COMPILER CONSTRUCTION

Principles and Practice

Kenneth C. Louden

5. Bottom-Up Parsing

PART ONE

Contents

5.1 Overview of Bottom-Up Parsing

- **A bottom-up parser uses an explicit stack to perform a parse**
  - **The parsing stack contains tokens, nonterminals as well as some extra state information**
  - **The stack is empty at the beginning of a bottom-up parse, and will contain the start symbol at the end of a successful parse**
- **A schematic for bottom-up parsing:**

  $ InputString $

  …….

  $ StartSymbol $

  accept

  - **Where the parsing stack is on the left,**
  - **The input is in the center, and**
  - **The actions of the parser are on the right.**

- **A bottom-up parser has two possible actions (besides "accept"):**

  Shift **a terminal from the front of the input to the top of the stack**

  Reduce **a string α at the top of the stack to a nonterminal A, given the BNF choice A→α**

- **A bottom-up parser is thus sometimes called a shift-reduce parser**
- **One further feature of bottom-up parsers is that, grammars are always augmented with a new start symbol**

$$S' \rightarrow S$$

- **Example 5. 1 The augmented grammar for balanced parentheses:**

  $S' \rightarrow S$

  $S \rightarrow (S)S|\varepsilon$

- **A bottom-up parser of the string ( ) using this grammar is given in following table**

	Parsing stack	Input	Action
1	$	( )$	shift
2	$(	)$	reduce S→ε
3	$(S	)$	shift
4	$(S)	$	reduce S→ε
5	$(S)S	$	reduce S → (S)S
6	$S	$	reduce S' → S
7	$S’	$	accept

- **Example. 5.2 The augmented grammar for rudimentary arithmetic expressions:**

  E'→E

  E→E + n | n

- **A bottom-up parse of the string n + n using this grammar is given in following table.**

	Parsing stack	input	Action
1	$	n+n$	shift
2	$n	+n$	reduce E→n
3	$E	+n$	shift
4	$E+	n$	shift
5	$E+n	$	reduce E→E + n
6	$E	$	reduce E'→E
7	$E'	$	accept

- **The handle of the right sentential form:**
  - **A string, together with**
  - **The position in the right sentential form where it occurs, and**
  - **The production used to reduce it.**
- **Determining the next handle is the main task of a shift-reduce parser.**

	Parsing stack	input	Action
1	$	n+n$	shift
2	$n	+n$	reduce E→n
3	$E	+n$	shift
4	$E+	n$	shift
5	$E+n	$	reduce E→E + n
6	$E	$	reduce E'→E
7	$E'	$	accept

- **Note:**
  - **The string of a handle forms a complete right-hand side of one production; The rightmost position of the handle string is at the top of the stack;**
  - **To be the handle, it is not enough for the string at the top of the stack to match the right-hand side of a production.**
    - **Indeed, if an ε-production is available for reduction, as in Example 5.1 , then its right-hand side (the empty string) is always at the top of the stack.**
    - **Reductions occur only when the resulting string is indeed a right sentential form.**
  - **For example, in step 3 of Table 5.1 a reduction by $S \rightarrow \varepsilon$ could be performed, but the resulting string (S S) is not a right sentential form, and thus ε is not the handle at this position in the sentential form (S) .**

5.2 FINITE AUTOMATA OF LR(0) ITEMS AND LR(0) PARSING

5.2.1 LR(0) Items

- **An LR(0) item of a context-free grammar:**
  - **A production choice with a distinguished position in its right-hand side**
  - **Indicating the distinguished position by a period**
- **Example:**
  - **if $A \rightarrow \alpha$ is a production choice, and if β and γ are any two strings of symbols (including the empty string ε) such that $\beta\gamma = \alpha$**
  - **then $A \rightarrow \beta \cdot \gamma$ is an LR(0) item.**
- **These are called LR(0) items because they contain no explicit reference to lookahead**

Example 5.3 The grammar of Example 5. l :

