Constructing Action and Goto Tables

MICHAEL WOLLOWSKI

Mostly based on ”Engineering a Compiler”

Constructing Action and Goto Tables

The compiler writer can build Action and *Goto* tables by hand.

However, the table-construction algorithm requires scrupulous bookkeeping.

It is a prime example of the kind of task that should be automated and relegated to a computer.

In order to understand the behavior of those programs, we will study one algorithm that can be used to construct LR(1) parse tables.

LR(1) Items

Represent potential handles and look-ahead symbols

An LR(1) item $[A \rightarrow \beta \bullet \gamma, a]$ consists of:

- A production $A \rightarrow \beta\gamma$
- A placeholder • that indicates the position of the stacktop in the productions rhs
- A specific terminal symbol a as a lookahead symbol.

LR(1) Items

The position of placeholder • distinguishes among the following three cases:

- $[A \rightarrow \bullet \beta\gamma, a]$ indicates that an A would be valid and that recognizing a β next would be one step toward discovering an A. We call such an item a *possibility*, because it represents a possible completion for the input already seen.
- $[A \rightarrow \beta \bullet \gamma, a]$ indicates that the parser has progressed from the state $[A \rightarrow \bullet \beta\gamma, a]$ by recognizing β. The β is consistent with recognizing an A. One valid next step would be to recognize a γ. We call such an item *partially complete*.
- $[A \rightarrow \beta\gamma \bullet, a]$ indicates that the parser has found $\beta\gamma$ in a context where an A followed by an a would be valid. If the look ahead symbol is a, then the item is a handle and the parser can reduce $\beta\gamma$ to A. Such an item is *complete*.

LR(1) Items for Parenthesis Grammar

Here you see the complete set of LR(1) items generated for the parentheses grammar listed below.

1. $Goal \rightarrow List$
2. $List \rightarrow List\ Pair$
3. $\quad\quad | \ Pair$
4. $Pair \rightarrow \underline{(}\ Pair\ \underline{)}$
5. $\quad\quad | \ \underline{(}\ \underline{)}$

$[Goal \rightarrow \bullet\ List, \texttt{eof}]$

$[Goal \rightarrow List\ \bullet, \texttt{eof}]$

$[List \rightarrow \bullet\ List\ Pair, \texttt{eof}]$	$[List \rightarrow \bullet\ List\ Pair, \underline{(}]$	
$[List \rightarrow List \bullet Pair, \texttt{eof}]$	$[List \rightarrow List \bullet Pair, \underline{(}]$	
$[List \rightarrow List\ Pair\ \bullet, \texttt{eof}]$	$[List \rightarrow List\ Pair\ \bullet, \underline{(}]$	
$[List \rightarrow \bullet\ Pair, \texttt{eof}]$	$[List \rightarrow \bullet\ Pair, \underline{(}]$	
$[List \rightarrow Pair\ \bullet, \texttt{eof}]$	$[List \rightarrow Pair\ \bullet, \underline{(}]$	
$[Pair \rightarrow \bullet\ \underline{(}\ Pair\ \underline{)}, \texttt{eof}]$	$[Pair \rightarrow \bullet\ \underline{(}\ Pair\ \underline{)}, \underline{)}]$	$[Pair \rightarrow \bullet\ \underline{(}\ Pair\ \underline{)}, \underline{(}]$
$[Pair \rightarrow \underline{(} \bullet Pair\ \underline{)}, \texttt{eof}]$	$[Pair \rightarrow \underline{(} \bullet Pair\ \underline{)}, \underline{)}]$	$[Pair \rightarrow \underline{(} \bullet Pair\ \underline{)}, \underline{(}]$
$[Pair \rightarrow \underline{(}\ Pair \bullet \underline{)}, \texttt{eof}]$	$[Pair \rightarrow \underline{(}\ Pair \bullet \underline{)}, \underline{)}]$	$[Pair \rightarrow \underline{(}\ Pair \bullet \underline{)}, \underline{(}]$
$[Pair \rightarrow \underline{(}\ Pair\ \underline{)} \bullet, \texttt{eof}]$	$[Pair \rightarrow \underline{(}\ Pair\ \underline{)} \bullet, \underline{)}]$	$[Pair \rightarrow \underline{(}\ Pair\ \underline{)} \bullet, \underline{(}]$
$[Pair \rightarrow \bullet\ \underline{(}\ \underline{)}, \texttt{eof}]$	$[Pair \rightarrow \bullet\ \underline{(}\ \underline{)}, \underline{(}]$	$[Pair \rightarrow \bullet\ \underline{(}\ \underline{)}, \underline{)}]$
$[Pair \rightarrow \underline{(} \bullet \underline{)}, \texttt{eof}]$	$[Pair \rightarrow \underline{(} \bullet \underline{)}, \underline{(}]$	$[Pair \rightarrow \underline{(} \bullet \underline{)}, \underline{)}]$
$[Pair \rightarrow \underline{(}\ \underline{)} \bullet, \texttt{eof}]$	$[Pair \rightarrow \underline{(}\ \underline{)} \bullet, \underline{(}]$	$[Pair \rightarrow \underline{(}\ \underline{)} \bullet, \underline{)}]$

