PERTEMUAN 14

ANALISIS SEMANTIK DAN PEMBENTUKAN KODE (2)

A. TUJUAN PEMBELAJARAN

Pada pertemuan ini akan dijelaskan mengenai pengertian Analisis Semantik, Ide Pokok Analisis Semantik, Struktur Konkret dan Sintaks Abtrak serta Aturan Validasi dan Visibilitas. Setelah menyelesaikan materi pada pertemuan ini, mahasiswa mampu:

1. Menjelaskan pengertian Atribut Grammars.
2. Menjelaskan pengertian Penilaian Atribut Berdasarkan Permintaan.
3. Menjelaskan pengertian Penilaian Atribut Statis.
4. Menjelaskan perbedaan Penilaian Atribut berdasarkan permintaan dan Penilaian Atribut Statis.

B. URAIAN MATERI

1. Atribute Grammars

Masing-masing pada dasarnya ternyata merupakan perhitungan pada pohon sintaksis. Itu akan senang rasanya memiliki mekanisme deskripsi juga untuk tugas-tugas ini, yang darinya implementasi dapat dibuat secara otomatis.

Mekanisme deskripsi yang elegan dan kuat yang dapat memenuhi tujuan ini adalah ditawarkan oleh tata bahasa atribut. Tata bahasa atribut memperluas tata bahasa bebas konteks dengan mengaitkan atribut dengan simbol tata bahasa bebas konteks yang mendasarinya. Atribut ini adalah wadah untuk informasi semantik statis.

Nilai mereka adalah dihitung dengan komputasi yang dilakukan pada pohon, dengan komputasi yang melintasi pohon sesuai kebutuhan. Himpunan atribut simbol *X* dilambangkan dengan *A(X).*

Dengan setiap atribut a dikaitkan dengan tipe τ_a yang menentuan himpunan nilai yang mungkin untuk instance atribut. Pertimbangan produksi *p : Xo → X1 ... Xk* dengan $k \geq 0$ muncul disisi kanan. Untuk membedakan kemunculan sibol yang berbeda dalam *p* produksi, diberikan penomoran dari

kiri ke kanan. Sisi kiri nonterminal *Xo* akan dilambangkan dengan *p[0]*, simbol ke-i disisi kanan dengan *p[i]* untuk *i = 1,...,k*. Atribut a pada simbol *X* memiliki kemunculan atribut pada setiap kemunculan X dalam sebuah produksi. Terjadinya atribut a pada simbol *Xi* dilambangkan dengan *p[i].a.*

Untuk setiap produksi, disediakan spesifikasi fungsional bagaimana atributnya simbol yang terjadi dapat ditentukan dari nilai atribut selanjutnya dari kemunculan simbol dari produksi yang sama. Spesifikasi ini disebut semantic aturan. Dalam contoh kami, aturan semantik direalisasikan dengan cara seperti OCAML bahasa pemrograman. Ini memiliki keuntungan ekstra dari spesifikasi eksplisit jenis dapat dihilangkan.

Contoh terbatas dari mekanisme seperti itu sudah disediakan oleh parser LR standar seperti YACC atau BISON: Di sini, setiap simbol tata bahasa dilengkapi dengan satu atribut. Untuk setiap produksi maka ada satu aturan semantik itu menentukan bagaimana atribut kejadian nonterminal yang terjadi di sisi kiri ditentukan dari atribut kemunculan simbol di sisi kanan.

Contoh 14.1 Pertimbangkan CFG dengan nonterminals E, T, F untuk ekspresi aritmatika. Set terminal terdiri dari simbol tanda kurung, operator, dan simbol var dan const, yang masing-masing mewakili variabel int dan konstanta. Itu non terminal harus dilengkapi dengan pohon atribut yang menerima internal representasi ekspresi.