S' → S
S → (S)S|ε

This grammar has three production choices and eight items:
S' → ·S
S' → S·
S → ·(S)S
S → (·S)S
S → (S·)S
S → (S)·S
S → (S)S·
S → ·

Example 5.4 The grammar of Example 5.2

There are the following eight items:

E’→·E

E’→E·

E→·E + n

E→E· + n

E→E +· n

E→E + n·

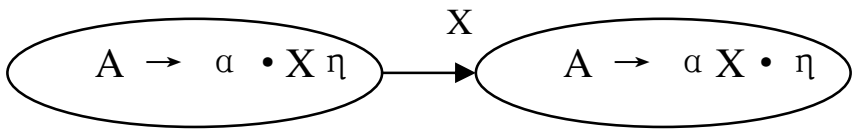
E→·n

E→ n·

5.2.2 Finite Automata of Items

- **The LR(0) items can be used as the states of a finite automaton**
  - **that maintains information about the parsing stack and the progress of shift-reduce parse.**
  - **This will start out as a nondeterministic finite automaton.**
- **From the NFA of LR(0) items to construct the DFA of sets of LR(0) items using the subset construction of Chapter 2**
- ***The transitions of the NFA of LR(0) items?***
  - **Consider the item $A \rightarrow \alpha \cdot \gamma$, and**
  - **Suppose γ begins with the symbol X, be either a token or a nonterminal,**
  - **So that the item can be written as $A \rightarrow \alpha \cdot X\eta$.**

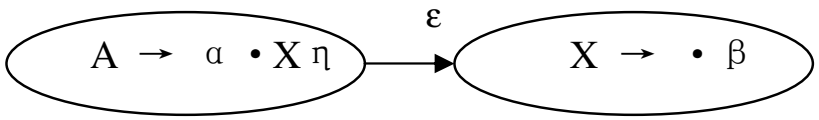
- **A transition on the symbol X from the state represented by this item to the state represented by the item $A \rightarrow \alpha X \cdot \eta$.**
- **In graphical form we write this as**



(1) If X is a token, then this transition corresponds to a shift of X from the input to the top of the stack during a parse.

(2) If X is a nonterminal, then the interpretation of this transition is more complex, since X will never appear as an input symbol.

- **In fact, such a transition will still correspond to the pushing of X onto the stack during a parse;**
- **This can only occur during a reduction by a production $X \rightarrow \beta$.**
  - **Such a reduction must be preceded by *the recognition of* β, and the state given by the initial item $X \rightarrow \cdot\beta$ represents the beginning of this process**
  - **(the dot indicating that we are about to recognize β,**
  - **then for every item $A \rightarrow \alpha \cdot X\eta$ we must add an *ε*-transition)**

