A Note and Example of constructing an SLR parse table

In this note we consider the following small grammar:

$$
\begin{array}{ll}
(1) & E \to E + T \\
(2) & E \to T \\
(3) & T \to T * F \\
(4) & T \to F \\
(5) & F \to (E) \\
(6) & F \to \mathbf{id}
\end{array}
$$

We are trying to produce the *action* and *goto* tables found in Figure 4.31 in the textbook. Once we have these we can deploy the following following parse algorithm (Figure 4.30 in the textbook):

```
set ip to the first symbol of w$;
repeat forever begin
      let s be the state on top of the stack and a the symbol pointed to by ip;
      if action[s, a] = shift s' then begin
            push a then s' on top of the stack;
            advance ip to the next input symbol;
      end
      else if action[s, a] = reduce A → β then begin
            pop 2* | β | of the stack;
            let s' be the state now on top of the stack;
            push A then goto[s', A] on top of the stack;
            output the production A → β;
      end
      else if action[s, a] = accept then
            return;
      else error();
end
```

A *configuration* of an SLR parser is a pair whose first component is the stack and second component the unparsed input:

$$(s_0\ X_1\ s_1\ X_2\ s_2\ \cdots X_m\ s_m, a_i\ a_{i+1}\ \cdots\ a_n\$)$$

which represents the sentential form $X_1\ X_2\ \cdots X_m\ a_i\ a_{i+1}\ \cdots\ a_n$.

We have 4 different moves:

1. If $action[s_m, a_i] =$ `shift` s we get the

$$(s_0\ X_1\ s_1\ X_2\ s_2\ \cdots X_m\ s_m\ a_i\ s, a_{i+1}\ \cdots\ a_n\$)$$

2. If $action[s_m, a_i] =$ `reduce` by $A \to \beta$, we get

$$(s_0\ X_1\ s_1\ X_2\ s_2\ \cdots X_{m-r}\ s_{m-r}\ A\ s, a_i\ a_{i+1}\ \cdots\ a_n\$)$$

   where $s = goto[s_{m-r}, A]$, and r is the length of β. We first pop $2r$ symbols of the stack exposing state s_{m-r}, then push A and the state $s = goto[s_{m-r}, A]$ onto the the stack. The input is left untouched.

3. If $action[s_m, a_i] =$ `accept`, we are done parsing.

4. If $action[s_m, a_i] =$ `error`, stop parsing and produce an error.

Remember, an *item* for a grammar G is a production of G with a dot at some position of the right side. Thus, production $A \to XYZ$ yields the four items:

$$A \to \cdot XYZ$$
$$A \to X \cdot YZ$$
$$A \to XY \cdot Z$$
$$A \to XYZ \cdot$$

and remember a production $A \to \epsilon$ generates only the item $A \to \cdot$

1 The *closure* Operation

If I is a set of items for a grammar G, then $closure(I)$ is the set of items constructed from I by the following two rules:

1. Initially, every item in I is added to $closure(I)$.

2. If $A \to \alpha \cdot B\beta$ is in $closure(I)$ and $B \to \gamma$ is a production, then add $B \to \cdot\gamma$ to I, if it is not already there. We apply this rule until no more new items can be added to $closure(I)$.

Example 4.34
Remembering that we have to add a production $S' \to S$ before using the grammar, where S is the original start symbol we can consider the augmented expression grammar presented on the first page. For completeness, let us first list all the possible items that can be generated from this grammar:

$$E' \to E$$
$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid \mathbf{id}$$

G with the extra start production.

$\underline{E' \to E}$:	$[E' \to \cdot E]$, $[E' \to E \cdot]$
$\underline{E \to E + T}$:	$[E \to \cdot E + T]$, $[E \to E \cdot\ + T]$, $[E \to E + \cdot T]$, $[E \to E + T \cdot]$
$\underline{E \to T}$:	$[E \to \cdot T]$, $[E \to T \cdot]$
$\underline{T \to T * F}$:	$[T \to \cdot T * F]$, $[T \to T \cdot\ * F]$, $[T \to T * \cdot F]$, $[T \to T * F \cdot]$
$\underline{T \to F}$:	$[T \to \cdot F]$, $[T \to F \cdot]$
$\underline{F \to (E)}$:	$[F \to \cdot(E)]$, $[F \to (\cdot E)]$, $[F \to (E \cdot)]$, $[F \to (E) \cdot]$
$\underline{F \to \mathbf{id}}$:	$[F \to \cdot\mathbf{id}]$, $[F \to \mathbf{id} \cdot]$

Now, if $I = \{[E' \to \cdot E]\}$, we compute $closure(I)$ in the following way.

