Đại học Bách Khoa Hà Nội

Khoa công nghệ thông tin và truyền thông

Tìm hiểu bộ phân tích cú pháp Bison

Giảng viên: Phạm Đăng Hải

Sinh viên: Nguyễn Tuấn Hưng – 20162031
Sinh viên: Nguyễn Tiến Tài – 20164837

Ngày 8 tháng 12 năm 2019

Mục lục

1 Giới thiệu về Bison

Flex và Bison là các công cụ để xây dựng các chương trình xử lý các đầu vào có cấu trúc. Ban đầu chúng là những công cụ nhằm xây dựng các bộ dịch, nhưng sau này đã chứng minh được sự hiệu quả trong các lĩnh vực khác.

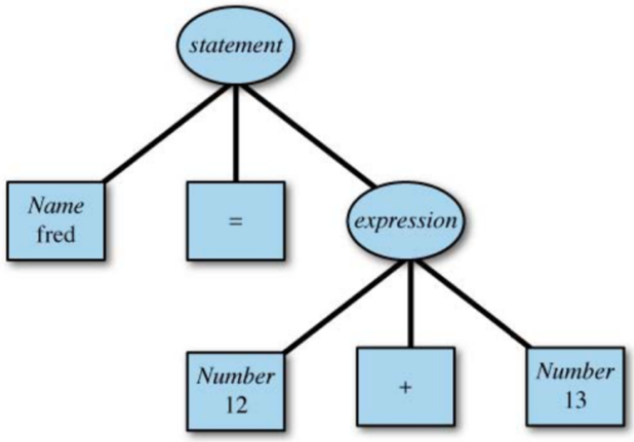
Bison là một trình tạo phân tích cú pháp là một phần của Dự án GNU. Bison đọc một đặc tả của ngôn ngữ không ngữ cảnh, cảnh báo về bất kỳ sự mơ hồ phân tích cú pháp nào và tạo một trình phân tích cú pháp (trong C, C ++ hoặc Java) để đọc các chuỗi mã thông báo và quyết định xem chuỗi có phù hợp với cú pháp được chỉ định bởi ngữ pháp hay không. Các trình phân tích cú pháp được tạo là di động: chúng không yêu cầu bất kỳ trình biên dịch cụ thể nào. Bison theo mặc định tạo ra các trình phân tích cú pháp LALR (1) nhưng nó cũng có thể tạo các trình phân tích cú pháp LR, ClassifiedR (1) và GLR chuẩn

Trong khi flex chỉ để nhận ra các biểu thức chính quy, bison nhận ra toàn bộ ngữ pháp. Flex chia luồng đầu vào thành các mảnh (mã thông báo), sau đó bison lấy các mảnh này và nhóm chúng lại với nhau một cách hợp lý

2 Cách sử dụng Bison

2.1 Cách một Bison Parser nối đến đầu vào của nó

Bison lấy một văn phạm cái mà bạn xác định và viết một parser nhận diện "các câu" có hiệu lực trong văn phạm đó. Chúng tôi sử dụng thuật ngữ câu ở đây theo cách khá chung chung cho văn phạm ngôn ngữ C, các câu là các chương trình C có giá trị cú pháp. Các chương trình có thể hợp lệ về mặt cú pháp nhưng không hợp lệ về mặt ngữ nghĩa, ví dụ, chương trình C gán một chuỗi cho một biến int. Bison chỉ xử lý cú pháp; xác nhận khác là tùy thuộc vào bạn.
Cách thường dùng để biểu diễn một câu được parse là sử dụng một cây. Ví dụ, nếu ta phân tích đầu vào của fred = 12 + 13 với văn phạm này, cây sẽ trông như sau:



Hình 1: Cây biểu diễn

Trong ví dụ này, 12 + 13 là một biểu thức, và fred = expression là một mệnh đề. Một bison parser không tự động tạo được cây này như một cấu trúc dữ liệu, mặc dù như ta sẽ thấy, nó không khó để tự làm. Mỗi ngữ nghĩa bao gồm một ký hiệu bắt đầu, cái mà phải ở gốc của cây phân tích. Trong văn phạm này, statement là ký hiệu bắt đầu. Các quy tắc có thể tham chiếu trực tiếp hoặc gián tiếp đến bản thân họ; khả năng quan trọng này làm cho nó có thể phân tích các chuỗi đầu vào dài tùy ý. Hãy để mở rộng ngữ pháp của chúng tôi để xử lý các biểu thức số học dài hơn.

3 Các đặc điểm kỹ thuật của Bison

3.1 Cấu trúc của một văn phạm Bison

Văn phạm của một chương trình Bison bao gồm 3 phần:

- Phần định nghĩa
- Phần các luật
- Phần các chương trình con của người sử dụng

```
... definition section ...
%%
... rules section ...
%%.
... user subroutines section ...
```

Các phần được chia bởi các dòng bao gồm 2 dấu phần trăm. Hai phần đầu là bắt buộc, mặc dù một phần có thể bị trống. Phần thứ ba và 2 ký hiệu %% đứng trước có thể bỏ qua.