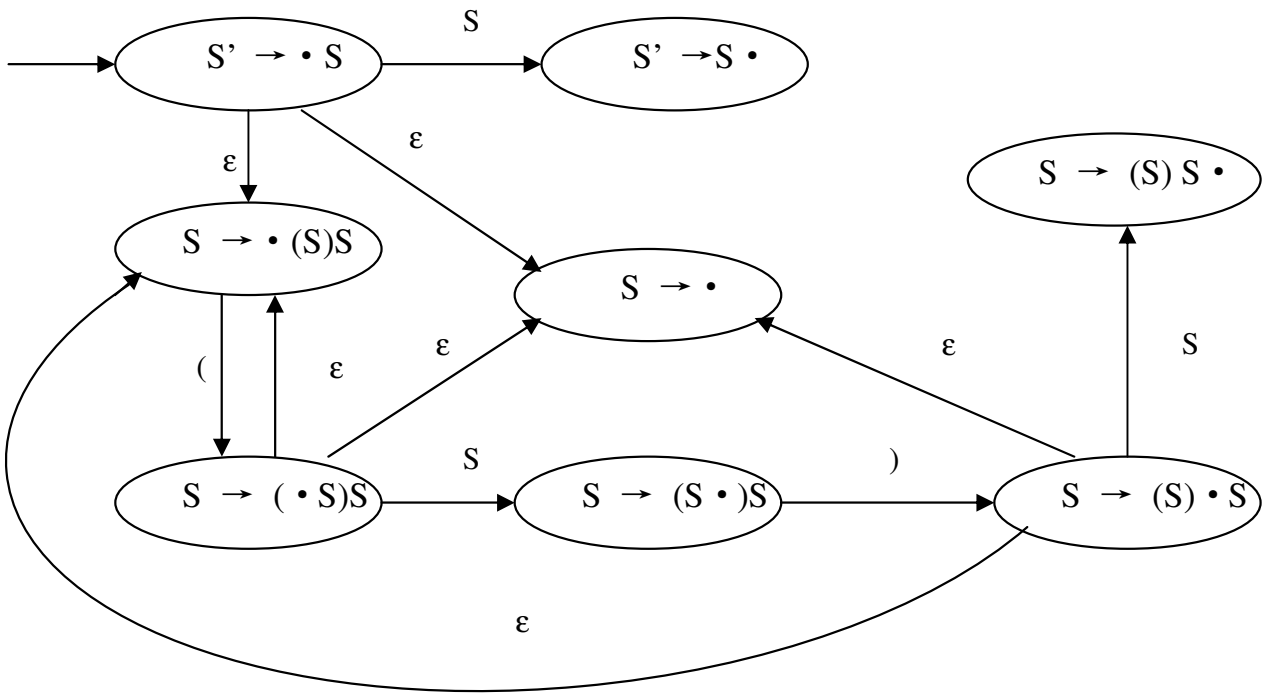


for every production choice $X \rightarrow \beta$ of X, indicating that X can be produced by recognizing any of the right-hand sides of its production choices.

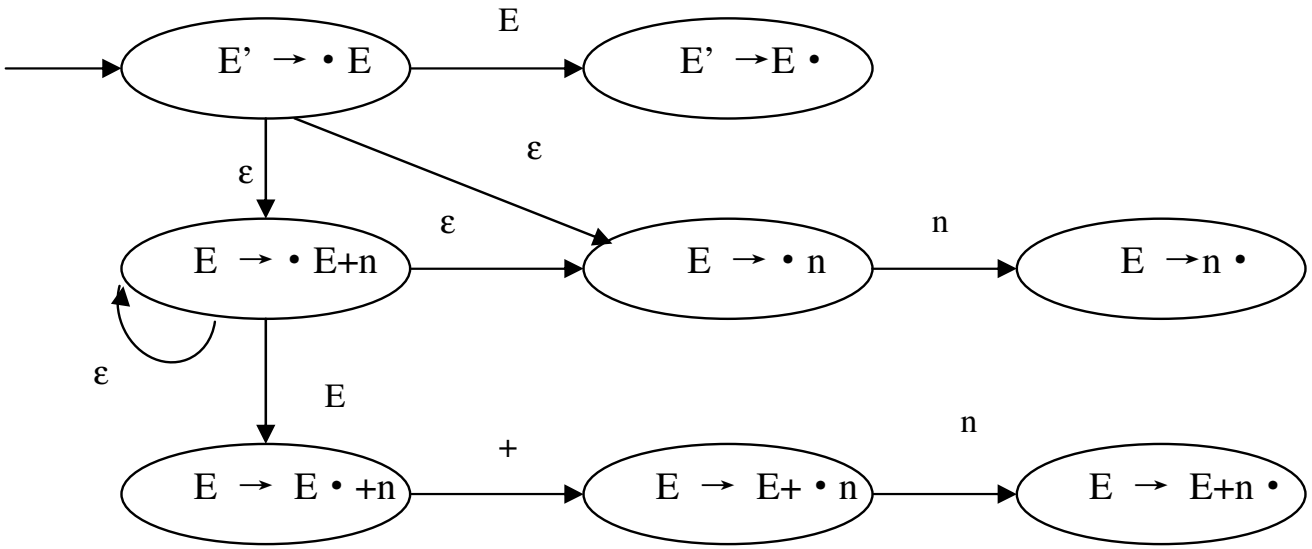
- **The start state of the NFA should correspond to the initial state of the parser:**
  - **The stack is empty, and**
  - **Be about to recognize an S, where S is the start symbol of the grammar.**
  - **Any initial item $S \rightarrow \cdot\alpha$ constructed from a production choice for S could serve as a start state.**
  - **Unfortunately, there may be many such production choices for S.**
- **The solution is to augment the grammar by a single production $S' \rightarrow S$**
  - **where S' is a new non-terminal**

