Bab 4

Parsing dan Syntax Analysis

Sub-CPMK yang Dicakup dalam Bab Ini:

- **Sub-CPMK 2.3:** Membangun recursive descent parser untuk grammar LL(1)
- **Sub-CPMK 2.4:** Menggunakan parser generator (Flex/Bison) untuk bahasa sederhana

4.1 Dasar Syntax Analysis dan CFG

Analisis sintaksis atau *parsing* bertujuan untuk memastikan bahwa urutan token yang dihasilkan oleh lexer sesuai dengan aturan tata bahasa (grammar) yang telah ditentukan. Dalam praktiknya, proses ini sering kali diotomatisasi menggunakan generator parser seperti GNU Bison [1, 2].

4.1.1 Context-Free Grammar (CFG)

CFG merupakan notasi formal untuk mendefinisikan bahasa. Sebuah CFG G didefinisikan sebagai 4-tuple (V, Σ, R, S) di mana:

- V: Himpunan variabel atau *non-terminals*.
- Σ: Himpunan *terminals* (token).
- R: Aturan produksi (*rules*).
- S: Simbol awal (*start symbol*).

4.1.2 Notasi BNF dan EBNF

BNF (Backus-Naur Form) menggunakan produksi eksplisit, sedangkan **EBNF (Extended BNF)** menambahkan operator seperti $\star$ (repetisi 0+), + (repetisi 1+), dan [ ] (opsional) untuk membuat grammar lebih ringkas.

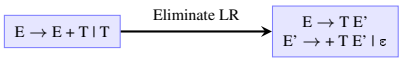
4.2 Top-Down Parsing dan LL(1)

4.2.1 Recursive Descent Parser

Merupakan parser top-down yang menggunakan sekumpulan prosedur rekursif untuk mengenali struktur bahasa pemrograman.

4.2.2 Eliminasi Masalah Grammar

Recursive descent tidak dapat menangani **Left Recursion** dan seringkali membutuhkan **Left Factoring** agar keputusan lookahead menjadi deterministik.



Gambar 4.1: Transformasi eliminasi Left Recursion

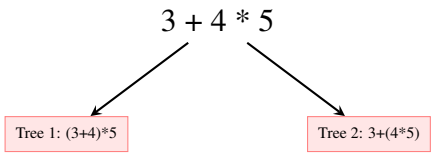
4.3 Struktur Derivasi dan Ambiguitas

4.3.1 Leftmost vs Rightmost Derivation

Leftmost derivation (LD) selalu mengganti non-terminal paling kiri, sering digunakan pada top-down parser. Sebaliknya, *Rightmost derivation* (RD) mengganti non-terminal paling kanan, umum pada bottom-up parser.

4.3.2 Ambiguitas Grammar

Grammar disebut ambigu jika satu input dapat menghasilkan lebih dari satu *parse tree*. Hal ini berbahaya karena dapat menyebabkan salah interpretasi logika program (misal: urutan operasi aritmatika).



Gambar 4.2: Ilustrasi ambiguitas grammar

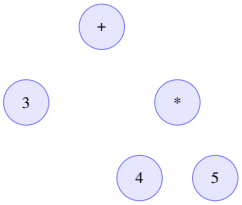
4.4 Parse Tree dan Abstract Syntax Tree (AST)

4.4.1 Parse Tree (Concrete Syntax Tree)

Representasi visual lengkap dari derivasi string, termasuk semua terminal dan non-terminal pendukung.

4.4.2 Abstract Syntax Tree (AST)

Versi ringkas dari parse tree yang hanya menyimpan informasi esensial untuk tahap kompilasi berikutnya (semantik dan *code generation*).



Gambar 4.3: Contoh struktur AST

4.5 Integrasi Lexer dan Parser (Bison)

Parser generator modern seperti **Bison** bekerja sama dengan **Flex** melalui mekanisme integrasi token.

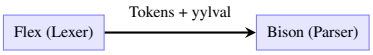
4.5.1 Mekanisme yylval

Lexer mengirimkan nilai semantik token (misal: nilai konstanta atau nama variabel) ke parser melalui variabel global `yylval`.

