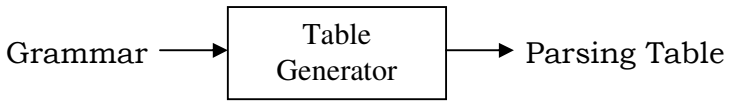
LR Parsers
(SLR, LALR, and Canonical LR Parser)

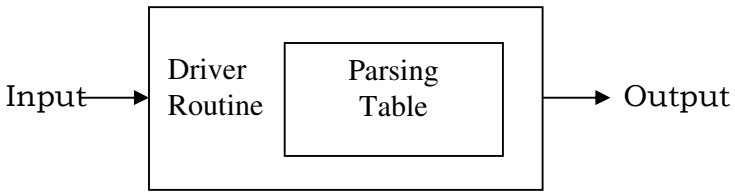
LR Parsers

The bottom – up parsers are called as ***LR parsers*** because they scan the input from left to right and construct a rightmost derivation in reverse order.

Logically an LR Parser consists of two parts, a driver routine and a parsing table. The driver routine is same for all LR parsers only the parsing table changes from one parser to another.

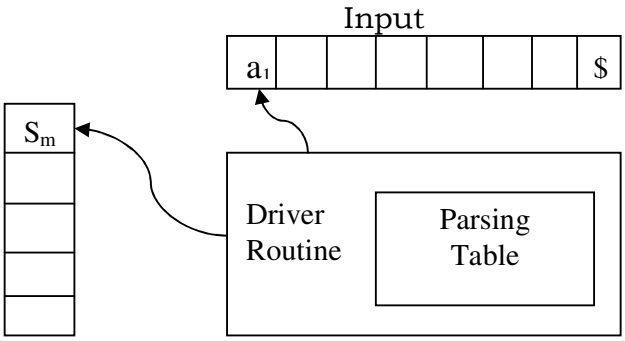


(a) Generating (Parsing Table) Parser



(b) Operation of the Parser

The parser has an input, a stack, and a parsing table as shown below:



LR Parser

There are three types of LR parsers:-

1) SLR Parser (Simple LR Parser)
2) CLR Parser (Canonical LR Parser)
3) LALR Parser (LookAhead LR Parser)

The program driving the LR parser behaves as follows:

It determines S_m, the state currently on top of the stack, and a_i, the current input symbol.

It then consults ACTION[S_m, a_i], the parsing action table entry for state S_m and input a_i. The entry ACTION[S_m, a_i] can have one of the following four values:-

1. Shift S
2. Reduce A →B
3. Accept
4. Error

1.1 Why LR Parsers?

LR parsing is attractive for a variety of reasons:

- LR parsers can be constructed to recognize virtually all programming language constructs for which context-free grammars can be written. Non - LR context-free grammars exist, but these can generally be avoided for typical programming-language constructs.
- The LR-parsing method is the most general nonbacktracking shift-reduce parsing method known, yet it can be implemented as efficiently as other, more primitive shift-reduce methods (see the bibliographic notes).
- An LR parser can detect a syntactic error as soon as it is possible to do so on a left-to-right scan of the input.
- The class of grammars that can be parsed using LR methods is a proper superset of the class of grammars that can be parsed with predictive or LL methods.

The principal drawback of the LR method is that it is too much work to construct an LR parser by hand for a typical programming-language grammar. A specialized tool, an LR parser generator, is needed.

1.2 Items and the LR(0) Automations

An LR(0) item (or ITEM for short) of a grammar G is defined to be a production of G with a dot at some position in the right side of the production. Thus a production A →XYZ generates the four items:-

A →.XYZ
A →X.YZ
A →XY.Z
A →XYZ.

The production A →Є generates only one item, "A →.". Inside the computer, items are easily represented by pairs of integers, the first giving the number of the production and the second the position of the dot.

Intuitively, an item indicates how much of a production we have seen at a given point in the parsing process.

A collection of sets of items, called the ***Canonical LR(0)*** collection, provides the basis for constructing a class of LR parsers called ***simple LR (SLR)*** parsers.

To construct the canonical LR(0) collection for a grammar, we define an augmented grammar and two functions, CLOSURE and GOTO. If G is a grammar with start symbol S, then G', the *augmented grammar* for G, is G with a new start symbol S' and production S' → S. The purpose of this new starting production is to indicate to the parser when it should stop parsing and announce acceptance of the input. That is, acceptance occurs when and only when the parser is about to reduce by S' → S.

Closure of Item Sets

If I is a set of items for a grammar G, then the set of items CLOSURE(I) is constructed from I by the rules:-

1. Every item in I is in CLOSURE(I).
2. If A →α.Bβ is in CLOSURE(I) and B →γ is a production, then add the item B →.γ to I, if it is not already there.

GOTO

The second useful function is GOTO(I,X) where I is a set of items and X is a grammar symbol.

GOTO(I,X) is defined to be the closure of the set of all items [A →αX.β] such that [A →αX.β] is in I. Intuitively, if I is the set of items that are valid for some workable prefix γ, then GOTO(I,X) is the set of items that are valid for the viable prefix γX.

Set of Items Construction

```
procedure ITEMS(G')
begin
        C := { CLOSURE ( {S' →.S } ) };
        repeat
                for each set of items I in C and each grammar symbol X such
                that GOTO (I,X) is not empty and is not in C
                        do add GOTO (I,X) to C
        until no more sets of items can be added to C
end
```

1.2 SLR Parsers

The SLR method for constructing parsing tables is a good starting point for studying LR parsing.

The SLR method begins with LR(0) items and LR(0) automata. That is, given a grammar, G, G is augmented to produce G', with a new start symbol S'. From G', we construct C, the canonical collection of sets of items for G' together with the GOT0 function.

1.1 Algorithm to construct SLR Parser:

INPUT: C, the canonical collection of sets of items for an augmented grammar G'.

OUTPUT: If possible, an LR parsing table consisting of a parsing action function ACTION and a GOTO function.

METHOD:

1. Procedure CLOSURE (I):

```
begin
        repeat
                for each item A →α.Bβ in I and each production B →γ in
                grammar G such that B →.γ is not in I
                        do add B →.γ to I;
        until no more items can be added to I
        return I;
end
```

2. GOTO:

The function GOTO (I,X) where I is a set of items and X is a grammar symbol.

GOTO(I,X) is defined to be the closure of the set of all items [A →αX.β] such that [A →αX.β] is in I.

3. Procedure ITEMS (G'):

```
begin
        C := { CLOSURE ( { S' →S } ) };
        repeat
                for each set of items I in C and each grammar symbol X
                such that GOTO (I,X) is not empty and is not in C
                        do add GOTO (I,X) to C
        until no more sets of items can be added to C
end
```