- **The solution is to augment the grammar by a single production S'→S, where S' is a new nonterminal.**
  - **S' then becomes the start state of the augmented grammar, and the initial item S' →· S becomes the start state of the NFA.**
  - **This is the reason we have augmented the grammars of the previous examples.**
- **The NFA will have no accepting states at all:**
  - **The purpose of the NFA is to keep track of the state of a parse, not to recognize strings;**
  - **The parser itself will decide when to accept, and the NFA need not contain that information.**

- **Example 5.5 The NFA with the eight LR(0) items of the grammar of Example 5. l in Figure 5.1**
- **Note : Every item in the figure with a dot before the non-terminal S has an *ε*-transition to every initial item of S.**

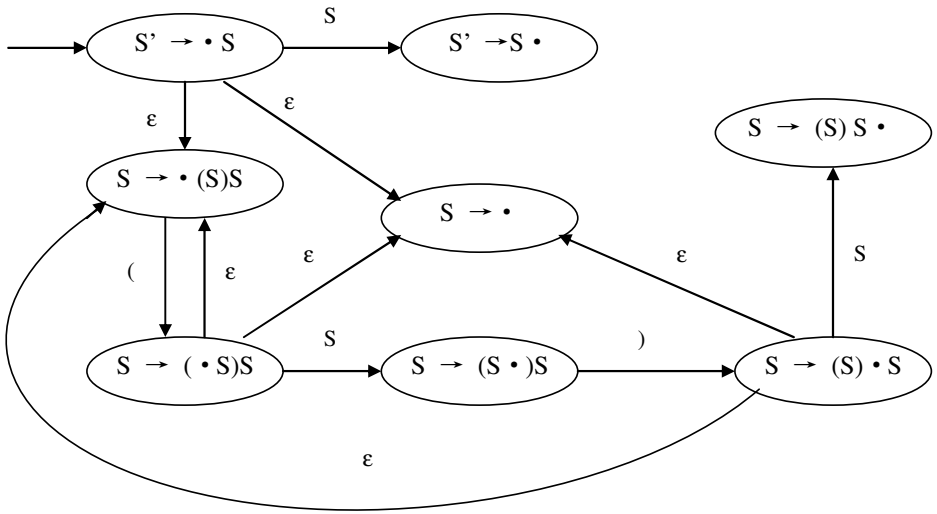


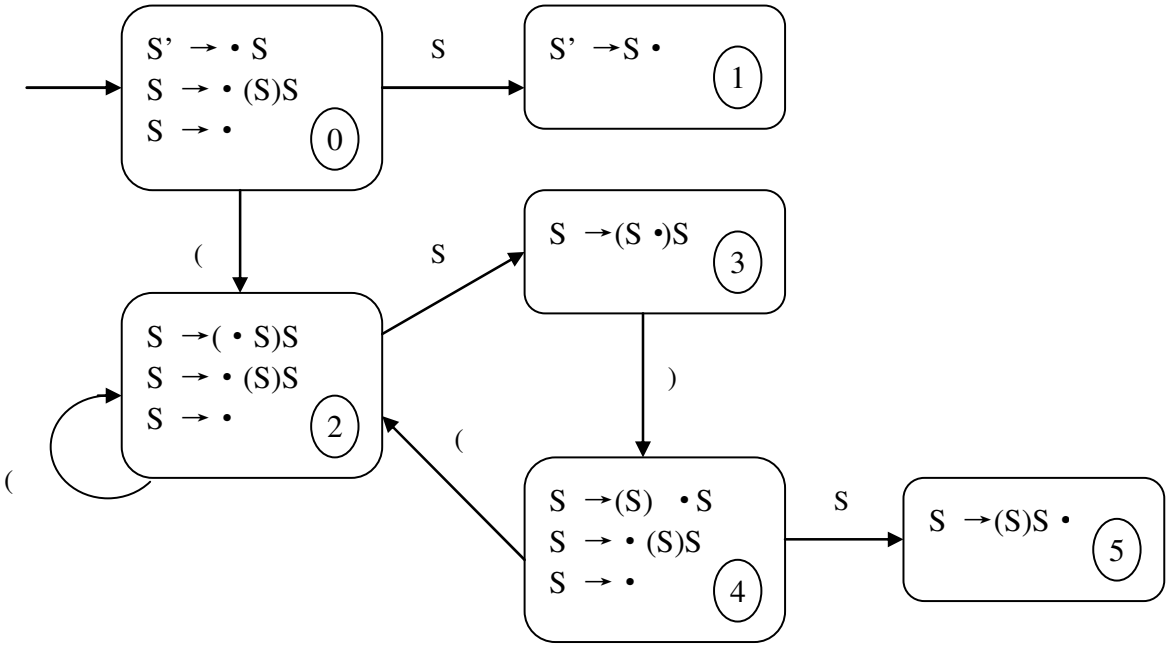
- **Example 5.6 The NFA of the LR(0) items associated with the grammar of Example 5.2. in Figure 5.2.**
- **Note: The initial item E→ . E + n has an ε-transition to itself**
  - **(This situation will occur in all grammars with immediate left recursion.)**



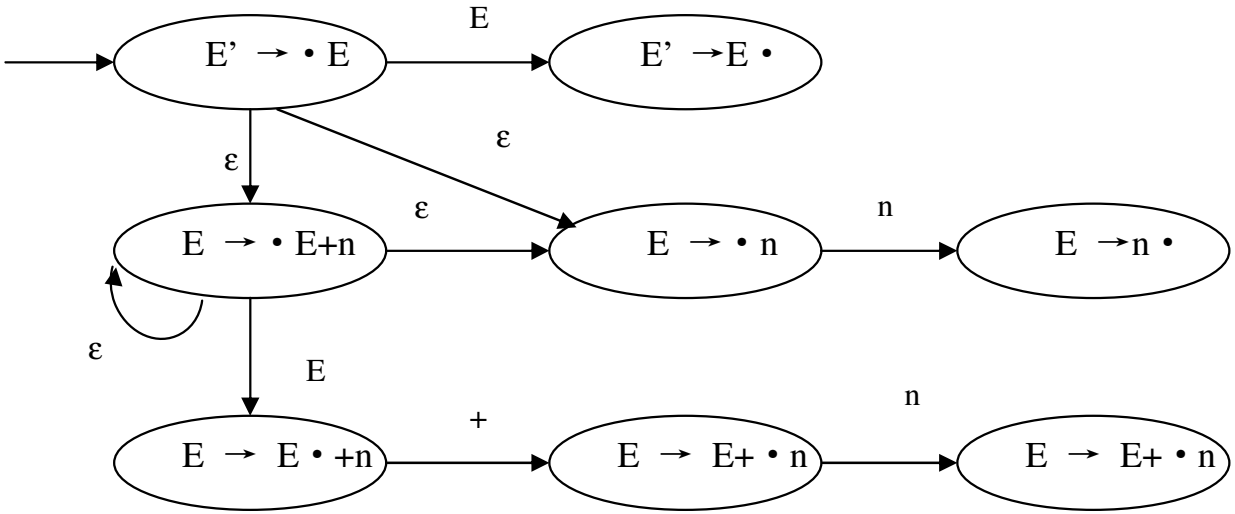
- In order to describe the use of items to keep track of the parsing state, must construct the DFA of sets of items corresponding to the NFA of items according to the subset construction.
- Perform the subset construction for the two examples of NFAs that have been just given.

