Semantics Analyser

- 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 sintak dan analisa semantik merupakan dua proses yang sangat erat kaitannya, dan sulit untuk dipisahkan

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

Semantics Analyser

Pengecekan yang dilakukan dapat berupa:

1. **Memeriksa penggunaan nama-nama (keberlakuannya)**

- **Duplikasi**

  Apakah sebuah nama terjadi pendefinisian lebih dari dua kali. Pengecekan dilakukan pada bagian pengelolaan block

- **Terdefinisi**

  Melakukan pengecekan apakah sebuah nama yang dipakai pada tubuh program sudah terdefinisi atau belum. Pengecekan dilakukan pada semua tempat kecuali blo

2. Memeriksa tipe

Melakukan pemeriksaan terhadap kesesuaian tipe dalam statement - statement yang ada. Misalkan bila terdapat suatu operasi, diperiksa tipe operand. Contohnya bila ekspresi yang mengikuti instruksi IF berarti tipenya boolean, akan diperiksa tipe identifier dan tipe ekspresi. Bila ada operasi antara dua operand, maka tipe operand pertama harus bisa dioperasikan dengan operand kedua

ANALISIS SEMANTIK

Analisis semantic dapat dibagi menjadi dua kategori :

1. Adalah analisis program yang dibutuhkan oleh aturan bahasa pemrograman untuk memberikan kebenaran dan menjamin pelaksanaan program penerjemah.

2. Kategori analisis adalah analisis yang dilakukan oleh complier untuk meningkatkan efisiensi pelaksanaan program pemerintah

ATRIBUT DAN ATRIBUT GRAMMAR

Atribut merupakan salah satu pembentuk konstruksi Bahasa pemrograman. Atribut mengandung berbagai informasi, kompeksitas dan waktu proses eksekusi.

Contoh atribut :

1. Tipe data dan variable
2. Nilai dari suatu ekspresi
3. Lokasi variable pada memory
4. Kode objek dari sebuah prosedur
5. Jumlah digit yang signifikan dari sebuah angka

Proses komputasi atribut dan asosiasinya dihitung dengan nilai konstruksi Bahasa yang disebut pengikat/ *binding* .

Atribut grammar

Jika X adalah sebuah grammar dan a adalah atribut yang dikaitkan dengan X maka ditulis X.a untuk nilai binding dengan X

Atribut tata Bahasa yang ditulis dalam bentuk tabel, dengan setiap aturan tata Bahasa yang terdaftar dengan persamaan atribut, atau aturan semantik yang berhubungan dengan aturannya, sebagai berikut :

Contoh :

Tata bahas berikut untuk unsigned number

Number → number digit | digit

Digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

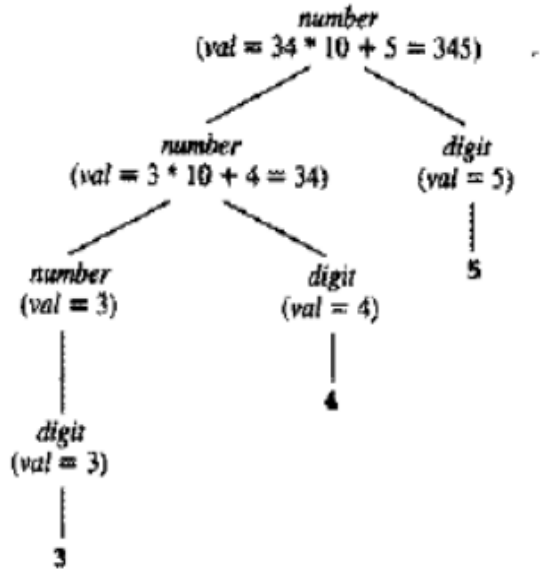
Pembentukan grammar rules :

number → digit	***number.val = digit.val***
number→ number digit	***number1 → number2 digit***

Sehingga tabel analisis semantik :

Grammar Rule	Semantic Rules
$number_1 \rightarrow number_2\ digit$	$number_1.val = number_2.val * 10 + digit.val$
$number \rightarrow digit$	$number.val = digit.val$
$digit \rightarrow 0$	$digit.val = 0$
$digit \rightarrow 1$	$digit.val = 1$
$digit \rightarrow 2$	$digit.val = 2$
$digit \rightarrow 3$	$digit.val = 3$
$digit \rightarrow 4$	$digit.val = 4$
$digit \rightarrow 5$	$digit.val = 5$
$digit \rightarrow 6$	$digit.val = 6$
$digit \rightarrow 7$	$digit.val = 7$
$digit \rightarrow 8$	$digit.val = 8$
$digit \rightarrow 9$	$digit.val = 9$

Pohon parsing untuk string “345” pada analisis semantic :



Algoritma komputasi atribut menggunakan depency graph dan evaluation order.