4.5.2 Struktur File Bison (.y)

```
%token NUMBER IDENTIFIER
%left '+' '-'
%left '*' '/'
```

```
%%
expr: expr '+' expr | NUMBER ;
%%
```



Gambar 4.4: Integrasi data antara Flex dan Bison

4.6 Implementasi: Hand-written Recursive Descent

Penerapan top-down parsing yang paling umum secara manual adalah melalui *Recursive Descent Parser*. Setiap *non-terminal* dalam grammar direpresentasikan oleh satu fungsi C++.

4.6.1 Arsitektur Parser

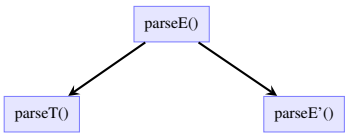
Fungsi-fungsi parser akan saling memanggil secara rekursif mengikuti struktur grammar. Terminal dicocokkan menggunakan fungsi `match(tokenType)`.

4.6.2 Contoh Implementasi

Berikut adalah kerangka parser sederhana untuk ekspresi:

```
void Parser::parseE() {
    parseT();
    parseEPrime();
}

void Parser::parseEPrime() {
    if (lookahead == '+') {
        match('+');
        parseT();
        parseEPrime();
    }
    // Epsilon case: do nothing
}
```



Gambar 4.5: Hierarki pemanggilan fungsi Recursive Descent

4.7 Precedence dan Error Recovery

4.7.1 Menangani Precedence

Dalam recursive descent, precedence diatur melalui level pemanggilan fungsi. Operator dengan precedence lebih tinggi (misal: perkalian) dipanggil lebih dalam dibanding operator dengan precedence rendah (misal: penjumlahan).

4.7.2 Error Recovery: Panic Mode

Agar parser tidak berhenti pada kesalahan pertama, kita menerapkan *Panic Mode Recovery*. Jika terjadi kesalahan, parser akan membuang token (*skipping*) sampai menemukan token sinkronisasi seperti `;` atau `}`.

```
void Parser::synchronize() {
    while (!isAtEnd()) {
        if (peek().type == SEMICOLON) return;
        switch (peek().type) {
            case CLASS: case FUN: case VAR: case IF: return;
        }
        advance();
    }
}
```

4.8 Bottom-Up Parsing: Shift-Reduce dan LR

4.8.1 Konsep Shift-Reduce

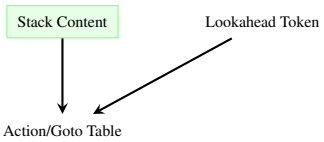
Bottom-up parsing bekerja dengan dua operasi utama:

- **Shift**: Memindahkan token dari input ke stack.
- **Reduce**: Mengganti simbol di puncak stack yang cocok dengan *handle* (RHS sebuah produksi) menjadi non-terminal (LHS).

4.8.2 LR Parsing

LR parsing (L: Left-to-right, R: Rightmost derivation in reverse) adalah metode yang paling powerful. Varian LR meliputi:

- **SLR(1)**: Simple LR, menggunakan *Follow set* dasar.
- **LALR(1)**: Look-Ahead LR, standar industri (digunakan Bison/Yacc).
- **Canonical LR(1)**: Paling powerful namun membutuhkan tabel sangat besar.



Gambar 4.6: Model konseptual LR Parser

4.9 Parser Generator: Bison Deep Dive

Integrasi antara Flex dan Bison merupakan standar industri dalam membangun front-end kompilator yang tangguh dan efisien [3]. Bison menghasilkan parser LALR(1) yang sangat efisien dari spesifikasi formal.

4.9.1 Resolusi Konflik

Bison dapat mendeteksi dua tipe konflik:

1. **Shift/Reduce Conflict**: Parser bingung apakah akan memasukkan token baru atau melakukan reduksi (misal: masalah *dangling else*).

2. **Reduce/Reduce Conflict**: Beberapa aturan produksi cocok dengan handle yang sama (biasanya menandakan grammar ambigu yang parah).