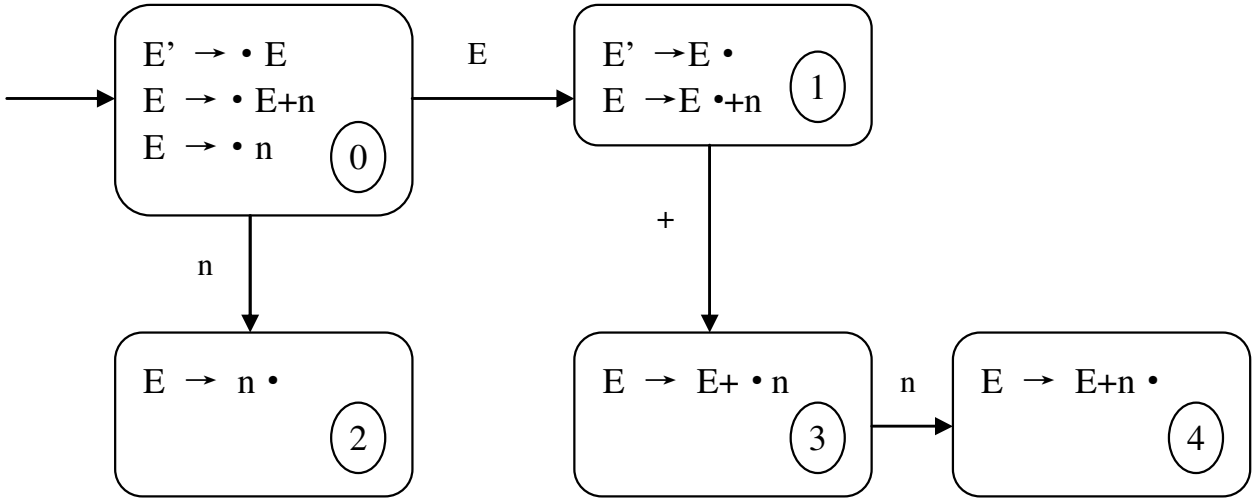
Example 5.7 Consider the NFA of Figure 5.l.





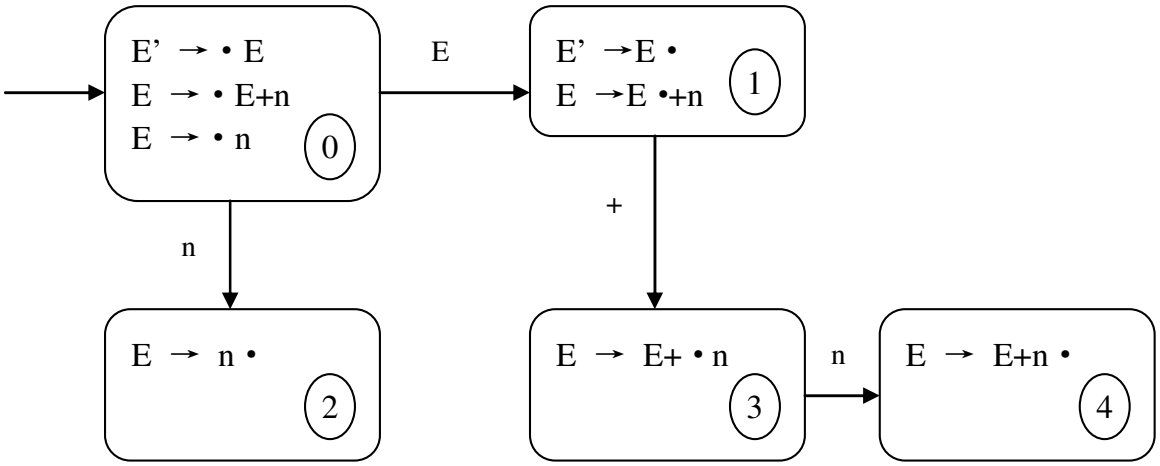
Example 5.8 **Consider the NFA of Figure 5.2**





5.2.3The LR(0) Parsing Algorithm

- The algorithm depends on keeping track of the current state in the DFA of sets of items;
- Modify the parsing stack to store not only symbols but also state numbers; Pushing the new state number onto the parsing stack after each push of a symbol.