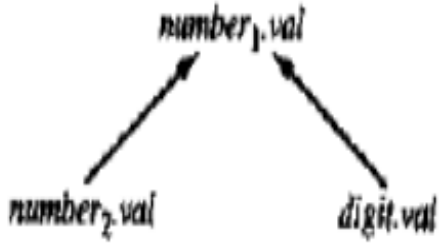
Diberikan sebuah grammar, dimana memiliki aturan dependency graph, setiap graph memiliki node yang dilabelkan sebagai atribut dengan aturan sebagai berikut :

$$X_i.a_j := f_{ij}(\ldots, X_m.a_k, \ldots)$$

Sehingga grammar berikut yang memiliki atribut :

$$Number_1.val = number_2.\ Val * 10 + digit.val$$

Dapat dibuat aturan dependency graph :



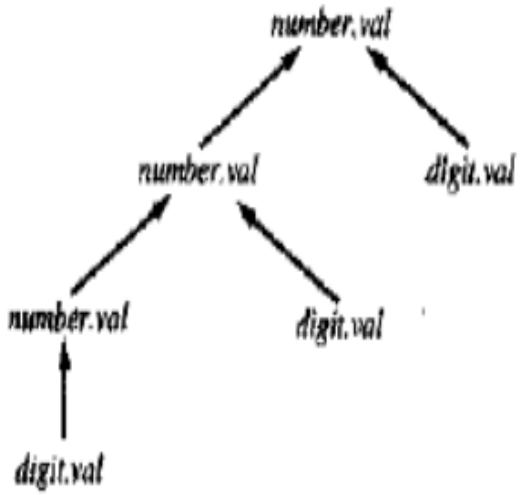
Atau (karena dependency graph menghilangkan subkrip untuk symbol yang berulang karena reperensi graph jelas membedakan kejadian yang berbeda sebagai node yang berbeda)

Number.val

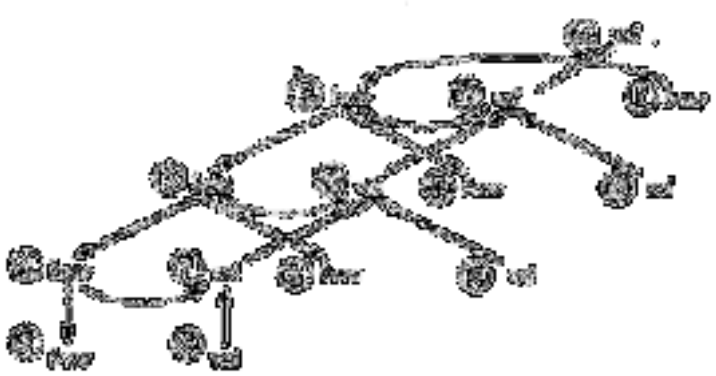
↑

Digit.val

Sehingga string 345, dependency graph nya sebagi berikut :



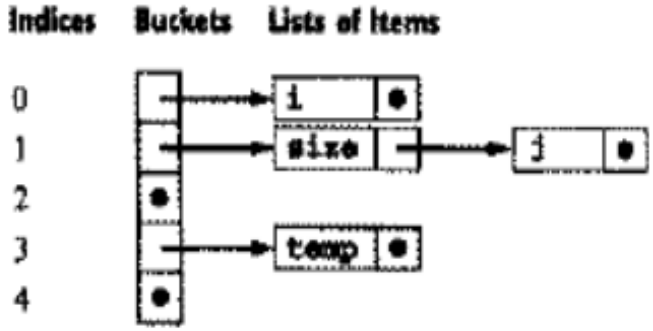
Diketahui bahwa grafik harus acyclic, maka graph juga bersifat directed acyclic graph (DAG). Contoh DAG untuk string 345o :



SYMBOL TABLE / Tabel Simbol

Symbol table dalam kompilator adalah sebuah struktur data tipe kamus data, degna 3 operator yaitu insert, lookup dan delete.

