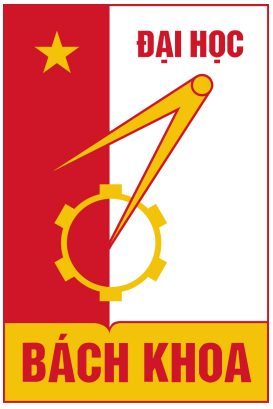
Trường đại học bách khoa hà nội



Báo cáo bài tập lớn môn học:

Lý thuyết ngôn ngữ và phương pháp dịch

TÌM HIỂU BỘ SINH TRÌNH PHÂN TÍCH CÚ PHÁP BISON

Sinh viên thực hiện:
Tạ Quang Tùng
MSSV: 20154280

Giáo viên hướng dẫn:
TS. Phạm Đăng Hải

Hà Nội, Ngày 18 tháng 11 năm 2018

Mục lục

1 Giới thiệu về bộ sinh trình phân tích cú pháp Bison

GNU Bison, hay thường được biết đến là Bison, là bộ sinh trình phân tích cú pháp parser, là một phần của dự án GNU. [3]

Bison đọc một file đặc tả của một ngôn ngữ phi ngữ cảnh (context-free language), cảnh báo về những nhập nhằng trong quá trình phân tích, và sinh ra một bộ phân tích cú pháp (Có thể bằng ngôn ngữ C, C++ hoặc Java).

Bison mặc định sinh ra LALR parser, nhưng cũng có thể tạo ra GLR parser.

Trong chế độ POSIX, Bison tương thích với Yacc, cùng với một vài những mở rộng. Đồng thời Flex, bộ sinh trình phân tích từ vựng, cũng thường được sử dụng cùng với Bison để cung cấp chuỗi các token đầu vào cho việc phân tích cú pháp.

Bison là một phần mềm miễn phí, mã nguồn mở được phát hành dưới giấy phép GPL.

2 Cấu trúc của file văn phạm trong Bison

2.1 Các thành phần của một file văn phạm

Một file mô tả văn phạm của Bison được chia thành 4 phần như sau: [4]

```
%{
    Prologue
%}

Bison declarations

%%
Grammar rules
%%

Epilogue
```

Trong đó, phần Prologue thường chứa các khai báo hàm, include của C.
Phần Bison Declarations chứa các khai báo được Bison sử dụng.
Phần Grammar Rules chứa các quy tắc ngữ pháp của ngôn ngữ đang cần biểu diễn và các lệnh thực hiện tương ứng.
Phần Epilogue thường chứa định nghĩa của các hàm C tương ứng với phần Prologue.

2.2 Phần Prologue

Phần Prologue chứa các định nghĩa macro và các khai báo của các hàm và biến được sử dụng trong các câu lệnh của phần Grammar Rules. [6]

Những đoạn code trong phần này sẽ được sao chép vào phần đầu file mã nguồn của chương trình dịch được sinh ra. Có thể sử dụng **#include "..."** để include các file header tương ứng.

Có thể có nhiều hơn một phần Prologue nằm đan xen với các phần Bison Declarations.

2.3 Phần Bison Declarations

Phần Bison Declarations định nghĩa các kí hiệu được sử dụng để xây dựng ngữ pháp chương trình và kiểu dữ liệu của các giá trị. [1]

Tất cả token (ngoại trừ các token có một kí tự như '+' hoặc '*') phải được khai báo trước khi có thể sử dụng. Các kí hiệu không kết thúc phải được khai báo nếu muốn định nghĩa kiểu của kí hiệu đó để sử dụng làm giá trị ngữ nghĩa.

Phần Bison Declarations cũng chỉ định kí hiệu không kết thúc bắt đầu của văn phạm. Mặc định là sử dụng kí hiệu bên trái của rule đầu tiên trong phần Grammar Rules.

Các khai báo trong phần Bison Declarations bao gồm:

- **Khai báo require**: Có thể chỉ định version nhỏ nhất của Bison trong khi xử lý file ngữ pháp. Nếu như yêu cầu về version không đủ, Bison dừng việc xử lý và báo lỗi.

```
%require ''version''
```

- **Khai báo token**: Để khai báo tên của token (kí tự kết thúc), ta sử dụng: [10]

```
%token name
```

  Bison sẽ chuyển khai báo token thành một hằng số trong file mã nguồn đầu ra (file header) để mà flex có thể sử dụng **name** để làm giá trị token cần trả về của hàm **yylex()**

  Có thể chỉ định giá trị số của token bằng cú pháp:

```
%token NUM 300
```

  Trong đó 300 là giá trị số của token **NUM**

  Nhưng tốt hơn cả là để Bison lựa chọn giá trị số cho tất cả các token. Bison sẽ tự động chọn các số mà không xung đột với giá trị của các kí tự ASCII và các token trước đó. (Giá trị Bison chọn luôn lớn hơn 255).

- **Khai báo union và kiểu dữ liệu**: Trong hầu hết các chương trình sẽ cần nhiều hơn một kiểu dữ liệu cho các token khác nhau. [5] Như giữa token đại diện cho số và token đại diện cho xâu.