Untuk menentukan nilai atribut, kami memperluas produksi tata bahasa dengan aturan semantik sebagai berikut:

P1 : E → E+ T

$P_1[0].tree = plus(p_1[1].tree, p_1[2].tree)$

P2 : E → T

$P_2[0].tree = p_2[1].tree$

*P3 : T → T * F*

$P_3[0].tree = mult\ (p_3[1].tree,\ p_3[2].tree)$

P4 : T → F

$P_4[0].tree = p_4[1].tree$

P5 : F → const

P_5[0].tree = int (p_5[1].val)

P6 : F → var

P_6[0].tree = var (p_6[1].id)

P7 : F → (E)

P_6[0].tree = p_6[2].tree

Untuk pembentukan representasi internal, konstruktor Plus, Mult, Int Var

Telah diterapkan. Selanjutnya, diasumsikan bahwa simbol const memiliki atribut *val* berisi nilai konstanta, dan simbol var memiliki atribut *id* berisi pengenal unik untuk variabel. Beberapa generator parser menangani berbagai kemunculan simbol dalam sebuah produksi dengan mengindeks kejadian menurut setiap simbol secara terpisah.

Contoh 14.2 Perhatikan kembali tata bahasa dari Contoh 14.1. Berdasarkan konvensi pengindeksan kejadian berbeda dari simbol yang sama dalam produksi, aturan semantik dilambangkan sebagai berikut:

P1 : E → E+ T

E[0].tree = plus(E[1].tree,T.tree)

P2 : E → T

E.tree = T.tree

*P3 : T → T * F*

T[0].tree = mult (T[1].tree, F.tree)

P4 : T → F

T.tree = F.tree

P5 : F → const

F.tree = int (const.val)

P6 : F → var

F.tree = var (var.id)

P7 : F → (E)

F.tree = E.tree

Indeks dihilangkan jika simbol muncul hanya sekali. Jika sebuah simbol muncul beberapa kali, indeks 0 mengidentifikasi kemunculan dari sisi kiri, sementara semua kemunculan di sisi kanan produksi diindeks secara berturut-turut mulai dari 1.

Dalam contoh konkret tata bahasa atribut, kami akan secara konsisten menggunakan konvensi dari Contoh 14.2, sedangkan konvensi untuk mengatasi kemunculan simbol di sebuah produksi $p : X_o \rightarrow X_1....X_k$ melalui *p[0],...,p[k]* seperti pda contoh 14.1 lebih nyaman untuk penalaran konseptual.

Atribut semantik untuk setiap simbol yang disediakan oleh generator LR dapat berupa digunakan oleh parser itu sendiri untuk membangun representasi pohon sintaks. Atribut tata bahasa menggeneralisasi gagasan ini dalam dua arah. Pertama, setiap simbol mungkin memiliki beberapa atribut. Kedua, atribut di sisi kiri belum tentu ditentukan oleh sarana atribut simbol di sisi kanan produksi. Dan nilai atribut sisi kanan sekarang dapat ditentukan dengan menggunakan nilai atribut sisi kiri atau atribut simbol sisi kanan lainnya.

Atribut individu *p[i].a* dari sebuah kemunculan *p[i]* sebuah simbol dalam produksi *p* disebut kemunculan atribut *a* di *p*. Kemunculan atribut milik tata bahasa atribut dan digunakan untuk spesifikasi perilaku lokal di sebuah node. Jika aturan yang menentukan untuk kejadian atribut *o* mengakses kejadian atribut lain *o'*, maka *o* tergantung secara *fungsional* pada *o'*.

Atribut kemunculan simbol di pohon sintaks, di sisi lain, disebut contoh atribut. Contoh atribut ada pada waktu kompilasi setelahnya Analisis sintaksis telah menghasilkan pohon parse.

