**Báo cáo giữa kỳ**

Nguyên lý các ngôn ngữ lập trình

Nhóm 8

* Bùi Tiến Dũng
* Phạm Vũ Anh Quân

**1. Phân tích cú pháp từ dưới lên là gì?**

* Phân tích cú pháp từ dưới lên bắt đầu với các ký hiệu đầu vào và cố gắng xây dựng cây phân tích cú pháp lên đến ký hiệu bắt đầu.
* Một trình phân tích cú pháp từ dưới lên bắt đầu từ các nút lá của cây phân tích cú pháp và hoạt động theo hướng đi lên cho đến khi nó đi đến nút gốc.

**2. Đặc điểm**

* Có kỹ thuật phân tích mạnh.
* Văn phạm lớp LR có khả năng mô tả tốt hơn văn phạm lớp LL, có thể mô tả văn phạm đệ quy trái.
* Dễ dàng mô tả các ngôn ngữ lập trình thông thường.

**3. So sánh phân tích cú pháp từ dưới lên với phân tích cú pháp từ trên xuống**

|  |  |
| --- | --- |
| Phân tích cú pháp từ dưới lên | Phân tích cú pháp từ trên xuống |
| - Xây dựng cây phân tích cú pháp từ dưới lên.  - Suy dẫn phải.  - Ngược lại so với phân tích từ trên xuống (Bắt đầu từ ký hiệu kết thúc, kết thúc tại ký hiệu bắt đầu).  - Phân tích từ dưới lên có nhiều thông tin hơn khi phân tích .  - Không cần sinh toàn bộ cây dẫn xuất trong quá trình phân tích. | - Xây dựng cây phân tích cú pháp từ trên xuống.  - Suy dẫn trái.  - Toàn bộ cây trên một ký hiệu được sinh ra (Xác định nút gốc của cây, sau đó di chuyển xuống các cây con cho đến khi tìm thấy ký hiệu kết thúc).  - Phải có khả năng đoán trước được các nút sinh ra. |

**4. Phân tích cú pháp Shift-Reduce**

* Sử dụng 2 hành động để phân tích cú pháp từ dưới lên, đó là shift và reduce.
* Shift: Đẩy ký hiệu vào ngăn xếp phân tích cú pháp. Ký hiệu được shift sẽ được coi như một nút duy nhất của cây phân tích cú pháp.
* Reduce: Thay thế a ở đỉnh của ngăn xếp bằng một non-terminal A, nếu như văn phạm của ta tồn tại luật sinh A → a. Nói cách khác, ta pop a rồi push A.
* Giúp phát hiện lỗi ngay khi nó xuất hiện và cho phép phục hồi khi lỗi ngay khi xảy ra.

**5. Trình phân tích cú pháp LR**

* Là trình phân tích cú pháp từ dưới lên, Shift-Reduce và không đệ quy.
* Nó sử dụng nhiều loại văn phạm phi ngữ cảnh, điều này làm cho nó trở thành kỹ thuật phân tích cú pháp hiệu quả nhất.
* Còn được gọi là trình phân tích cú pháp LR(k), trong đó:
  + L (left-to-right): Duyệt chuỗi đầu vào từ trái sang phải.
  + R (right-most derivation): Xây dựng dẫn xuất phải nhất.
  + k: Biểu thị số lượng nhìn trước để đưa ra quyết định. Nếu không được để cập đến, ta ngầm hiểu k = 1.
* Khái niệm LR item: Một luật sinh đang được phân tích sẽ sử dụng dấu . để tách phần trước với phần sau.
* Ví dụ luật sinh *expr -> expr add\_op term* sẽ có 4 LR item sau:
  + *expr -> . expr add\_op term*
  + *expr -> expr . add\_op term*
  + *expr -> expr add\_op . term*
  + *expr -> expr add\_op term .*
* Trình phân tích cú pháp họ LR theo dõi các trạng thái mà nó đi qua bằng cách đẩy các trạng thái đó vào ngăn xếp cùng với các ký hiệu văn phạm.

**6. Các thuật toán xây dựng trình phân tích cú pháp LR phổ biến**

* LR(0)
  + Thuật toán cơ bản, mọi thuật toán họ LR đều dựa trên nó.
* SLR(1)
  + Trình phân tích cú pháp simple LR.
  + Cải tiến một chút từ LR(0).
  + Triển khai đơn giản và nhanh chóng.
  + Hoạt động trên lớp văn phạm nhỏ nhất.
  + Rất ít trạng thái nên bảng sẽ nhỏ.
* LR(1)
  + Còn được gọi là LR chính tắc.
  + Làm việc trên bộ văn phạm LR(1) hoàn chỉnh.
  + Sử dụng được cho nhiều loại văn phạm.
  + Kích thước bảng và số lượng trạng thái lớn.
  + Triển khai chậm.
* LALR(1)
  + Trình phân tích cú pháp look-ahead LR.
  + Cân bằng giữa LR và SLR.
  + Phổ biến nhất trong thực tế.
  + Hoạt động trên kích thước trung bình của văn phạm.
  + Có kích thước và tốc độ tương đương với SLR nhưng có thể giải quyết được nhiều xung đột hơn.
* Các thành viên của họ phân tích cú pháp LR đều sử dụng một automaton, đó là máy trạng thái hữu hạn đặc trưng (CFSM).