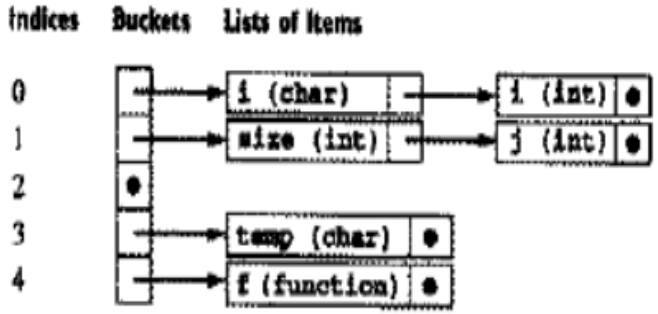
Struktur dari tabel symbol / tabel hash adalah sebagai berikut :



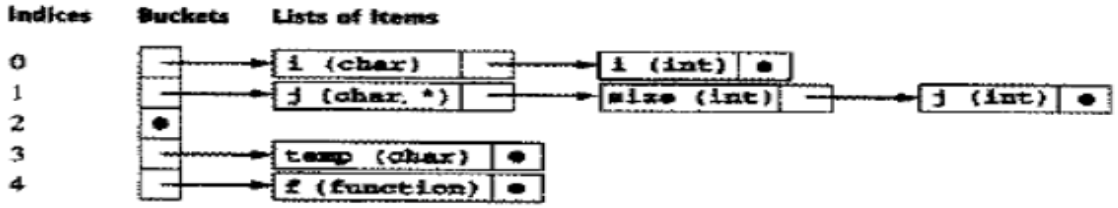
```
program Ex;
var i,j: integer;

function f(size: integer): integer;
var i,temp: char;

  procedure g;
  var j: real;
  begin
    ...
  end;

  procedure h;
  var j: ^char;
  begin
    ...
  end;

begin (* f *)
  ...
end;

begin (* main program *)
  ...
end.
```

Maka tabel symbol untuk procedure f :

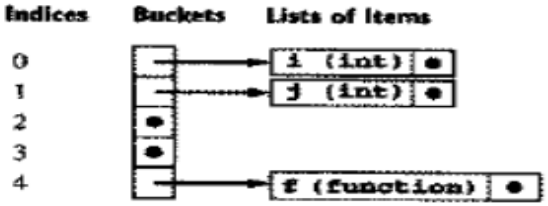
Tabel symbol sebelum deklarasi procedure f



Tabel symbol setelah deklarasi pada procedure f :



Tabel symbol setelah keluar dari deklarasi (dan menghapusnya)



DATA TYPERS DAN TYPE CHECKING (TIPE DATA DAN PEMERIKSAAN TIPE DATA)

DATA TYPES

Salah satu tugas pokok kompilator adalah komputasi dan maintenance informasi tentang jenis data dan penggunaan informasi untuk memastikan bahwa setiap bagian dari sebuah program sesuai dengan aturan Bahasa (type checking)

Tipe data adalah seperangkat nilai atau lebih, dimana satu set nilai memiliki operasi tertentu pada nilai tersebut. Contoh, nilangan integer pada Bahas programming merupakan subset bilangan bulat dan operasi aritmatika seperti + dan *.

Type Checking

Type checking meliputi :

1. Deklarasi menyebabkan jenis identifier yang akan dimasukkan ke dalam tabel symbol. Contoh grammar berikut :

program → *var-decls* **;** *stmts*
var-decls → *var-decls* **;** *var-decl* | *var-decl*
var-decl → **id** **:** *type-exp*
type-exp → **int** | **bool** | **array** **[num]** **of** *type-exp*
stmts → *stmts* **;** *stmt* | *stmt*
stmt → **if** *exp* **then** *stmt* | **id** **:=** *exp*

Apabila yang akan di check adalah :

Var-decl→id : type-exp

Maka Action semantic analisis nya :

Insert (id.name.type-exp.type)

Terlihat bahwa types dari grammar diatas, memiliki struktur pohon yang disebut dengan struktur Array :