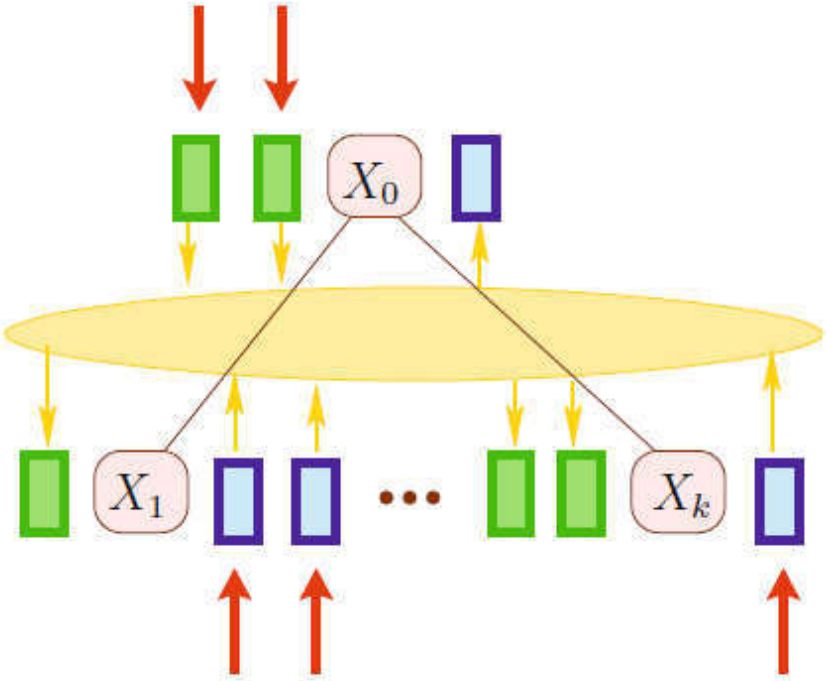
Ketergantungan fungsional antara kejadian atribut menentukan di mana memesan contoh atribut pada node dari pohon parse dapat dievaluasi. Argumen aturan semantik perlu dievaluasi sebelum aturan dapat diterapkan untuk menghitung nilai atribut terkait (instance). Batasan yang sesuai pada fungsional dependensi memastikan bahwa aturan semantik lokal dari tata bahasa atribut untuk kejadian atribut dalam produksi dapat disusun ke komputasi global dari semua contoh atribut dalam pohon parse. Nilai-nilai contoh atribut di node individu dari pohon parse dihitung oleh algoritma global, yaitu dihasilkan dari tata bahasa atribut, dan yang di setiap node n menganut aturan semantik lokal produksi diterapkan di n. Bab ini akan membahas

pertanyaan-pertanyaan bagaimana algoritma semacam itu dapat secara otomatis dihasilkan dari suatu pemberian atribut tata bahasa.

Atribut juga memiliki arahan, yang bisa lebih dipahami jika kita memikirkannya aplikasi produksi di pohon parse. Atribut simbol X dapat diwariskan atau disintesis. Nilai (contoh) atribut yang disintesis pada sebuah node dihitung dari (nilai instance atribut di) subpohon di node ini. Itu nilai dari (contoh) atribut yang diwariskan pada sebuah node dihitung dari konteksnya node. Semua contoh atribut dengan tepi kuning masuk adalah dihitung dalam produksi. Ini adalah kemunculan atribut yang disintesis dari sisi kiri produksi dan kejadian atribut yang diwariskan dari sisi kanan produksi. Bersama-sama kita menyebutnya mendefinisikan kemunculan atribut dalam produksi ini. Semua atribut kemunculan lainnya dalam produksi dipanggil kejadian atribut yang diterapkan. Setiap produksi memiliki aturan semantik, yang menjelaskan bagaimana nilai-nilai dari kejadian atribut yang menentukan dalam produksi dihitung dari nilai kemunculan atribut lain dari produksi yang sama. Jadi, semantic aturan perlu diberikan untuk setiap kejadian atribut yang diwariskan di sisi kanan produksi dan setiap kejadian atribut yang disintesis di sisi kiri. Sekumpulan dari atribut yang diwariskan dan disintesis dari tata bahasa atribut dilambangkan dengan I dan S, masing-masing, dan himpunan atribut yang diwariskan dan disintesis dari simbol X oleh $I(X)$ dan *S(X)*.