3.2 Quy ước văn phạm YACC

Một văn phạm bison được cấu trúc từ các ký hiệu, được coi như các "từ"của văn phạm. Ký hiệu là chuỗi các ký tự, chữ số, dấu chấm, và các dấu gạch dưới không bắt đầu bởi một chữ số. Ký hiệu **error** được dành riêng để phục hồi lỗi; mặt khác, bison không có ý nghĩa cố định với bất kỳ biểu tượng nào. (Vì bison xác định ký hiệu tiền xử lý C cho mỗi mã thông báo, nên bạn cũng cần chắc chắn tên mã thông báo donith va chạm với các từ dành riêng C hoặc các ký hiệu riêng của bison, như **yyparse** hoặc các lỗi lạ sẽ xảy ra)

3.3 Phần định nghĩa

Phần định nghĩa có thể bao gồm một khối ngữ nghĩa, cái là phần mã C được sao chép nguyên văn tới phần đầu của file C được sinh, thường bao gồm phần khai báo và các dòng include trong % % hoặc các khối mã %. Có thể có *%union, %start, %token, %type, %left, %right và %nonassoc* các khai báo. Phần định nghĩa cũng có thể chứa các nhận xét ở định dạng C thông thường, được bao quanh bởi / * và * /. Tất cả những thứ này là tùy chọn, vì vậy trong một trình phân tích cú pháp rất đơn giản, phần định nghĩa có thể hoàn toàn trống.

3.4 Phần luật

Phần luật bao gồm các luật ngữ pháp và các hoạt động bao gồm mã C.

3.5 Phần chương trình

Bison sao chép nội dung của phần chương trình con người dùng nguyên văn vào tệp C. Phần này thường bao gồm các thói quen được gọi từ các hành động. Trong một chương trình lớn, thường sẽ thuận tiện hơn khi đặt mã hỗ trợ vào một tệp nguồn riêng biệt để giảm thiểu lượng tài liệu được biên dịch lại khi bạn thay đổi tệp bison.

4 Xử lý lỗi

Lỗi có thể xảy ra ở bất cứ tình huống hay file đầu vào nào mà thường thì việc xư lý lỗi thường khó khăn vì ta không dự đoán được trường hợp sẽ xảy ra.Có hai cách thông báo lỗi đó là thông báo ngay khi gặp lỗi và cách thứ hai là bỏ qua để tiếp tục phân tích và thông báo khi phân tích xong.

Dưới đây là phương thức ứng xử của YACC khi gặp lỗi.

Khi trình phân tích cú pháp đọc một luồng đầu vào, luồng đầu vào đó có thể không khớp với các quy tắc trong tệp ngữ pháp.

Trình phân tích cú pháp phát hiện vấn đề càng sớm càng tốt để thông báo cho người dùng và thay đổi file đầu vào input (source code). Nếu có một chương trình

con xử lý lỗi trong file ngữ pháp, trình phân tích cú pháp có thể cho phép nhập lại dữ liệu và người dùng sẽ thực hiện việc sửa lỗi file đầu vào, bỏ qua dữ liệu xấu hoặc bắt đầu hành động dọn dẹp và khôi phục. Ví dụ, khi trình phân tích cú pháp tìm thấy lỗi, có thể cần phải lấy lại lưu trữ cây phân tích cú pháp, xóa hoặc thay đổi các mục trong bảng ký hiệu và đặt các công tắc để tránh tạo thêm đầu ra.

Khi xảy ra lỗi, trình phân tích cú pháp dừng lại trừ khi bạn cung cấp các chương trình con xử lý lỗi. Để tiếp tục xử lý đầu vào để tìm thêm lỗi, hãy khởi động lại trình phân tích cú pháp tại một điểm trong luồng đầu vào nơi trình phân tích cú pháp có thể cố gắng nhận ra nhiều đầu vào hơn. Một cách để khởi động lại trình phân tích cú pháp khi xảy ra lỗi là loại bỏ một số mã thông báo sau lỗi. Sau đó thử khởi động lại trình phân tích cú pháp tại điểm đó trong luồng đầu vào.

5 Chạy thực nghiệm chương trình

File bison test.y

```
%{
/* Khai báo lại các biến từ flex */
#include <stdio.h>
int yylex();
int yyerror(char *s);
%}

/*Bison hoạt động bằng cách gọi đến flex để lấy token tiếp theo (dạng yystype).
yytype là một typedef của kiểu int , tuy nhiên token lại có thể là bất kì kiểu nào.
Để giải quyết ta sẽ ghi đè yystype trong mã C ở phần union trong file bison.
Union sẽ giữ các loại token mà flex sẽ trả lại. */
%union{
char name[20];
    int number;
}

/*Sử dụng chuẩn CAPS để liên kết mỗi kiểu với union*/
%token STRING NUM OTHER SEMICOLON
%type <name> STRING
%type <number> NUM
%%

/*Thực thi các hàm grammar mà bison sẽ ghép cặp.
Grammar là các bộ quy tắc xây dựng  */ //

prog:
  stmts;
```

```
stmts:
| stmt SEMICOLON stmts
stmt:
STRING {
printf("Chuỗi vừa nhập - %s", $1);
}
| NUM {
printf("Số vừa nhập - %d", $1);
}
| OTHER
;
%%
int main()
{
    yyparse();
    return 0;
}
```

File flex test.l

```
%{
// Khai báo hàm
#include <stdio.h>
#include <string.h>
#include "test.tab.h"
void showError();
%}

//Khai báo biến
numbers     ([0-9])+
alpha       ([a-zA-Z])+
%%

//yylval lấy từ bison
{alpha}    {sscanf(yytext, "%s", yylval.name); return (STRING);}
{numbers}      {yylval.number = atoi(yytext); return (NUM);}
";"          {return (SEMICOLON);}
.            {printf("Other input"); return(OTHER);}
%%
```

Chạy chương trình

```
 bison -d test.y
flex test.l
g++ test.tab.c lex.yy.c -lfl -o test
./test
```

6 Tài liệu tham khảo

- John R. Levine, Chapter 3: Using Bison.Flex bison - O'reilly
- Lan Gao, http://alumni.cs.ucr.edu/ lgao/teaching/bison.html,