4.9.2 Manajemen Precedence di Bison

Bison memungkinkan pengaturan precedence tanpa mengubah grammar melalui direktif `%left`, `%right`, dan `%nonassoc`.

```
%left '+' '-'
%left '*' '/'
%right '^'
%%
expr: expr '+' expr | expr '*' expr | NUMBER ;
```

Dengan deklarasi di atas, Bison secara otomatis memproses `3 + 4 * 5` sebagai `3 + (4 * 5)` tanpa perlu grammar multi-level yang kompleks.

Aktivitas Pembelajaran

1. **Grammar Design**: Buat CFG untuk bahasa ekspresi aritmatika sederhana.

2. **LL(1) Conversion**: Konversi grammar dengan left recursion ke LL(1).

3. **Recursive Descent**: Implementasikan recursive descent parser untuk kalkulator.

4. **Bison Practice**: Buat parser untuk bahasa dengan assignment dan control flow.

5. **Error Recovery**: Implementasikan panic mode recovery dalam parser Anda.

Latihan dan Refleksi

1. Identifikasi apakah grammar berikut LL(1): $S \rightarrow aSb|\epsilon$!

2. Hapus left recursion dari grammar: $E \rightarrow E + T|T$!

3. Buat recursive descent parser untuk grammar: $S \rightarrow (S)S|\epsilon$!

4. Jelaskan perbedaan antara SLR(1) dan LALR(1) parser!

5. Implementasikan error recovery with panic mode untuk parser Anda!

6. **Refleksi**: Konsep parsing mana yang paling sulit dan bagaimana cara memahaminya?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 2.3-2.4

A. Pilihan Ganda

1. Grammar dengan left recursion TIDAK cocok untuk:
   (a) LL(1) parser
   (b) LR(1) parser
   (c) Recursive descent parser
   (d) Top-down parser

2. Tool yang digunakan untuk generate LR parser adalah:
   (a) Flex
   (b) Bison
   (c) Make
   (d) GCC

3. Lookahead dalam LL(1) berarti:

(a) 1 token lookahead

(b) 1 character lookahead

(c) 1 production lookahead

(d) 1 derivation lookahead

B. Essay

1. Jelaskan langkah-langkah mengkonversi grammar dengan left recursion ke LL(1)!
2. Desain dan implementasikan parser untuk bahasa dengan variabel assignment dan arithmetic expressions!

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat membangun recursive descent parser untuk grammar LL(1)

☐ Saya dapat menggunakan parser generator (Flex/Bison) untuk bahasa sederhana

☐ Saya dapat mengidentifikasi apakah suatu grammar LL(1)

☐ Saya dapat menghapus left recursion dari grammar

☐ Saya dapat mengimplementasikan error recovery dalam parser

☐ Saya memahami perbedaan top-down dan bottom-up parsing

Rangkuman

Bab ini membahas parsing dan syntax analysis, termasuk CFG, LL(1) parsing, recursive descent, LR parsing, dan parser generator tools. Mahasiswa belajar membangun parser manual dan menggunakan tools modern.

Poin Kunci:

- Parser memverifikasi sintaks dan membangun parse tree dari token stream
- LL(1) grammar cocok untuk recursive descent parser
- LR parser lebih powerful tapi lebih kompleks
- Parser generator (Bison) otomatisasi pembuatan parser dari grammar

- Error recovery penting untuk parser yang robust

Kata Kunci: ***Parsing***, ***CFG***, ***LL(1)***, ***Recursive Descent***, ***LR Parser***, ***Bison***, ***Syntax Analysis***, ***Error Recovery***

Daftar Pustaka

[1] John R. Levine. *flex & bison: Text Processing Tools*. O'Reilly Media, 2009.

[2] GNU Project. *GNU Bison Manual*. Official Bison documentation. 2024. URL: https://www.gnu.org/software/bison/manual/.

[3] IT Trip. *C Parser Flex Bison*. Tutorial. 2024. URL: https://en.ittrip.xyz/c-language/c-parser-flex-bison.