1. Start by by setting $closure(I) = \{[E' \to \cdot E]\}$.
2. Since there is a $\cdot$ immediately left of E we have to add all the items that are derived from the production $E \to \alpha$ **with dots at the left end**, that is, items of the form $E \to \cdot\alpha$. This amounts to the set $\{[E \to \cdot E + T]$,$[E \to \cdot T]\}$.
   We now have $closure(I) = \{[E' \to \cdot E]$, $[E \to \cdot E + T]$,$[E \to \cdot T]\}$.
3. Since the previous step introduced the item $[E \to \cdot T]$, we need to do step (2) again for this item. The set of items that will be added because of this consists of items of the form $T \to \cdot\alpha$, which are the two items in $\{[T \to \cdot T * F]$,$[T \to \cdot F]\}$, which now gives us the following:
   $closure(I) = \{[E' \to \cdot E]$, $[E \to \cdot E + T]$, $[E \to \cdot T]$, $[T \to \cdot T * F]$, $[T \to \cdot F]\}$.
4. The previous step added the item $[T \to \cdot F]$ to $closure(I)$, which means we have to add any item of the form $F \to \cdot\alpha$. Two items satisfy this this, and those are in the set $\{[F \to \cdot(E)]$, $[F \to \cdot\mathbf{id}]\}$. This gives us the final value of $closure(I)$:
   $closure(I) = \{[E' \to \cdot E]$, $[E \to \cdot E + T]$, $[E \to \cdot T]$, $[T \to \cdot T * F]$, $[T \to \cdot F]$, $[F \to \cdot(E)]$, $[F \to \cdot\mathbf{id}]\}$.

The *closure*-set gives rise to two additional sets:

- The set of *kernel items*, which include the initial item, $S' \to \cdot S$, and all items whose dots are not at the left end.
- The set of *nonkernel items*, which have their dots at the left end.

Note, for any set of items I, $closure(I) = closure(kernel(\text{closure(I)}))$; in other words, we only need to store the *kernel* items of a set, the rest can be computed using the *closure* function at any time.

2 The *goto* Operation

In order to compute the *action* and *goto* tables we need to consider another function, the *goto* function or operation. This function is defined in the following way: $goto(I, X)$, where I is a set of items, and X is a grammar symbol (terminal or nonterminal), is defined as:

$$goto(I, X) = closure(\{[A \to \alpha X \cdot \beta] \mid [A \to \alpha \cdot X\beta] \in I\})$$

or more intuitively, for a set of items I and a grammar symbol X, we find all the items in I of the form $[A \to \alpha \cdot X\beta]$ (call this set J), and for a set J' which consists of the same items as J with the dot moved to right of X in all items, we can compute $goto(I, X)$ as $closure(J')$.

Example 4.35
If I is the set of two items $\{[E' \to E\cdot]$,$[E \to E\cdot + T]\}$, and we are looking to compute $goto(I, +)$, we start by finding all items in I where a dot appears left of a $+$. Only the second item $[E \to E\cdot + T]$ satisfies that constraint. We thus have $J = \{[E \to E\cdot + T]\}$ and $J' = \{[E \to E + \cdot T]\}$. Now simply compute $closure(J')$, which is $\{[E \to E + \cdot T]$,$[T \to \cdot T * F]$,$[T \to \cdot F]$,$[F \to \cdot(E)]$,$[F \to \cdot\mathbf{id}]\}$.