Tata bahasa atribut dalam bentuk normal jika semua atribut yang menentukan terjadi di produksi hanya bergantung pada kejadian yang diterapkan dalam produksi yang sama. Jika tidak secara eksplisit menyatakan sebaliknya, kami berasumsi bahwa tata bahasa atribut dalam bentuk normal.

Definisi kami juga memungkinkan atribut yang disintesis untuk simbol terminal dari tatabahasa. Dalam desain kompiler, tata bahasa atribut digunakan untuk menentukan analisis semantik. Fase ini mengikuti analisis leksikal dan sintaksis. Dalam analisis semantik, file atribut yang disintesis dari simbol terminal memainkan peran penting. Atribut khas yang disintesis dari simbol terminal adalah nilai konstanta, representasi eksternal atau pengkodean unik nama, dan alamat konstanta string. Nilai atribut ini diproduksi oleh pemindai, setidaknya jika diperluas oleh semantic fungsional.



Gambar 14.1 Node yang didistribusikan dalam pohon parse dengan penerusnya yang dikaitkan.

Keterangan Contoh yang diwariskan atribut digambar sebagai sebagai kotak disebelah kiri simbol sintaks, contoh stribut yang disintesiskan sebagai kotak disebelah kanan simbol. Panah merah (lebih gelap) menunjukkan aliran informasi kedalam produksi misalnya dari luar, panah kuning (lebih terang) melambangkan ketergantungan fungsional antara contoh atribut yang diberikan melalui aturan semantik yang terkait produksi.

Untuk instance atribut yang juga diwariskan pada root pohon parse, tidak ada aturan semantik yang disediakan oleh tatabahasa untuk menghitung nilainya. Disini aplikasi harus memberikan nilai yang berarti untuk inisialisai mereka.

2. Atribut Dinamis

Bagian ini membahas evaluasi atribut, lebih tepatnya evaluasi contoh atribut dalam pohon parse, dan pembuatan evaluator yang sesuai. Tata bahasa atribut mendefinisikan untuk setiap pohon parse t dalam CFG yang mendasari sistem persamaan AES(t), sistem evaluasi atribut. Yang tidak diketahui dalam

sistem persamaan ini adalah contoh atribut pada simpul t. Mari kita asumsikan bahwa tatabahasa atribut dibentuk dengan baik. Dalam hal ini, sistem persamaan tidak bersifat rekursif sehingga dapat diselesaikan dengan metode eliminasi. Setiap langkah eliminasi memilih satu contoh atribut untuk dievaluasi berikutnya yang hanya harus bergantung pada contoh atribut yang nilainya telah ditentukan. Penilai atribut seperti itu murni dinamis jiika tidak mengeksploitasi informasi apapun tentang ketergantungan dalam atribut grammar.

Evaluasi atribut dinamis yang cukup evisien untuk tata bahasa atribut yang dibentuk dengan baik diperoleh jika contoh atribut dievaluasi sesuai permintaan. Evaluasi berdasarkan permintaan berarti bahwa tidak semua contoh atribut menerima nilainya. Sebaliknya, nilai contoh atribut dievaluasi hanya ketika nilainya dipertanyakan.

Evaluasi yang digerakkan oleh permintaan ini dilakukan oleh penyelesaian fungsi rekursif, yang disebut node *n* dan salah satu atribut *a* dari simbol yang memberi label n. Evaluasi dimulai dengan memeriksa apakah instance atribut yang ditanyakan *n.a* sudah menerima nilainya. Jika demikian, fungsi rekursif, yang disebut node n dan salah satu atribut a dari simbol yang memberi label n. Evaluasi dmulai dengan memeriksa apakah instance atribut yang dinyatakan n.a sudah menerima nilainya. Jika demikian, fungsi akan kembali dengan nilai yang telah dihitung. Jika tidak, n.a akan dihitung. Evaluasi ini pada gilirannya dapat menanyakan apakah nilai dari instance atribut lain, yang evaluasinya dipicu secara rekrusif. Strategi ini memiliki konsekuensi bahwa untuk setiap instance atribut dalam pohon parse, sisi kanan aturan semantiknya dievaluasi paling banyak sekali.