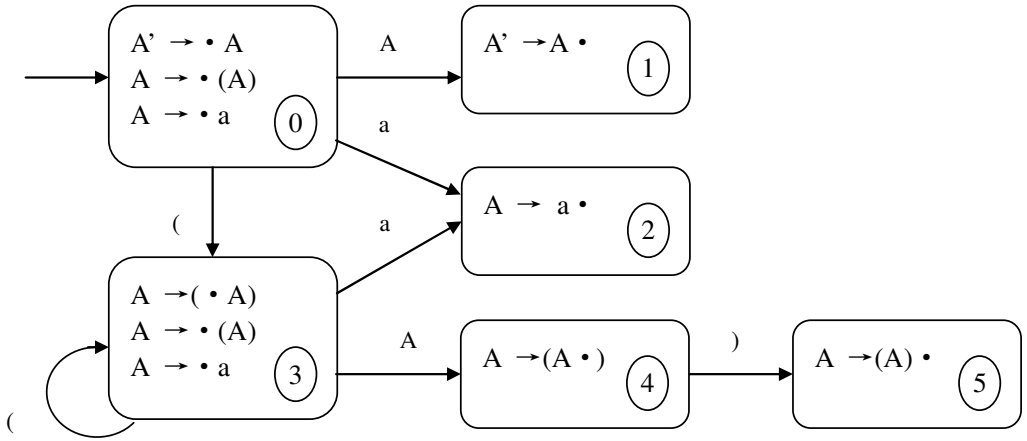
$0	Input String
$0 n 2	Rest input string

Definition: The LR (0) parsing algorithm

- Let *s* be the current state (at the top of the parsing stack). Then actions are defined as follows:
- 1. If state s contains any item of the form $A \rightarrow \alpha \cdot X\beta$, where X is a terminal. Then the action is to shift the current input token on to the stack.
  - If this token is X. and state s contains item $A \rightarrow \alpha \cdot X\beta$, then the new state to be pushed on the stack is the state containing the item $A \rightarrow \alpha X \cdot \beta$.
  - If this token is not X for some item in state s of the form just described, an error is declared.

- **2. If state s contains any complete item (an item of the form A → γ·), then the action is to reduce by the rule A → γ·**
  - **A reduction by the rule S` → S, where S is the start state, is equivalent to acceptance, provided the input is empty, and error if the input is not empty**
  - **In all other cases, for new state is computed as follows:**
    - **Remove the string γ and all of its corresponding states from the parsing stack**
    - **Correspondingly, back up in the DFA to the state from which the construction of γ began**
    - **Again, by the construction of the DFA, this state must contain an item of the form B → α·Aβ**
  - **Push A onto the stack, and push (as the new state) the state containing the item B → αA·β.**

Example 5.9 Consider the grammar: A → (A) | a



	Parsing stack	input	Action
1	$ 0	((a))$	shift
2	$ 0 (3	(a))$	shift
3	$ 0 (3 (3	a)$	shift
4	$ 0 (3 (3 a 2	))$	reduce A→a
5	$ 0 (3 (3 A 4	))$	shift
6	$ 0 (3 (3 A 4) 5	)$	reduce A→(A)
7	$ 0 (3 A 4	)$	shift
8	$ 0 (3 A 4) 5	$	reduce A→(A)
9	$ 0 A 1	$	accept

The parsing table

- **The DFA of sets of items and the actions specified by the LR(0) parsing algorithm can be combined into a parsing table.**
- **Then, the LR(0) parsing becomes a table-driven parsing method.**

An example of such a parsing table

State	Action	Rule	Input			Goto
			(	a	)	A
0	shift		3	2		1
1	reduce	A'→A				
2	reduce	A→a				
3	shift		3	2		4
4	shift				5	
5	reduce	A→(A)				

1. One column is reserved to indicate the *actions* for each state;
2. A further column is used to indicate the grammar choice for the reduction;
3. For each token, there is a column to represent the new state;
4. Transitions on non-terminals are listed in the Goto sections.

End of Part One

THANKS