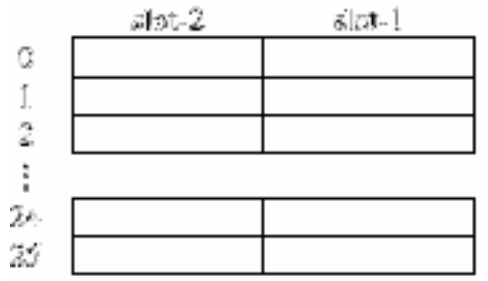
makeTypeNode (array, size, type)

Dapat dibuat pohon untuk struktur array :

1. Statement, tidak memiliki tipe data. Tapi bagian struktur harus diperiksa untuk koreksi tipe data.
2. Ekspresi, seperti angka dan nilai Boolean (true dan false) akan dijelaskan sebagai tipe integer dan Boolean
3. Overloading, periksaan operator, yang mana disebut overload apabila nama operator yang sama digunakan untuk 2 atau lebih operasi yang berbeda.

2. Hash Table

- Struktur data yang layak digunakan untuk mengimplementasikan tabel simbol adalah *linked list* terutama dalam tiga manifetasinya :
  - *linear linked list*,
  - *tree*, atau
  - *hash*.
- Dari ketiga manifestasi tersebut *hash* (atau sering disebut *hash tabel*) adalah yang paling layak untuk dipilih.
- Sebuah *hash table* terdiri dari beberapa *bucket*. Setiap *bucket* terdiri dari beberapa *slot*. Setiap *bucket* diberi nomor, dimulai dari nomor 0 (nol).
- Berikut ini adalah contoh sebuah *hash table* yang terdiri dari *26 bucket* (b = 26) dan *2 slot/bucket* (s = 2) :



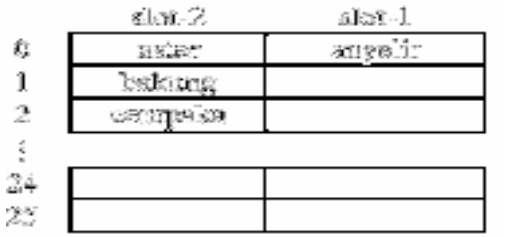
- Dalam pembicaraan tentang *hash table* dikenal istilah *hash function f*. Fungsi f ini digunakan untuk memasukkan nama_pengenal ke dalam tabel simbol.
- Salah satu contoh ungkapan fungsi f adalah :
  f(X) = kode dari karakter pertama dari X dimana X adalah nama_pengenal.

  Misalkan terdapat 5 nama_pengenal :

  - *aster*,
  - *anyelir*,
  - *anggrek*,
  - *bakung*, dan
  - *cempaka*.

  Misalkan pula aturan pengkodean adalah :
  - a = 0,
  - b = 1, dan
  - d = 2.

Tabel simbol yang dihasilkan adalah :



- Perhatikan bahwa nama_pengenal *aster*, *anyelir*, dan *anggrek* mempunyai nilai f(X) yang sama.
- Karena setiap *bucket* hanya terdiri dari 2 slot maka nama_pengenal *anggrek* tidak dapat dimasukkan ke dalam tabel simbol.
- Jika sebuah nama_pengenal dipetakan ke dalam *bucket* yang sudah penuh akan terjadi *overflow*.
- *Collision* terjadi jika dua nama_pengenal berbeda mempunyai nilai fungsi yang sama (artinya dipetakan ke *bucket* yang sama).
- Jika s = 1 maka *collision* dan *overflow* terjadi bersamaan.
- Fungsi hash yang lebih baik dan sering dipakai adalah *uniform hash function*. Dikatakan *uniform* karena dengan fungsi ini semua nama_pengenal mempunyai peluang yang sama untuk dipetakan ke setiap *bucket*.
- Salah satu bentuk *unform hash function* ini adalah : f(X) = kode(X) mod M

  dengan M adalah bilangan prima yang lebih besar dari 20.
- Dengan menggunakan fungsi f ini maka ukuran *bucket* adalah b = M dengan s tertentu.
- Bagaimanapun dengan s tertentu dapat juga terjadi *overflow*.
- Untuk mengatasi hal ini *chaining hash table* adalah solusi yang tepat.
- Dengan *chaining* ini maka ukuran *slot* setiap *bucket* dapat bervariasi.

Berikut ini adalah bentuk *chaining hash table* :