First and Follow Sets for our Grammar

Follow(*Goal*) = {**eof**}

Follow(*List*) = {**eof, (**}

Follow(*Pair*) = {**eof, (,)**}

First(*Goal*) = { **(**}

First(*List*) = { **(**}

First(*Pair*) = { **(**}

1. $Goal \rightarrow List$
2. $List \rightarrow List\ Pair$
3. $\quad\quad | \ Pair$
4. $Pair \rightarrow \underline{(}\ Pair\ \underline{)}$
5. $\quad\quad | \ \underline{(}\ \underline{)}$

Canonical Collection

A *Canonical Collection* $\mathcal{CC}$ of a set of LR(1) items is a model of all transitions that can occur, beginning at the start state.

$\mathcal{CC} = \{CC_0, CC_1, \ldots, CC_n\}$

Each CC_i

- is a set of LR(1) items
- will represents a parser state

Two operations are used to calculate them:

- Closure
- Goto

Closure

The closure operation completes a state.

Given a core set of LR(1) items, it adds to that set any related LR(1) items that they imply.

For example, any set that contains *Goal* → *List* may also contain the productions that derive a *List*.

Thus, we may add items [*List* → •*List Pair,* **`eof`**] and [*List* → • *Pair,* **`eof`**] to the list containing

[*Goal* → •*List,* **`eof`**].

Closure

To simplify the task of finding the goal symbol, we require that the grammar have a unique goal symbol that does not appear on the right-hand side of any production.

The item [*Goal* → •*List,* **`eof`**] represents the parser's initial state for the parentheses grammar.

Every valid parse recognizes *Goal* followed by **`eof`**.

This item forms the core of the first state in $\mathcal{CC}$, labelled cc_0.

If the grammar has multiple productions for the goal symbol, each of them generates an item in the initial core of cc_0.

The Closure Procedure

Finds equivalence class of LR(1) items

- s : a set of LR(1) items

```
closure(s)
  while (s is changing)
    for each item [A→β•Cδ, a] in s
      for each production C → γ
        for each b in First(δa)
          s ← s ∪ {[C →•γ,b]}
  return s
```

The Closure Procedure: Example

For the parentheses grammar, the initial item is

[*Goal* → •*List*, **eof**]

Applying *closure* to that set adds the following items:

1. [*List*→ •*List Pair*, **eof**]
2. [*List*→ •*List Pair*, **(**]
3. [*List*→ •*Pair*, **eof**]
4. [*List*→ •*Pair*, **(**]
5. [*Pair*→ • **(***Pair***)**, **eof**]
6. [*Pair*→ • **(***Pair***)**, **(**]
7. [*Pair*→ • **()**, **eof**]
8. [*Pair*→ • **()**, **(**]

1	*Goal* → *List*
2	*List* → *List Pair*
3	| *Pair*
4	*Pair* → (*Pair*)
5	| ( )

This set is cc_0

Goto

To model the transition that the parser would make from a given state on some grammar symbol, *x,* the algorithm computes the set of items that would result from recognizing an *x.*

To do so, the algorithm selects the subset of the current set of LR(1) items where • precedes *x* and advances the • past the *x* in each of them.