Evaluasi contoh atribut yang tidak pernah diminta dihindari. Untuk mewujudkan ide ini semua contoh atribut yang diinisialisasi dengan non-undef *d* diatur kenilai *d.* Untuk navigasi pada pohon parse kita mneggunakan operator postfix [i] untuk berpindah dari simpul n ke penerusnya yang ke-i. Untuk i=0 navigasi tetap di n. Lebih jauh, kita membutuhkan pangkal operator yang kita beri simpul n, mengembalikan nilai pasangannya (n', j) terdiri dari pangkal n' simpul n dan informasi kearah mana, dilihat dari n' untuk mencari n. Informasi terakhir ini mengatakan turunan dari pangkal n' node argumennya. Untuk

mengimplementasikan penyelesaian fungsi evaluasi rekursif, kita membutuhkan fungsi evaluasi. Jika p adalah produk yang diterapkan pada node n, dan jika

$$f(p[i_1].a_1,.....,p[i_r].a_r)$$

adalah sisi kanan dari aturan semantik untuk kejadian atribut p[i].a, sama dengan n(i,a) mengembalikan nilai f, dimana untuk setiap instance atribut yang diminta, penyelesaian fungsi dipanggil. Oleh karena itu, dapat didefinisikan:

$$eval\ n\ (i,a) = f\ (solve\ n[i_1]a_1,.....,solve\ n[i_r]a_r)$$

dalam rekrusi simultan dengan fungsi eval, penyelesaian fungsi diimplementasikan oleh:

```
solve n a = match n.a
        with value d → d
        undef → if b ∈ S(symb(n))
                then let d = eval n (0,a)
                        in let _ =n.a ← value d
                        in d
                        else let (n' ,j') = pangkal n
                        in let d' = eval n' (j', a')
                        in let _ = n.a ← value d'
                        in d'
```

fungsi penyelesaian memeriksa apakah instance atribut n.a dipohon parse sudah memiliki nilai. Jika instance n.a belum memiliki nilai maka, n.a diberikan label undef. Dalam hal ini aturan semantik untuk n.a dicari. Jika a adalah atribut yang disintetiskan dari simbol di node n, aturan semantik untuk a disediakan oleh produksi p di node n. Sisi kanan f aturan ini dimodifikasi sedemikian rupa sehingga secara langsung mencoba mengakses instance atribut argumennya, tetapi memanggil fungsi penyelesaian secara rekursif untuk instance ini pada node n.

Jika nilai d untuk contoh atribut n.a diperoleh, itu ditugaskan ke contoh atribut n.a dan sebagai tambahan dikembalikan sebagai hasil. Sebaliknya, jika a adalah atribut yang diwariskan dari simbol pada node n, aturan semantik

untuk n.a tidak disediakan oleh produksi pada n, tetapi oleh produksi pada pangkal dari n'.

Misalkan n' adalah pangkal dari n dan n menjadi turunan ke 0 dari aturan n'. Aturan semantik untuk kejadian atribut *p'[j'].a* dipilih jika produksi p' diterapkan pada node n'. Sisi kanannya lagi-lagi dimodifikasi dengan cara yang sama sehingga sebelum akses ke nilai atribut apapun, fungsi penyelesaian dipanggil.

Nilai yang dihitung kembali disimpan dalam contoh atribut n.a dan dikembalikan sebagai hasilnya. Jika tatabahasa atribut dibentuk dengn baik, evaluator berdasarkan permintaan selalu menghitung untuk setiap pohon parse dan untuk setiap contoh atribut nilai yang benar. Jika tatabahasa atribut tidak dibentuk dengan baik, sistem evaluasi atribut untuk beberapa pohon parse mungkin rekursif.