  Mặc định Bison chỉ có một kiểu giá trị ngữ nghĩa là int. Để sử dụng nhiều hơn một kiểu dữ liệu cho các giá trị ngữ nghĩa, Bison yêu cầu trong file ngữ pháp phải có:

  - Chỉ định toàn bộ tập các kiểu dữ liệu sẽ được sử dụng (có thể sử dụng **%union**)
  - Trong các kí hiệu mà giá trị ngữ nghĩa được sử dụng (kết thúc hoặc không kết thúc), Chọn một trong các kiểu được chỉ định ở trên. (Sử dụng khai báo **%token** bổ sung kiểu cho các kí hiệu kết thúc, sử dụng **%type** cho các kí hiệu không kết thúc)

  Để khai báo union, ta dùng cú pháp:

```
%union {
    int num;
    char *name;
}
```

  Để khai báo token có chứa kiểu, ta dùng cú pháp:

```
%token <num> NUMBER
```

  Trong đó token **NUMBER** có kiểu là kiểu của biến **num** trong **union**.

  Để khai báo kí hiệu không kết thúc có chứa kiểu, ta dùng:

```
%type <num> NONTERMINAL
```

Trong đó kí hiệu không kết thúc **NONTERMINAL** có kiểu là kiểu của biến **num** trong **union**. [11]

- **Khai báo kí hiệu bắt đầu văn phạm**: [9] Bison mặc định rằng kí hiệu bắt đầu văn phạm là kí tự không kết thúc đầu tiên bên trái trong Phần Grammar Rules. Có thể thay đổi kí hiệu này bằng:

```
%start symbol
```

2.4 Phần Grammar Rules

Phần khai báo ngữ pháp của Bison bao gồm các grammar rule có dạng tổng quát như sau: [8]

```
result: components...;
```

Trong đó **result** là kí hiệu không kết thúc mà rule này mô tả, và **components** là một chuỗi các kí hiệu kết thúc và không kết thúc tạo nên một sản xuất.

Ví dụ:

```
exp: exp '+' exp;
```

Mô tả một sản xuất $exp \to exp + exp$. Các kí tự dấu trắng không ảnh hưởng đến định nghĩa các rule.

Giữa các **component** có thể chứa các **action** dùng để định nghĩa ngữ nghĩa của rule đó. Một **action** có dạng:

```
{C statements}
```

C statements là một tập các câu lệnh của C được bao bọc trong cặp ngoặc nhọn, các lệnh này là các lệnh giống như được sử dụng bên trong định nghĩa hàm C. Bison không kiểm tra tính đúng đắn của các lệnh C này, chỉ copy nguyên vẹn vào file mã nguồn đầu ra.

Nhiều các **rule** cho cùng một **result** có thể được định nghĩa bằng cách kết nối chúng bằng kí tự '|':

```
result  :   rule1-components...
        |   rule2-components...
        ...
        ;
```

Ví dụ:

```
E   :   E '+' T     { printf(``E\n'');
    |   T           { printf(``T\n'');
    ;
```

Mô tả một sản xuất $E \to E + T|T$.

Một **rule** là trống rỗng nếu như bên phải của nó (các **component**) là trống rỗng. [2] Nó giống như kí hiệu ε thể hiện một xâu rỗng.

Ví dụ:

```
E   :   TE1
E1  :   '+' T E1
    |
    ;
```

Mô tả hai sản xuất là $E \to TE_1$ và $E_1 \to +TE_1|\varepsilon$

Tuy nhiên việc không sử dụng kí tự gì khiến việc xác định rule trống rỗng khó khăn, ta có thể sử dụng **%empty**:

```
E    :    TE1
E1   :    '+' T E1
     |    %empty
     ;
```

Tuy nhiên **%empty** là một mở rộng của Bison và không tồn tại trên Yacc.

Trong Bison ta nên khai báo các rule (hay các sản xuất) dưới dạng đệ quy trái thay vì đệ quy phải để tránh sử dụng stack trong quá trình hoạt động của parser. [7]

Tài liệu tham khảo

[1] Declarations (Bison 3.2.1). https://www.gnu.org/software/bison/manual/html_node/Declarations.html#Declarations.

[2] Empty Rules (Bison 3.2.1). https://www.gnu.org/software/bison/manual/html_node/Empty-Rules.html#Empty-Rules.

[3] GNU Bison - Wikipedia. https://en.wikipedia.org/wiki/GNU_Bison.

[4] Grammar Outline (Bison 3.2.1). https://www.gnu.org/software/bison/manual/html_node/Grammar-Outline.html#Grammar-Outline.

[5] Multiple Types (Bison 3.2.1). https://www.gnu.org/software/bison/manual/html_node/Multiple-Types.html#Multiple-Types.

[6] Prologue (Bison 3.2.1). https://www.gnu.org/software/bison/manual/html_node/Prologue.html#Prologue.

[7] Recursion (Bison 3.2.1). https://www.gnu.org/software/bison/manual/html_node/recursion.html#recursion.

[8] Rules Syntax (Bison 3.2.1).

[9] Start Decl (Bison 3.2.1). https://www.gnu.org/software/bison/manual/html_node/Start-Decl.html#Start-Decl.

[10] Token Decl (Bison 3.2.1). https://www.gnu.org/software/bison/manual/html_node/Token-Decl.html#Token-Decl.

[11] Type Decl (Bison 3.2.1). https://www.gnu.org/software/bison/manual/html_node/Type-Decl.html#Type-Decl.