuan loncatan, untuk label1 posisinya tepat sebelum statemen2, dan label2 setelah statemen2"

BZ → branch if zero (zero = salah) (bercabang/meloncat jika kondisi yang di tes salah)
BR → branch (bercabang/meloncat tanpa ada kondisi yang di tes)

3. if a > b THEN
   c := d
   ELSE
   c := e

   Diubah ke postfix :

```
11.   a
12.   b
13.   >
14    22          {menunjuk label1}
15    BZ
16.   c
17.   d
18.   :=
19.
20.   25          {menunjuk label2}
21.   BR
22.   c
23.   e
24.   :=
25.
```

   artinya :

   - bila ekspresi (a>b) salah, maka loncat ke instruksi no. 22
   - bila ekspresi (a>b) benar, tidak terjadi loncatan, instruksi berlanjut ke no 16 sampai 18, lalu loncat ke no 25.

2. Notasi N-TUPLE

- Triples Notation
- Quadruples Notation

Bila pada postfix setiap baris instruksi hanya terdiri dari suatu tupel, maka pada notasi N-Tuple setiap baris dapat terdiri dari beberapa tuple.

Bentuk Umum : Operator ……. N-1 operand

Triples Notation (3 Tuple)

Bentuk Umum : **<operator> <operand> <operand>**

Contoh :

1. A := D * C + B / E

   Kode antara tripel :
   1. *, D, C
   2. /, B, E
   3. +, (1), (2)
   4. :=, A, (3)

2. IF x > y THEN
   x := a – b
   ELSE
   x := a + b

   Kode antara tripel :
   1. >, x, y
   2. BZ, (1), (6) {bila kondisi (1) salah, loncat ke no (6) }
   3. -, a, b
   4. :=, x, (3)
   5. BR, , (8)
   6. +, a, b
   7. :=, x, (6)

Kekurangan Notasi Tripel : sulit pada saat melakukan optimasi, maka dikembangkan indirect triples yang memliliki 2 list (senerai), yaitu :

- list instruksi : berisi notasi tripel
- list eksekusi : mengatur urutan eksekusinya

Quadruples Notation (4 tuple)

Bentuk Umum : **<operator> <operand> <operand> <hasil>**

Hasil adalah temporary variable yang dapat di tempatkan pada memory atau register. Masalah yang ada bagaimana mengelola temporary variable (hasil) seminimal mungkin.

Contoh :

A := D * C + B / E

Kode antaranya adalah :

1. *, D, C, T1
2. /, B, E, T2
3. +, T1, T2, A