Jika t adalah pohon parse, ada simpul n dan atribut a pada n di t sedemikian rupa sehingga n.a bergantung, secara langsung atau tidak langsung pada dirinya sendiri terdapat pada penyelesaian n a tidak mungkin berakhir. Untuk menghindari nontermination, instance atribut diberi label dengan called jika evaluasinya telah dimulai, tetapi belum dihentikan. Selain itu, penyelesaian fungsi dimodifikasi untuk menghentikan dan mengembalikan beberapa nilai kesalahan setiap kali memenuhi instance atribut yang diberi label called.

3. Atribut Statis

Evaluasi atribut dinamis tidak memanfaatkan informasi tentang tata bahasa atribut untuk meningkatkan efisiensi evaluasi atribut. Metode evaluasi atribut yang lebih efisien dimungkinkan jika pengetahuan tentang ketergantungan fungsional dalam produksi diperhitungkan. Kemunculan atribut p'[i'].a dalam produksi p secaa fungsional bergantung pada kejadian p'[j'].b jika p'[j'].b adalah argumen dari aturan semantik untuk p'[i'].a. ketergantungan produksi lokal menetukan ketergantungan dalam sistem persamaan AES(t).

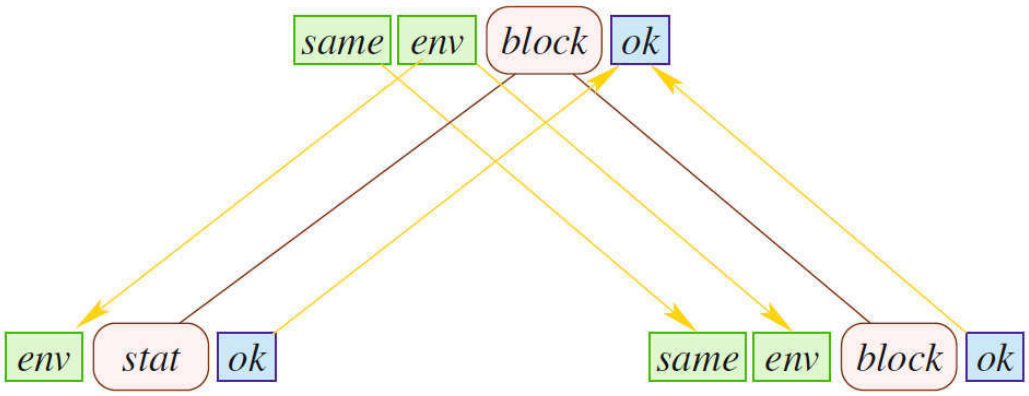
Berdasarkan ketergantungan fungsional, terkadang atribut dapat dievaluasi sesuai dengan urutan kunjungan yang ditentukan secara statis.

Urutan kunjungan menjamin bahwa contoh atribut hanya dijadwalkan untuk evaluasi ketika contoh argumen untuk aturan semantik terkait sudah dievaluasi.

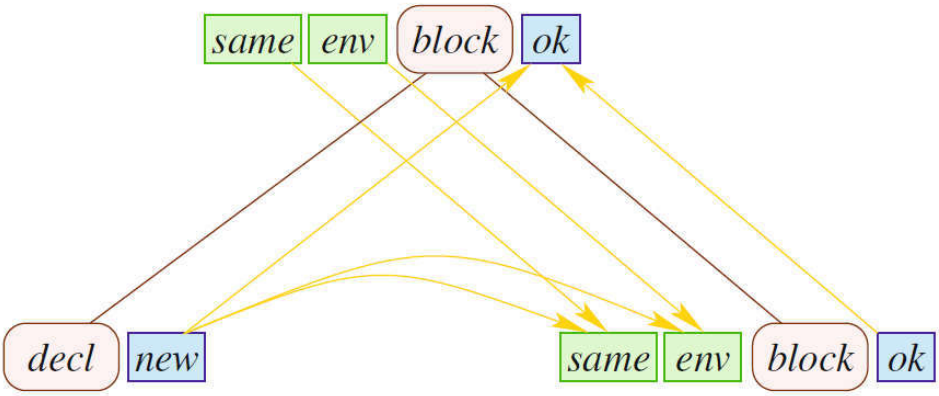
Evaluasi atribut membutuhkan kerjasama dari komputasi lokal pada node n dan penerusnya $n_1,.....,n_k$ dan yang ada dalam konteks contoh roduksi ini. Perhitungan lokal dari instance atribut yang disintetiskan pada node n berlabel X_o memberikan nilai atribut untuk digunakan oleh komputasi lokal pada n dalam konteks diatas.