Q2

The Procedure Goto

Finds state transitions

- s : a set of LR(1) items
- x : a terminal or non-terminal symbol

```
goto(s, x)
  moved ← {}
  for each item i in s
    if i is like [A→β•xδ, a] then
      moved ← moved ∪ { [A→βx•δ, a] }
  return closure(moved)
```

The Procedure Goto: Example

Given cc_0, we now compute goto(cc_0, `(`)

This set includes the following items:

1. [*Pair*→ `(`•*Pair*`)`, `eof`]
2. [*Pair*→ `(`•*Pair*`)`, `(`]
3. [*Pair*→ `(`•`)`, `eof`]
4. [*Pair*→ `(`•`)`, `(`]
5. [*Pair*→ • `(`*Pair*`)`, `)`]
6. [*Pair*→ • `()`, `)`]

Algorithm to Build $\mathcal{CC}$

$CC_0 \leftarrow$ closure({[S′ → •S, **eof**]})
$\mathcal{CC} \leftarrow$ { CC_0 }
while (new sets still being added to $\mathcal{CC}$)
  for each unmarked set CC_j in $\mathcal{CC}$
    mark CC_j as processed
    for each X following a • in an item of CC_j
      temp ← goto(CC_j, X)
      if (temp not in $\mathcal{CC}$)
        then $\mathcal{CC} \leftarrow \mathcal{CC} \cup$ {temp}
      record transition from CC_j to temp on X

EITHER A TERMINAL OR A NON-TERMINAL

CC for Parentheses Grammar

$$CC_0 = \left\{ \begin{array}{lll} [Goal \rightarrow \bullet\, List, \texttt{eof}] & [List \rightarrow \bullet\, List\ Pair, \texttt{eof}] & [List \rightarrow \bullet\, List\ Pair, \underline{(}] \\ [List \rightarrow \bullet\, Pair, \texttt{eof}] & [List \rightarrow \bullet\, Pair, \underline{(}] & [Pair \rightarrow \bullet\, \underline{(}\ Pair\ \underline{)}, \texttt{eof}] \\ [Pair \rightarrow \bullet\, \underline{(}\ Pair\ \underline{)}, \underline{(}] & [Pair \rightarrow \bullet\, \underline{(}\ \underline{)}, \texttt{eof}] & [Pair \rightarrow \bullet\, \underline{(}\ \underline{)}, \underline{(}] \end{array} \right\}$$

goto(CC_0, () is CC_3

$$CC_3 = \left\{ \begin{array}{lll} [Pair \rightarrow \bullet\, \underline{(}\ Pair\ \underline{)}, \underline{)}] & [Pair \rightarrow \underline{(}\ \bullet\, Pair\ \underline{)}, \texttt{eof}] & [Pair \rightarrow \underline{(}\ \bullet\, Pair\ \underline{)}, \underline{(}] \\ [Pair \rightarrow \bullet\, \underline{(}\ \underline{)}, \underline{)}] & [Pair \rightarrow \underline{(}\ \bullet\, \underline{)}, \texttt{eof}] & [Pair \rightarrow \underline{(}\ \bullet\, \underline{)}, \underline{(}] \end{array} \right\}$$

goto(CC_0, *List*) is CC_1.

$$CC_1 = \left\{ \begin{array}{lll} [Goal \rightarrow List \bullet, \texttt{eof}] & [List \rightarrow List \bullet\, Pair, \texttt{eof}] & [List \rightarrow List \bullet\, Pair, \underline{(}] \\ [Pair \rightarrow \bullet\, \underline{(}\ Pair\ \underline{)}, \texttt{eof}] & [Pair \rightarrow \bullet\, \underline{(}\ Pair\ \underline{)}, \underline{(}] & [Pair \rightarrow \bullet\, \underline{(}\ \underline{)}, \texttt{eof}] \\ & [Pair \rightarrow \bullet\, \underline{(}\ \underline{)}, \underline{(}] & \end{array} \right\}$$