**7. Bảng phân tích cú pháp**

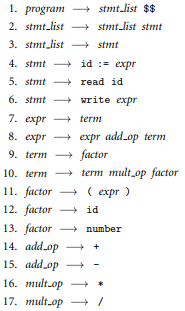
* Các trình phân tích cú pháp họ LR thực hiện kiểm tra bảng hai chiều để tìm kiếm hành động cần thực hiện.
* Thay vì chỉ sử dụng token đầu vào và non-terminal ở trên cùng của ngăn xếp, trình phân tích cú pháp họ LR sử dụng trạng thái hiện tại và token đầu vào để lập chỉ mục vào bảng. Ta tạm gọi action[s, a] là giá trị của trạng thái s và token a trong bảng.
  + action[s, a] có thể nhận 1 trong 4 giá trị sau:
    - shift s: đẩy ký hiệu input và trạng thái s vào trong stack.
    - reduce n: reduce bởi luật sinh thứ n trong văn phạm, trạng thái sẽ được đẩy ra khỏi ngăn xếp và non-terminal sẽ được đẩy trở lại chuỗi đầu vào.
    - accept: phân tích thành công.
    - error: phát hiện lỗi.
* Ngoài ra, ta cho phép bảng phân tích cú pháp chứa thêm giá trị “shift and then reduce n”, nghĩa là ta sẽ đẩy ký hiệu input vào và ngay sau đó reduce bởi luật sinh thứ n trong văn phạm.

**8. Các luật sinh Epsilon**

* Các epsilon trong các văn phạm phân tích từ dưới lên sử dụng quy tắc đệ quy trái.
* Một trạng thái vẫn là có ý nghĩa nếu nó có một danh sách câu lệnh trống.
* Trong ngôn ngữ máy tính, nó chỉ cho phép một chương trình trống. Tuy nhiên, trong ngôn ngữ thực, nó cho phép phần nội dung của một câu lệnh có cấu trúc.
* Ví dụ như trong cấu trúc của mệnh đề if … then … else thì câu lệnh else có thể để trống.

**7. Ví dụ minh họa**

Dựa vào văn phạm LR(1) cho ngôn ngữ máy tính (Hình 2.25, Trang 91)



và bảng phân tích cú pháp SLR(1) cho ngôn ngữ máy tính (Hình 2.28, Trang 98) trong tài liệu tham khảo

Table

Description automatically generated

(\*) s đại diện cho shift, r đại diện cho reduce và b đại diện cho shift and then reduce

Ta sẽ phân tích cú pháp cho chương trình dưới đây:

read A

res := A + 1

write res

Ta sẽ phân tích như sau:

|  |  |  |
| --- | --- | --- |
| Stack | Input stream | Comment |
| 0 | read A… |  |
| 0 read 1 | A res… | action[0, read] = s1 |
| 0 | stmt res… | action[1, id(A)] = b5 |
| 0 | smtm\_list res… | action[0, stmt] = b3 |
| 0 stmt\_list 2 | res :=… | action[0, stmt\_list] = s2 |
| 0 stmt\_list 2 id 3 | := A… | action[2, id(res)] = s3 |
| 0 stmt\_list 2 id 3 := 5 | A + 1… | action[3, :=] = s5 |
| 0 stmt\_list 2 id 3 := 5 | factor + 1… | action[5, id(A)] = b12 |
| 0 stmt\_list 2 id 3 := 5 | term + 1… | action[5, factor] = b9 |
| 0 stmt\_list 2 id 3 := 5 term 7 | + 1… | action[5, term] = s7 |
| 0 stmt\_list 2 id 3 := 5 | expr + 1… | action[7, +] = r7 |
| 0 stmt\_list 2 id 3 := 5 expr 9 | + 1… | action[5, expr] = s9 |
| 0 stmt\_list 2 id 3 := 5 expr 9 | add\_op 1… | action[9, +] = b14 |
| 0 stmt\_list 2 id 3 := 5 expr 9 add\_op 10 | 1 write… | action[9, add\_op] = s10 |
| 0 stmt\_list 2 id 3 := 5 expr 9 add\_op 10 | factor write… | action[10, number(1)] = b13 |
| 0 stmt\_list 2 id 3 := 5 expr 9 add\_op 10 | term write… | action[10, factor] = b9 |
| 0 stmt\_list 2 id 3 := 5 expr 9 add\_op 10 term 13 | write res… | action[10, term] = s13 |
| 0 stmt\_list 2 id 3 := 5 | expr write res… | action[13, write] = r8 |
| 0 stmt\_list 2 id 3 := 5 expr 9 | write res… | action[5, expr] = s9 |
| 0 stmt\_list 2 | stmt write res… | action[9, write] = r4 |
| 0 | stmt\_list write res… | action[2, stmt] = b2 |
| 0 stmt\_list 2 | write res… | action[0, stmt\_list] = s2 |
| 0 stmt\_list 2 write 4 | res $$ | action[2, write] = s4 |
| 0 stmt\_list 2 write 4 | factor $$ | action[4, id(res)] = b12 |
| 0 stmt\_list 2 write 4 | term $$ | action[4, factor] = b9 |
| 0 stmt\_list 2 write 4 term 7 | $$ | action[4, term] = s7 |
| 0 stmt\_list 2 write 4 | expr $$ | action[7, $$] = r7 |
| 0 stmt\_list 2 write 4 expr 6 | $$ | action[4, expr] = s6 |
| 0 stmt\_list 2 | stmt $$ | action[6, $$] = r6 |
| 0 | stmt\_list $$ | action[2, stmt] = b2 |
| 0 stmt\_list 2 | $$ | action[0, stmt\_list] = s2 |
| 0 | program | action[2, $$] = b1 |
| [done] |  |  |

Tài liệu tham khảo: Programming Language Pragmatics by Micheal Lee Scott.