Perhitungan nilai turunan atribut yang diwariskan pada node n yang sama terjadi pada induk n dan dapat memungkinkan evaluasi lebih lanjut sesuai dengan aturan semantik produksi yang sesuai dengan n. Pertukaran data serupa terjadi melalui contoh atribut di node $n_1,.....,n_k$ dengan perhitungan dalam cabang.

Untuk menjadwalkan interaksi komputasi ini, ketergantungan fungsional global



Gambar 14.2 hubungan ketergantungan antara (block) → (start) (block) pada AG_{scopes}



Gambar 14.3 hubungan ketergantungan antara (block) → (decl) (block) pada AG_{scopes}

Ketergantungan antara contoh atribut perlu diturunan dari ketergantungan produksi lokal.

Untuk produksi *p* biarkan *O[p]* menjadi himpunan kejadian atribut di *p*. Aturan semantik untuk produksi *p* mendefinisikan relasi $D(p) \subseteq O(p) \times O(p)$ ketergantungan fungsional lokal produksi pada himpunan *O(p)*. Relasi *D(p)* berisi pasangan *(p[j].b, p[i].a)* dari atribut muncul jika dan hanya jika *p[j].b* muncul sebagai argumen dalam aturan semantik untuk *p[i].a.*

C. SOAL LATIHAN

1. Jelaskan yang dimaksud dengan Attribute Grammars?
2. Jelaskan bagaimana mendeklarasikan attribute grammars?
3. Bagaimana cara untuk membedakan penilaian secara dinamis dan statis?
4. Bagaimana memeriksa perubahan pada grammars?
5. Jelaskan yang dimaksud dengan penilaian atribut secara dinamis?
6. Jelaskan yang dimaksud penilain atribut statis?
7. Buatlah contoh kerja analisa didalam teknik kompilasi menggunakan bahasa pemrograman sederhana.

D. REFERENSI

Alfred V. Aho, Monica S, Ravi Sethi, Jeffrey D. (2007). "*Compilers "Principles, Techniques, & Tools"*, 2nd Edition. Boston, San Fransisco, New York.

Dick Grune, Kees van Reeuwijk, Henri E. Bal, Ceriel J.H. Jacobs, Koen Langendoen. (2012). "*Modern Compiler Design"*. British Library Catalogue In Publication Data.

Kenneth C. Louden. Kenneth A. Lambert. (2011). "*Programming Languages Principles and Practices"*, Third Edition. USA.

Pattrick D. Terry (2000). "*Compilers and Compiler Generators an Introduction with C++".* Rhodes University.

Reinhard Wilhelm, Helmut Seidl, Sebastian Hack. (2013). "*Compilers Design "Syntatic and Semantic Analysis".* Sprinder-Verlag Berlin Heidelberg.

GLOSARIUM

Atribut adalah spesifikasi yang mendefinisikan properti dari sebuah objek, elemen, atau file. Ini juga dapat merujuk dan mengatur pada nilai spesifik dari suatu objek.

Grammars adalah model yang berguna saat merancang perangkat lunak yang memproses data dengan struktur rekursif.

Evaluasi adalah proses mendapatkan makna atau arti dari sebuah kode.