

Báo cáo BTL 01

Vũ Minh Châu - 22021105

Hoàng Minh Tú - 21020029

Mục lục

1. Ngôn ngữ lập trình UPL	1
2. Đề xuất văn phạm phi ngữ cảnh	3
2.1. Symbols	3
2.2. Production Rule	3
2.3. Lời bình	4
3. Bộ phân tích từ vựng	5
3.1. Giới thiệu chương trình	5
3.2. Giới thiệu bộ phân tích từ vựng	6
3.3. Thí nghiệm	6

1. Ngôn ngữ lập trình UPL

Ngôn ngữ lập trình UPL có các đặc điểm như sau:

1. Bắt đầu và kết thúc chương trình bằng cặp từ khóa begin và end
2. Có hai kiểu dữ liệu là số nguyên (int) và boolean (bool)
3. id: bắt đầu bằng ký tự chữ, nếu có ký tự số thì phải ở cuối (có thể có nhiều)
4. Biến phải được khai báo trước khi sử dụng. Biến có thể được khởi tạo cùng với câu lệnh khai báo (sử dụng phép gán).
5. Có 3 phép toán so sánh: lớn hơn (>), lớn hơn hoặc bằng (>=), bằng (==)
6. Có phát biểu lựa chọn if then và if then else
7. Có phát biểu lặp do ...while
8. Có phát biểu for
9. Có phát biểu in ra màn hình (print) với tham số là một biểu thức
10. Các phép toán cộng (+) và nhân (*) cho số nguyên. * được ưu tiên hơn +. Lập trình viên có thể sử dụng dấu ngoặc () để xác định lại thứ tự tính toán
11. Có cơ chế comment như của Java (nghĩa là có cả /*...*/ và //...)

Ví dụ của một chương trình viết bằng UPL:

```
begin
    int x;
    int y=x+1;
    /* comments
    cho nhiều dòng
```

```
*/

bool a;//comment cho một dòng
if (x>a) then{
    int c=1;
}else{
    y=x;
    x=x+1;
}
print(a);
if (x>=a) then{
    x=x+1;
}
bool x=a==b;
do{
    int b=1;
    b=b*10;
    a=(b+10)*b;
}while(a>1);
print(a+1);
end
```

2. Đề xuất văn phạm phi ngữ cảnh

2.1. Symbols

Phân loại	Ký hiệu	Chú thích
	Program	Ký hiệu bắt đầu đại diện cho chương trình
Các loại câu lệnh	BracketStatementLists	Mục đích của symbol này là để tổng quát cho các tình huống có {} bao quanh code
	StatementLists	
	Statement	
	AssignmentStatement	
	IfStatement	
	DoWhileStatement	
	ForStatement	
	ElseStatement	
	PrintStatement	
	DeclarationStatement	
	ConditionalStatement	
	ComparationOperator	
Ký tự liên quan đến phép toán	Expression	
	Term	
	Factor	
	Literal	
	Type	
	ID	
Ký tự rỗng	ε	

Bảng 1: Danh sách các Symbol

2.2. Production Rule

LHS	RHS
Program	begin StatementLists end (1)
BracketStatementLists	{StatementLists} (2)
StatementLists	Statement StatementLists (3.1)
	ε (3.2)
Statement	AssignmentStatement (4.1)
	IfStatement (4.2)

LHS	RHS
	DoWhileStatement (4.3)
	ForStatement (4.4)
	PrintStatement (4.5)
	DeclarationStatement (4.6)
AssignmentStatement	ID = Expression ; (5)
IfStatement	if (ConditionalStatement) BracketStatementLists ElseStatement (6)
ElseStatement	else BracketStatementLists (7.1)
	ε (7.2)
PrintStatement	print (Expression) (8)
DeclarationStatement	Type ID; (9.1)
	Type AssignmentStatement; (9.2)
DoWhileStatement	do BracketStatementLists while (ConditionalStatement) (10)
ForStatement	for (DeclarationStatement;ConditionalStatement; Expression) BracketStatementLists (11)
Expression	Expression + Term (12.1)
	Term (12.2)
Term	Term * Factor (13.1)
	Factor (13.2)
Factor	ID (14.1)
	Literal (14.2)
	(Expression) (14.3)
ConditionalStatement	Expression ComparisonOperator Expression (15)
ComparisonOperator	> (16.1)
	>= (16.2)
	== (16.3)
Type	int (17.1)
	bool (17.2)
Literal	Number (18.1)
	true (18.2)
	false (18.3)

2.3. Lời bình

Yêu cầu 1 được thỏa mãn thông qua việc chỉ có một symbol bắt đầu là Program và luật (1).