3 The $Sets - of - Items$ Construction

Using the *closure* and the *goto* functions we can now compute a *set-of-items* called C in the following way:

```
procedure items(G')
begin
    I_0 := closure({[S' → ·S]});
    C := {I_0};
    i := 1;
    repeat
        for each (I ∈ C, X ∈ G') such that (goto(I, X) ≠ ∅ ∧ goto(I, X) ∉ C) do
            I_i := goto(I, X);
            C := C ∪ {I_i};
            i := i + 1;
        until no more sets can be added to C;
end
```

Example 4.36 We can now, using the above procedure, compute the *canonical collection of sets of LR(0) items* for the grammar shown at the beginning. We start by computing I_0, that is, the closure of $[E' \to \cdot E]$.

$\underline{I_0}$:

$$
\begin{aligned}
I_0 &= closure(\{[E' \to \cdot E]\}) \\
&= closure(\{[E' \to \cdot E],[E \to \cdot E + T],[E \to \cdot T]\}) \\
&= closure(\{[E' \to \cdot E],[E \to \cdot E + T],[E \to \cdot T],[T \to \cdot T * F],[T \to \cdot F]\}) \\
&= closure(\{[E' \to \cdot E],[E \to \cdot E + T],[E \to \cdot T],[T \to \cdot T * F],[T \to \cdot F],[F \to \cdot(E)],[F \to \cdot \mathbf{id}]\}) \\
&= \{[E' \to \cdot E],[E \to \cdot E + T],[E \to \cdot T],[T \to \cdot T * F],[T \to \cdot F],[F \to \cdot(E)],[F \to \cdot \mathbf{id}]\}
\end{aligned}
$$

Having successfully computed I_0 we now proceed to computing $goto(I_0, X)$ for $X \in \{E, T, F, +, *, (,), id\}$.

$\underline{I_1 = goto(I_0, E)}$:

We start by finding the items in I_0 of the form $[A \to \alpha \cdot E\beta]$, that is, the J set as described above. Since only two items in I_0 have the dot left of E, we get: $J = \{[E' \to \cdot E],[E \to \cdot E + T]\}$. Now create J' by moving the dot to the right of E in all items in J. We get $J' = \{[E' \to E\cdot],[E \to E\cdot + T]\}$. Thus:

$$
\begin{array}{lcl}
goto(I_0, E) & = & closure(J') \\
& = & closure(\{[E' \to E\cdot],[E \to E \cdot + T]\}) \\
& = & closure([E' \to E \cdot]) \cup closure([E \to E \cdot + T]) \\
& = & \{[E' \to E \cdot]\} \cup \{[E \to E \cdot + T]\} \\
& = & \{[E' \to E \cdot], [E \to E \cdot + T]\} \\
& =: & I_1
\end{array}
$$

Remember, computing the closure of an item of the form $[A \to \alpha \cdot a\beta]$, where a is a terminal is easy, it only consists of itself, that is, $closure([A \to \alpha \cdot a\beta]) = \{[A \to \alpha \cdot a\beta]\}$. This is because no production of the form $a \to \gamma$ exists as a is a terminal and G is context-free.

We can now add the first entry to our *goto* table:

$$\boxed{goto[0, E] = 1}$$

$\underline{I_2 = goto(I_0, T)}$:

Again, first find J. $J = \{[A \to \alpha \cdot T\beta] \in I_0\} = \{[E \to \cdot T + F],[E \to \cdot T]\}$. We then construct $J' = \{[E \to T \cdot + F],[E \to T \cdot]\}$, and compute like before:

$$
\begin{array}{lcl}
goto(I_0, T) & = & closure(J') \\
& = & closure(\{[E \to T \cdot + F],[E \to T \cdot]\}) \\
& = & closure([E \to T \cdot + F]) \cup closure([E \to T \cdot]) \\
& = & \{[E \to T \cdot + F]\} \cup [E \to T \cdot]\} \\
& = & \{[E \to T \cdot + F], [E \to T \cdot]\} \\
& =: & I_2
\end{array}
$$

We can now add the second entry to our *goto* table:

$$\boxed{goto[0, T] = 2}$$

$\underline{I_3 = goto(I_0, F)}$:

$J = \{[T \to \cdot F]\}$. We then construct $J' = \{[T \to F \cdot]\}$, and compute like before:

$$
\begin{array}{lcl}
goto(I_0, F) & = & closure(J') \\
& = & closure(\{[T \to F \cdot]\}) \\
& = & \{[T \to F \cdot]\} \\
& =: & I_3
\end{array}
$$

which adds yet another entry to the *goto* table:

$$\boxed{goto[0, F] = 3}$$

$\underline{I_4 = goto(I_0, ():}$

$J = \{[F \to \cdot(E)]\}$. $J' = \{[T \to (\cdot E)]\}$, and compute:

$$\begin{array}{lcl} goto(I_0, () & = & closure(J') \\ & = & closure(\{[F \to (\cdot E)]\}) \\ & = & \{[F \to (\cdot E)],\ [E \to \cdot E + T], [E \to \cdot T],\ [T \to \cdot T * F], [T \to \cdot F], \\ & & \quad [F \to \cdot(E)], [F \to \cdot \mathbf{id}]\} \\ & =: & I_4 \end{array}$$

Since (is a terminal, we do not create any entry in the *goto* table. We return to this case when we fill in the *action* table. In short, it adds an entry to $action[0, \mathbf{id}]$ to $shift$ and go to state 4.

$goto(I_0,))$, $goto(I_0, +)$, $goto(I_0, *)$:

$J = \{[A \to \alpha \cdot)\beta] \in I_0\} = \emptyset$. No items in I_0 has a dot before a), thus J is empty, and no *goto* set can be computed. The same goes for + and $*$.

$I_5 = goto(I_0, \mathbf{id})$:

$J = \{[F \to \cdot \mathbf{id}]\}$. $J' = \{[F \to \mathbf{id} \cdot]\}$, thus we get:

$$\begin{array}{lcl} goto(I_0, \mathbf{id}) & = & closure(J') \\ & = & closure(\{[F \to \mathbf{id} \cdot]\}) \\ & = & \{[F \to \mathbf{id} \cdot]\} \\ & =: & I_5 \end{array}$$

We have now considered all possibilities for $goto(I_0, X)$, so we can move on to $goto(I_1, X)$ for all grammar symbols X.

$goto(I_1, X)$:

Recall, $I_1 = \{[E' \to E \cdot], [E \to E \cdot + T]\}$, which makes it obvious that only $goto(I_1, +)$ will yield a non-empty set. For + we have $J = \{[E \to E \cdot + T]\}$. $J' = \{[E \to E + \cdot T]\}$, and

$$\begin{array}{lcl} goto(I_1, E) & = & goto(I_1, T) \\ & = & goto(I_1, F) \\ & = & goto(I_1, () \\ & = & goto(I_1,)) \\ & = & goto(I_1, *) \\ & = & \emptyset \\ goto(I_1, +) & = & closure(J') \\ & = & closure(\{[E \to E + \cdot T]\}) \\ & = & \{[E \to E + \cdot T], [T \to \cdot T + F], [T \to \cdot F],\ [F \to \cdot(E)], [F \to \cdot \mathbf{id}]\} \\ & =: & I_6 \end{array}$$

Again, since + is a terminal, we do not add any entry to the *goto* table.

$goto(I_2, X)$:

Where only + yielded non-empty *goto* sets for I_1, so does * for I_2, so for *, $J = \{[E \to T \cdot * F]\}$, and $J' = \{[E \to T * \cdot F]\}$, so we get

$$\begin{array}{lcl} goto(I_2, E) & = & goto(I_2, T) \\ & = & goto(I_2, F) \\ & = & goto(I_2, () \\ & = & goto(I_2,)) \\ & = & goto(I_2, +) \\ & = & \emptyset \\ goto(I_2, *) & = & closure(J') \\ & = & closure(\{[E \to T * \cdot F]\}) \\ & = & \{[E \to T * \cdot F], [F \to \cdot(E)], [F \to \cdot \mathbf{id}]\} \\ & =: & I_7 \end{array}$$

$goto(I_3, X)$:

Since $I_3 = \{[T \to F \cdot]\}$ no non-empty *goto* set can be computed.

$goto(I_4,))$, $goto(I_4, +)$, $goto(I_4, *)$:

Since I_4 does not contain any items with a dot left of a), a +, or a *, these three *goto* are empty.