CC for Parentheses Grammar

goto(CC_0, *Pair*) is CC_2.

$$CC_2 = \{[List \rightarrow Pair \bullet, \texttt{eof}] \quad [List \rightarrow Pair \bullet, \underline{(}]\}$$

goto(CC_1, *Pair*) is CC_4.

$$CC_4 = \{[List \rightarrow List\ Pair \bullet, \texttt{eof}] \quad [List \rightarrow List\ Pair \bullet, \underline{(}]\}$$

goto(CC_1,$\underline{(}$) is CC_3, which represents the future need to find a matching $\underline{)}$

goto(CC_3, *Pair*) is CC_5.

$$CC_5 = \{[Pair \rightarrow \underline{(}\ Pair \bullet \underline{)}, \texttt{eof}] \quad [Pair \rightarrow \underline{(}\ Pair \bullet \underline{)}, \underline{(}]\}$$

CC for Parentheses Grammar

goto(CC_3,$\underline{(}$) is CC_6.

$$CC_6 = \left\{\begin{array}{ll}[Pair \rightarrow \bullet\ \underline{(}\ Pair\ \underline{)}, \underline{)}] & [Pair \rightarrow \underline{(} \bullet Pair\ \underline{)}, \underline{)}] \\ [Pair \rightarrow \bullet\ \underline{(}\ \underline{)}, \underline{)}] & [Pair \rightarrow \underline{(} \bullet\ \underline{)}, \underline{)}]\end{array}\right\}$$

goto(CC_3,$\underline{)}$) is CC_7.

$$CC_7 = \{[Pair \rightarrow \underline{(}\ \underline{)} \bullet, \texttt{eof}] \quad [Pair \rightarrow \underline{(}\ \underline{)} \bullet, \underline{(}]\}$$

goto(CC_5,$\underline{)}$) is CC_8.

$$CC_8 = \{[Pair \rightarrow \underline{(}\ Pair\ \underline{)} \bullet, \texttt{eof}] \quad [Pair \rightarrow \underline{(}\ Pair\ \underline{)} \bullet, \underline{(}]\}$$

CC for Parentheses Grammar

goto(CC_5, ( ) is CC_8.

$$CC_8 = \{[Pair \rightarrow \underline{(}\ Pair\ \underline{)}\ \bullet, \texttt{eof}] \quad [Pair \rightarrow \underline{(}\ Pair\ \underline{)}\ \bullet, \underline{(}]\}$$

goto(CC_6, *Pair*) is CC_9.

$$CC_9 = \{[Pair \rightarrow \underline{(}\ Pair \bullet\ \underline{)}, \underline{)}]\}$$

goto(CC_6,)) is CC_{10}.

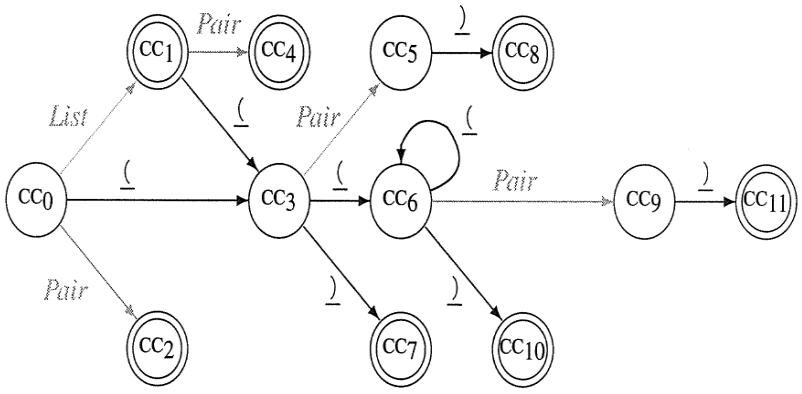
$$CC_{10} = \{[Pair \rightarrow \underline{(}\ \underline{)}\ \bullet, \underline{)}]\}$$