Yêu cầu 2 được thỏa mãn thông qua luật (17.1) và (17.2), nếu như có thêm các kiểu biến khác thì có thể bổ sung cho Type.

Phần sau của yêu cầu 4 được thỏa mãn thông qua luật (9.1) và (9.2).

Yêu cầu 5 được thỏa mãn thông qua luật (16.1), (16.2), (16.3), chú ý rằng nếu như cần bổ sung thêm phép toán có thể bổ sung ở đây.

Yêu cầu 6 được thỏa mãn thông qua luật (6), (7.1), (7.2), chú ý rằng ở đây các câu lệnh rẽ nhánh chỉ gồm if hoặc if và else, nếu như muốn hỗ trợ if else thì chỉ cần thay đổi luật 6.

Yêu cầu 7 được thỏa mãn qua luật (10). Chú ý rằng ở đây chỉ có cấu trúc do while chứ không có while.

Yêu cầu 8 được giải quyết bởi luật (11), tuy nhiên hiện tại luật này có một bất cập về việc các tham số nên là optional, tuy nhiên để đơn giản hóa bài toán do ta có thể thêm các statement dummy không liên quan đến vòng for trong trường hợp không cần tham số.

Yêu cầu 9 được cung cấp bởi luật (8).

Yêu cầu 10 được đảm bảo thông qua các luật (12.1), (12.2), (13.1), (13.2), do ở đây chỉ có 2 phép toán là + và * nên ta lập ra các luật trực tiếp bằng tay, việc xây dựng luật theo cấu trúc được trình bày ở trên sẽ đảm bảo rằng phép toán * trong việc sinh ra syntax tree luôn sâu hơn phép toán +, như vậy thì phép * sẽ được ưu tiên hơn.

Về yêu cầu 3 và 11 thì ta xử lý bằng lexer thay vì sử dụng văn phạm. Trong đó phần đầu của yêu cầu 4 có thể được giải quyết thông qua việc yêu cầu mọi biến phải được khai báo trước nhưng điều này khá bất tiện nên không xử lý trong văn phạm.

3. Bộ phân tích từ vựng

3.1. Giới thiệu chương trình

- Link mã nguồn: <https://github.com/tu-hm/upl>
- Đây là chương trình thực hiện biên dịch ngôn ngữ lập trình UPL và hiện đã hoàn thành bộ phân tích từ vựng của chương trình
- Cách chạy chương trình: Chương trình lấy đầu vào thông qua tham số tên file, như vậy có thể chạy thông qua file jar bằng phương pháp sau:

```
java -jar compiler.jar path_to_file [--jflex]
```

Ngoài ra, do có cả module `jflex` và module xây dựng lexer và chương trình mặc định đang chạy module do nhóm tự lập trình, nếu như muốn đổi sang module sử dụng `jflex` thì cần sử dụng thêm tham số `--jflex`.

3.2. Giới thiệu bộ phân tích từ vựng

Dựa vào văn phạm thì có thể thấy trong bộ phân tích từ vựng thì ta sẽ có những token sau:

- Các từ khóa đặc biệt thể hiện giới hạn của chương trình: `begin`, `end`.
- Các từ khóa quan trọng khác: `if`, `then`, `else`, `print`, `do`, `while`, `true`, `false`.
- Các dấu quan trọng: `>`, `>=`, `==`, `=`, `(`, `)`, `{`, `}`, `;`, `+`, `*`.
- Các kiểu biến: `int`, `bool`.
- id thỏa mãn regex: `{Letter}+{Digit}*`, number thỏa mãn regex: `{Digit}+`.

Ngoài ra, do các comment sẽ được xử lý riêng bằng thuật toán tham lam để lấy nên các dấu cách và tab sẽ bị bỏ qua.

Tiếp theo nhóm sẽ trình bày về phương pháp lập trình sử dụng công cụ ở đây là `jflex` và không sử dụng công cụ.