$goto(I_4, E)$:

$J = \{[F \to (\cdot E)], [E \to \cdot E + T]\}$. So we get $J' = \{[F \to (E \cdot)], [E \to E \cdot + T]\}$. So we can compute

$$\begin{array}{lcl} goto(I_4, E) & = & closure(J') \\ & = & closure(\{[F \to (E \cdot)], [E \to E \cdot + T]\}) \\ & = & closure(\{[F \to (E \cdot)]\}) \ \cup \ closure(\{[E \to E \cdot + T]\}) \\ & = & \{[F \to (E \cdot)]\} \ \cup \ \{[E \to E \cdot + T]\} \\ & = & \{[F \to (E \cdot)], [E \to E \cdot + T]\} \\ & =: & I_8 \end{array}$$

which gives an entry in the *goto* table:

$$\boxed{goto[4, E] = 8}$$

$goto(I_4, T)$:

$J = \{[E \to \cdot T], [T \to \cdot T * F]\}$. $J' = \{[E \to T \cdot], [T \to T \cdot * F]\}$, and we compute:

$$
\begin{aligned}
goto(I_4, T) &= closure(J') \\
&= closure(\{[E \to T\cdot], [T \to T \cdot * F]\}) \\
&= closure(\{[E \to T\cdot]\}) \ \cup \ closure(\{[T \to T \cdot * F]\}) \\
&= \{[E \to T\cdot]\} \ \cup \ \{[T \to T \cdot * F]\} \\
&= \{[E \to T\cdot], [T \to T \cdot * F]\} \\
&= I_2
\end{aligned}
$$

Since the *goto* set computed is exactly I_2 we don't add anything new set to C, but we do get the following entry in the *goto* table:

$$\boxed{goto[4, T] = 2}$$

$goto(I_4, F)$:

$J = \{[T \to \cdot F]\}$, so $J' = \{[T \to F\cdot]\}$, which gives us the following:

$$
\begin{aligned}
goto(I_4, F) &= closure(J') \\
&= closure(\{[T \to F\cdot]\}) \\
&= \{[T \to F\cdot]\} \\
&= I_3
\end{aligned}
$$

which doesn't give any new sets, but the following entry in the *goto* table:

$$\boxed{goto[I_4, F] = 3}$$

$goto(I_4, ($:

$J = \{[F \to \cdot(E)]\}$, so $J' = \{[F \to (\cdot E)]\}$, which means we get the following:

$$
\begin{aligned}
goto(I_4, (&= closure(J') \\
&= closure(\{[F \to (\cdot E)]\}) \\
&= I_4
\end{aligned}
$$

so again, no new set is added to C.

$goto(I_4,))$, $goto(I_4, +)$, $goto(I_4, *)$, $goto(I_4, \mathbf{id})$:

Since no dot appears before +, * and) in I_4 we get:

$$
\begin{aligned}
goto(I_4, +) &= goto(I_4, *) \\
&= goto(I_4,)) \\
&= \emptyset
\end{aligned}
$$

and for **id**, we get $J = \{[F \to \cdot \mathbf{id}]\}$, with $J' = \{[F \to \mathbf{id}\cdot]\}$, which yields

$$
\begin{array}{rcl}
goto(I_4, \mathbf{id}) & = & closure(J') \\
& = & closure(\{[F \to \mathbf{id}\cdot]\}) \\
& = & \{[F \to \mathbf{id}\cdot]\} \\
& = & I_5
\end{array}
$$

again, no new set is added to C.

$goto(I_5, X)$:

Since $I_5 = \{[F \to \mathbf{id}\cdot]\}$ we get:

$$
goto(I_5, X) = \emptyset, \text{ for all } X.
$$

$goto(I_6, E)$:

No dot appears left of any E in I_6, so $goto(I_6, E) = \emptyset$.