CC for Parentheses Grammar

The closure sets produced for our grammar:

Iteration	Item	*Goal*	*List*	*Pair*	(	)	eof
0	CC_0	Ø	CC_1	CC_2	CC_3	Ø	Ø
1	CC_1	Ø	Ø	CC_4	CC_3	Ø	Ø
	CC_2	Ø	Ø	Ø	Ø	Ø	Ø
	CC_3	Ø	Ø	CC_5	CC_6	CC_7	Ø
2	CC_4	Ø	Ø	Ø	Ø	Ø	Ø
	CC_5	Ø	Ø	Ø	Ø	CC_8	Ø
	CC_6	Ø	Ø	CC_9	CC_6	CC_{10}	Ø
	CC_7	Ø	Ø	Ø	Ø	Ø	Ø
3	CC_8	Ø	Ø	Ø	Ø	Ø	Ø
	CC_9	Ø	Ø	Ø	Ø	CC_{11}	Ø
	CC_{10}	Ø	Ø	Ø	Ø	Ø	Ø
4	CC_{11}	Ø	Ø	Ø	Ø	Ø	Ø

DFA of cc_is



Producing the Action and Goto Tables

Each cc_i becomes a state.

Shift:

- An item of the form $[A \rightarrow \beta \bullet C\gamma, a]$ indicates that encountering the terminal symbol C would be a valid next step toward discovering the nonterminal *A.* Either β or γ can be ε.
- It generates a *shift* item on C in the current state.
- The next state for the recognizer is the state generated by computing goto on the current state with the terminal c.

Reduce:

- An item of the form $[A \rightarrow \beta \bullet, a]$ indicates that the parser has recognized a β and if the lookahead is a, then the item is a handle.
- It generates a *reduce* item for the production $A \rightarrow \beta$ on a in the current state.

Producing the Action and Goto Tables

Accept:

- An item of the form $[S' \rightarrow S \bullet, \text{eof}]$ where S' is the goal symbol indicates the accepting state for the parser.
- This item generates an *accept* action on eof in the current state.

Producing the Action and Goto Tables

Action and Goto tables for our grammar:

Iteration	Item	*Goal*	*List*	*Pair*	(	)	eof
0	CC_0	Ø	CC_1	CC_2	CC_3	Ø	Ø
1	CC_1	Ø	Ø	CC_4	CC_3	Ø	Ø
	CC_2	Ø	Ø	Ø	Ø	Ø	Ø
	CC_3	Ø	Ø	CC_5	CC_6	CC_7	Ø
2	CC_4	Ø	Ø	Ø	Ø	Ø	Ø
	CC_5	Ø	Ø	Ø	Ø	CC_8	Ø
	CC_6	Ø	Ø	CC_9	CC_6	CC_{10}	Ø
	CC_7	Ø	Ø	Ø	Ø	Ø	Ø
3	CC_8	Ø	Ø	Ø	Ø	Ø	Ø
	CC_9	Ø	Ø	Ø	Ø	CC_{11}	Ø
	CC_{10}	Ø	Ø	Ø	Ø	Ø	Ø
4	CC_{11}	Ø	Ø	Ø	Ø	Ø	Ø

	Action Table			*Goto* Table	
State	eof	(	)	*List*	*Pair*
0		s 3		1	2
1	acc	s 3			4
2	r 3	r 3			
3		s 6	s 7		5
4	r 2	r 2			
5			s 8		
6		s 6	s 10		9
7	r 5	r 5			
8	r 4	r 4			
9			s 11		
10			r 5		
11			r 4		

Producing the Action and Goto Tables

Notice that:

- there are no shift/reduce actions associated with any of the non-terminals in the grammar.
- the CC table tells us transitions outright.
- it does not tell us shift, reduce or accept actions.
- an application of a rule does not change state per se. As such our CC table has no state change information for them.
- the number associated with each rule is the rule number, not the state number.