Sử dụng công cụ `jflex`: Nhóm sử dụng dạng file gắn với bộ phân tích từ vựng này là `jflex`, file được lưu ở `jflex/JFlexScanner.jflex` và định nghĩa thành các thành phần như ở trên. Sau đó `jflex` sẽ tự sinh ra file `JFlexScanner.java` và sử dụng DFA để giải quyết bài toán.

Không sử dụng công cụ: Do tính chất đơn giản của của DFA trong bài toán này vì hầu hết ký tự bắt đầu của các token là khác nhau nên nhóm sẽ xử lý trực tiếp việc duyệt từng ký tự mà không thông qua xây dựng máy hữu hạn trạng thái. Thật vậy, khi chưa có token nào thì ta sẽ lấy ra một ký tự, nếu ký tự đó là dấu thì có thể dễ dàng kết luận đây là token gì, trừ trường hợp `==` thì ta sẽ xét riêng nếu gặp 1 dấu `=` thì cần kiểm tra ký tự đằng sau. Mặt khác, nếu ta gặp một số thì ta sẽ tách phần number, còn khi gặp chữ thì ta sẽ lấy đoạn dài nhất thỏa mãn regex của id là `{Letter}+{Digit}*`. Đến đây có 2 khả năng có thể xảy ra chính là phần vừa thu được là keyword hoặc thực sự là id, ta có thể kết luận thông qua một map của các keyword.

3.3. Thí nghiệm

Để thể hiện sự khác nhau giữa 2 phiên bản thì đối với `JFlex` ta sẽ in ra điểm bắt đầu của token với chỉ số là 0, trong khi đó với phiên bản làm thủ công thì sẽ in ra điểm cuối cùng. Chú ý rằng cả 2 phiên bản đều không đưa ra các token là comment vì như đã trình bày ở trên thì comment không ảnh hưởng đến cú pháp nên được xử lý riêng.

Nhóm chủ yếu sinh các dữ liệu bằng tay và trình bày kết quả thông qua chương trình sau:

```

begin

int x1=11;

bool y2=false;

for (int i = 3; i <= 5; i = i + 1) {
    // hmm
    int c3 = true;
}
hihi 2321 huhu;
do {
    print(c4 * c5 + c3);
} while (true);
end

```

Ví dụ đầu vào dòng 7 là một dòng phức tạp của cấu trúc for, đầu ra của chương trình thử công là:

```

Token: FOR at line 7 column 3
Token: LEFT_PAREN at line 7 column 5
Token: INT at line 7 column 8
Token: ID at line 7 column 10
Token: EQUAL at line 7 column 12
Token: NUMBER at line 7 column 14
Token: SEMICOLON at line 7 column 15
Token: ID at line 7 column 17
Token: EQUAL at line 7 column 20
Token: NUMBER at line 7 column 22
Token: SEMICOLON at line 7 column 23
Token: ID at line 7 column 25
Token: EQUAL at line 7 column 27
Token: ID at line 7 column 29
Token: PLUS at line 7 column 31
Token: NUMBER at line 7 column 33
Token: RIGHT_PAREN at line 7 column 34
Token: LEFT_BRACE at line 7 column 36

```

Còn đầu ra của chương trình jflex là:

```

Token: FOR at line 7 column 0
Token: LEFT_PAREN at line 7 column 4
Token: INT at line 7 column 5
Token: ID at line 7 column 9
Token: EQUAL at line 7 column 11
Token: NUMBER at line 7 column 13
Token: SEMICOLON at line 7 column 14
Token: ID at line 7 column 16
Token: EQUAL at line 7 column 19

```

Token: NUMBER at line 7 column 21
Token: SEMICOLON at line 7 column 22
Token: ID at line 7 column 24
Token: EQUAL at line 7 column 26
Token: ID at line 7 column 28
Token: PLUS at line 7 column 30
Token: NUMBER at line 7 column 32
Token: RIGHT_PAREN at line 7 column 33
Token: LEFT_BRACE at line 7 column 35

Có thể thấy rằng cả 2 phiên bản đều đã thực hiện được phân tích đối với dòng này. Ngoài ra nhóm cũng đã cung cấp thêm một vài dữ liệu thử nghiệm khác ở github.

Mặt khác, về mặt thời gian xử lý thì hai chương trình đều đang giải quyết bài toán với thời gian tuyến tính nên đối với các chương trình nhỏ thì gần như sẽ có kết quả ngay tức thời.