$goto(I_6, T)$:

$J = \{[E \to E + \cdot T],\ [T \to \cdot T * F]\}$, so $J' = \{[E \to E + T\cdot],\ [R \to R\cdot * F]\}$.

$$
\begin{array}{rcl}
goto(I_6, T) & = & closure(J') \\
& = & closure(\{[E \to E + T\cdot],\ [T \to T\cdot * F]\}) \\
& = & closure(\{[E \to E + T\cdot]\}) \ \cup \ closure(\{[T \to T\cdot * F]\}) \\
& = & \{[E \to E + T\cdot]\} \ \cup \ \{[T \to T\cdot * F]\}) \\
& = & \{[E \to E + T\cdot], [T \to T\cdot * F]\}) \\
& =: & I_9
\end{array}
$$

which gives the following entry in the *goto* table:

$$\boxed{goto[6, T] = 9}$$

$goto(I_6, F)$:

$J = \{[T \to \cdot F]\}$, so $J' = \{[T \to F\cdot]\}$, which quickly gives us:

$$
\begin{array}{rcl}
goto(I_6, F) & = & closure(J') \\
& = & closure(\{[T \to F\cdot]\}) \\
& = & \{[T \to F\cdot]\} \\
& = & I_3
\end{array}
$$

No new sets are added to C, but we get an entry in the *goto* table:

$$\boxed{goto[6, F] = 3}$$

$goto(I_6, (),\ goto(I_6, \mathbf{id})$:

For (we get $goto(I_6, () = I_4$, and for **id** we get $goto(I_6, \mathbf{id}) = I_5$.

$goto(I_6,))$, $goto(I_6, *)$, $goto(I_6, +)$:

No dot appear left of either), * or + in I_6, so only empty goto sets are generated.

Only two more sets (I_{10} and I_{11}) are added to C, the rest of the sets computed are either empty or already in C.

$$
\begin{array}{lcl}
goto(I_7, E) & = & \emptyset \\
goto(I_7, T) & = & \emptyset \\
goto(I_7, \mathbf{id}) & = & I_5 \\
goto(I_7, +) & = & \emptyset \\
goto(I_7, *) & = & \emptyset \\
goto(I_7, () & = & I_4 \\
goto(I_7,)) & = & \emptyset
\end{array}
$$

For $goto(I_7, F)$ we get the following: $J = \{[T \to T * \cdot F]\}$, and $J' = \{[T \to T * F \cdot]\}$, which of course gives us $goto(I_7, F) = closure(J') = \{[T \to T * F \cdot]\} =: I_{10}$; in addition we get an entry into the *goto* table:

$$\boxed{goto[7, F] = 10}$$

$$
\begin{array}{lcl}
goto(I_8, E) & = & goto(I_8, T) \\
 & = & goto(I_8, F) \\
 & = & goto(I_8, \mathbf{id}) \\
 & = & goto(I_8, *) \\
 & = & goto(I_8,)) \\
 & = & \emptyset \\
goto(I_8, +) & = & I_6 \\
goto(I_8,)) & = & closure(\{[F \to (E) \cdot]\}) \\
 & = & \{[F \to (E) \cdot]\} \\
 & =: & I_{11}
\end{array}
$$

$$
\begin{array}{lcl}
goto(I_9, E) & = & goto(I_9, T) \\
 & = & goto(I_9, F) \\
 & = & goto(I_9, \mathbf{id}) \\
 & = & goto(I_9, () \\
 & = & goto(I_9, +) \\
 & = & goto(I_9,)) \\
 & = & \emptyset \\
goto(I_9, *) & = & I_7
\end{array}
$$

$$
\begin{array}{lcl}
goto(I_{10}, X) & = & goto(I_{11}, X) \\
 & = & \emptyset, \text{ for all } X
\end{array}
$$

4 The *goto* table

We now have the following entries in the goto table (State i is associated with I_i):

State	E	T	F
0	1	2	3
1			
2			
3			
4	8	2	3
5			
6		9	3
7			10
8			
9			
10			
11			

We can now turn our attention to creating the rest of the parsing table, that is, filling in the *action* entries.

5 The SLR Parsing Table Algorithm

Here is the algorithm for creating both the *goto* and the *action* part of the parsing table.

1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(0) items for G'.
2. State i is constructed for I_i. The parsing actions for state i are determined as follows:
   (a) If $[A \to \alpha \cdot a\beta]$ is in I_i and $goto(I_i, a) = I_j$, then set $action[i, a]$ to `shift` j; a must be a terminal.
   (b) If $[A \to \alpha\cdot]$ is in I_i, then set $action[i, a]$ to `reduce` $[A \to \alpha]$ for all a in FOLLOW(A); here A may not be S'.
   (c) If $[S' \to S]$ is in I_i, then set $action[i, \$]$ to `accept`.
3. The goto transition is created for all nonterminals A using the rule: if $goto(I_i, A) = I_j$ then $goto[i, A] = j$.
4. All entries not defined by rules (2) and (3) are marked `error`.
5. The initial state of the parser is the one constructed from the set of items containing $[S' \to S]$.

6 The SLR Parsing Table (Part 1)

We can easily fill in the parsing table with shift entries, that is, the ones defined by (2a) above

Since $I_0 = \{[E' \to \cdot E], [E \to \cdot E + T], [E \to \cdot T], [T \to \cdot T * F], [T \to \cdot F], [F \to \cdot(E)], [F \to \cdot \mathbf{id}]\}$, the subset of items of the form $[A \to \alpha \cdot a\beta]$ is $\{[F \to \cdot(E)], [F \to \cdot \mathbf{id}]\}$, so we get the following:

- $[F \to \cdot(E)]$: Since $goto(I_0, () = I_4$, we get the entry $action[0, (] = s4$.
- $[F \to \cdot \mathbf{id}]$: Since $goto(I_0, \mathbf{id}) = I_5$, we get the entry $action[0, \mathbf{id}] = s5$.

If we do this for all items $[A \to \alpha \cdot a\beta]$ is any I_i as described above in (2a) we get the following:

	action						*goto*		
State	**id**	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2			s7						
3									
4	s5			s4			8	2	3
5									
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9			s7						
10									
11									

However, we still have not filled in the actions from part (2b), that is, the `reduce` actions.

In order to do so we need to find all sets of items that contain items of the form $[A \to \alpha\cdot]$. By inspecting the 11 item sets we get the following table:

Item set	Items of the form $[A \to \alpha\cdot]$
I_1	$[E' \to E\cdot]$ — this one gave `acc` for $action[1, \$]$
I_2	$[E \to T\cdot]$
I_3	$[T \to F\cdot]$
I_5	$[F \to \mathbf{id}\cdot]$
I_9	$[E \to E + T\cdot]$
I_{10}	$[T \to T * F\cdot]$
I_{11}	$[F \to (E)\cdot]$

In order to finish the parsing table we need to calculate the following $FOLLOW$ sets:

A	$FOLLOW(A)$
E	$\{\$, +,)\}$
T	$\{\$, +,), *\}$
F	$\{\$, +,), *\}$

Remember, E can be followed by \$ because we have added the extra production $E' \rightarrow E$.

We can now compute the last of the entries in the *action* table, that is, the ones that will be `reduce` entries.

- $[E \rightarrow T\cdot]$ in I_2. Since $FOLLOW(E) = \{\$, +,)\}$ we get the following en-ties:
  - $action[2, \$] =$ `reduce` $E \rightarrow T =$ r2
  - $action[2, +] =$ `reduce` $E \rightarrow T =$ r2
  - $action[2,)] =$ `reduce` $E \rightarrow T =$ r2
- $[T \rightarrow F\cdot]$ in I_3. Since $FOLLOW(T) = \{\$, +,), *\}$ we get the following entries:
  - $action[3, \$] =$ `reduce` $T \rightarrow F =$ r4
  - $action[3, +] =$ `reduce` $T \rightarrow F =$ r4
  - $action[3,)] =$ `reduce` $T \rightarrow F =$ r4
  - $action[3, *] =$ `reduce` $T \rightarrow F =$ r4
- $[F \rightarrow \mathbf{id}\cdot]$ in I_5. Since $FOLLOW(F) = \{\$, +,), *\}$ we get the following entires:
  - $action[5, \$] =$ `reduce` $F \rightarrow \mathbf{id} =$ r6
  - $action[5, +] =$ `reduce` $F \rightarrow \mathbf{id} =$ r6
  - $action[5,)] =$ `reduce` $F \rightarrow \mathbf{id} =$ r6
  - $action[5, *] =$ `reduce` $F \rightarrow \mathbf{id} =$ r6
- $[E \rightarrow E + T\cdot]$ in I_9. Since $FOLLOW(E) = \{\$, +,)\}$ we get the following entices:
  - $action[9, \$] =$ `reduce` $E \rightarrow E + T =$ r1
  - $action[9, +] =$ `reduce` $E \rightarrow E + T =$ r1
  - $action[9,)] =$ `reduce` $E \rightarrow E + T =$ r1
- $[T \rightarrow T * F\cdot]$ in I_{10}. Since $FOLLOW(T) = \{\$, +,), *\}$ we get the following en-ties:
  - $action[10, \$] =$ `reduce` $T \rightarrow T * F =$ r3
  - $action[10, +] =$ `reduce` $T \rightarrow T * F =$ r3
  - $action[10,)] =$ `reduce` $T \rightarrow T * F =$ r3
  - $action[10, *] =$ `reduce` $T \rightarrow T * F =$ r3
- $[F \rightarrow (E)\cdot]$ in I_{11}. Since $FOLLOW(F) = \{\$, +,), *\}$ we get the following en-ties:
  - $action[11, \$] =$ `reduce` $F \rightarrow (E) =$ r5
  - $action[11, +] =$ `reduce` $F \rightarrow (E) =$ r5
  - $action[11,)] =$ `reduce` $F \rightarrow (E) =$ r5
  - $action[11, *] =$ `reduce` $F \rightarrow (E) =$ r5

7 The SLR Parsing Table (Part 2)

We can now easily fill the reduce actions into the table and get:

	action						*goto*		
State	**id**	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Note, the number associated with the `reduce` action, e.g., r3 means reduce by the production numbered (3) in the original grammar on page 1.

8 One Last Example

Let us parse the string **id** $*$ **id** $+$ **id** using our SLR parsing table.

	Stack	Input	Action
(1)	0	**id** * **id** + **id** $	`shift` **id** 5 as $action[0, \mathbf{id}] = \text{s5}$
(2)	0 **id** 5	* **id** + **id** $	`reduce` by $[F \to \mathbf{id}]$ as $action[5, *] = \text{r6}$
			`goto` state 3 as $goto[0, F] = 3$
(3)	0 F 3	* **id** + **id** $	`reduce` by $[T \to F]$ as $action[3, *] = \text{r4}$
			`goto` state 3 as $goto[0, T] = 2$
(4)	0 T 2	* **id** + **id** $	`shift` * 7 as $action[2, *] = \text{s7}$
(5)	0 T 2 * 7	**id** + **id** $	`shift` **id** 5 as $action[7, \mathbf{id}] = \text{s5}$
(6)	0 T 2 * 7 **id** 5	+ **id** $	`reduce` by $[F \to \mathbf{id}]$ as $action[5, +] = \text{r6}$
			`goto` state 10 as $goto[7, F] = 10$
(7)	0 T 2 * 7 F 10	+ **id** $	`reduce` by $[T \to T * F]$ as $action[10, +] = \text{r3}$
			`goto` state 2 as $goto[0, T] = 2$
(8)	0 T 2	+ **id** $	`reduce` by $[E \to T]$ as $action[2, + = \text{r2}$
			`goto` state 1 as $goto[0, E] = 1$
(9)	0 E 1	+ **id** $	`shift` + 6 as $action[1, +] = \text{s6}$
(10)	0 E 1 + 6	**id** $	`shift` **id** 5 as $action[6, \mathbf{id}] = \text{s5}$
(11)	0 E 1 + 6 **id** 5	$	`reduce` by $[F \to \mathbf{id}]$ as $action[5, \$] = \text{r6}$
			`goto` state 3 as $goto[6, F] = 3$
(12)	0 E 1 + 6 F 3	$	`reduce` by $[T \to F]$ as $action[3, \$] = \text{r4}$
			`goto` state 9 as $goto[6, T] = 9$
(13)	0 E 1 + 6 T 9	$	`reduce` by $[E \to E + T]$ as $action[9, \$] = \text{r1}$
			`goto` state 1 as $goto[0, E] = 1$
(14)	0 E 1	$	accept as $action[1, \$] =$ `accept`