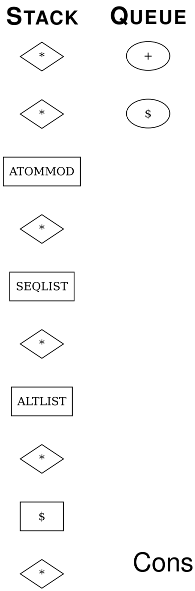
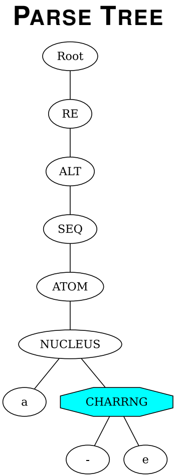
Operation: token `char` match (`char`)



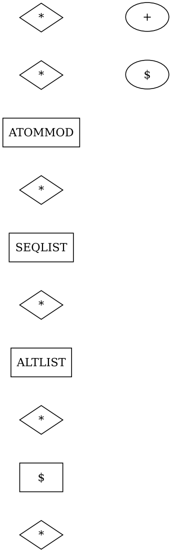


Consider the **semantic action** for converting the $NUCLEUS$ of `a-e` to a `range` node.

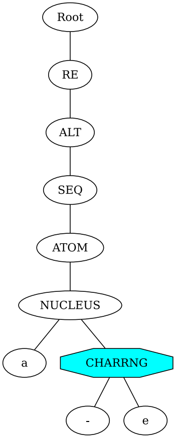
Input RE: **`a-e+`**

Operation: token `char` match (`char`)

STACK QUEUE



PARSE TREE

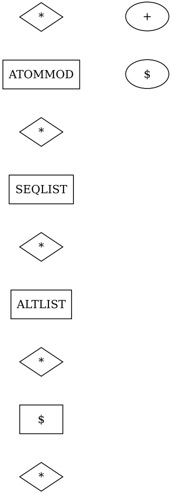


```
CHARRNG_dash_char( $parent$, $node$ )
  $rangeNode \leftarrow node($range$, children = [parent.firstChild, node.lastChild])$
  replace $parent$ with $rangeNode$ in parse tree
```

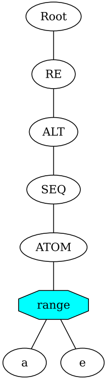
Input RE: **`a-e+`**

Operation: end of *CHARRNG* production

STACK **QUEUE**



PARSE TREE



Input RE: **`a-e+`**

Context Available to LL Semantic Actions

Semantic actions in LL (recursive-descent) parses have an execution context with access to the following information:

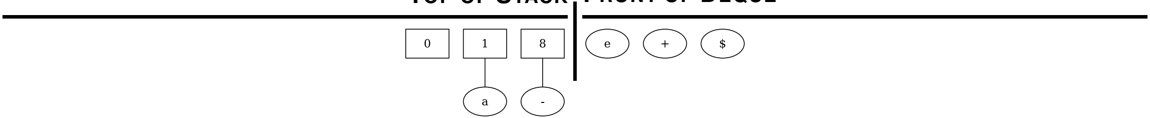
1. The node and all it's descendants,
2. The node's parent (in fact, its ancestors all the way to the starting goal root node),
3. The node's left hand siblings

`CHARRNG_dash_char` took advantage of this information and made a substantial change to the tree under the $ATOM$ node.

Now let's consider the same semantic action logic during an LR parse of the same input. . .

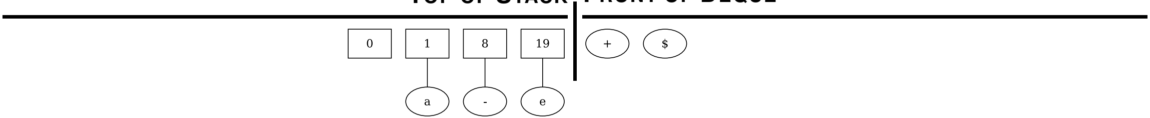
Operation: shift dash to stack, goto state 8

TOP OF STACK **FRONT OF DEQUE**

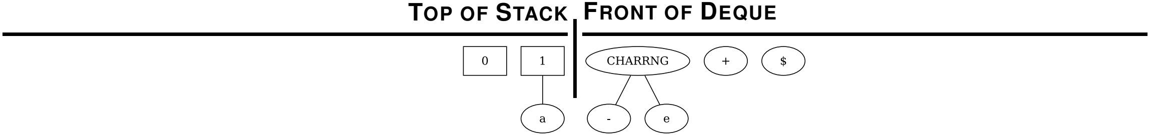


Operation: shift char to stack, goto state 19

TOP OF STACK **FRONT OF DEQUE**



Operation: post LR reduction — before *CHARRNG* rule 16 SDT procedure



```
CHARRNG_dash_char( $parent$, $node$ )
   $rangeNode \leftarrow node(\texttt{range}, children = [parent.firstChild, node.lastChild])$
   replace $parent$ with $rangeNode$ in parse tree
```

This is going to cause problems! *parent* is unknown, and the left hand *char* is still in the stack,it's not in the same subtree as the *CHARRNG* node!

Input RE: **`a-e+`**

Context Available to LR Semantic Actions

What information (compared to semantic actions during LL parses) do we have in LR parses?

Context Available to LR Semantic Actions

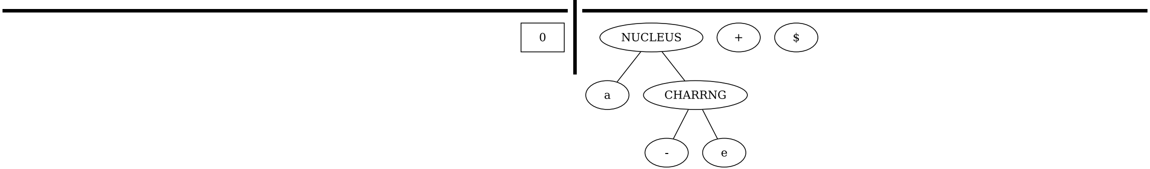
What information (compared to semantic actions during LL parses) do we have in LR parses?

1. Only one: The node and all it's descendants,
2. The node's parent (in fact, its ancestors all the way to the starting goal root node),
3. The node's left hand siblings

But we can work around this by moving the $CHARRNG$ logic from the `CHARRNG_dash_char` semantic action to the `NUCLEUS_char_CHARRNG` semantic action. . .

Operation: post LR reduction — before $NUCLEUS$ rule 14 SDT procedure

TOP OF STACK **FRONT OF DEQUE**

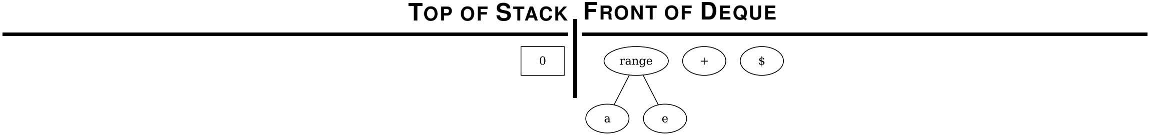


```
NUCLEUS_char_CHARRNG( $node$ )
    $rangeNode \leftarrow node(\texttt{range}, children = [node.firstChild, node.lastChild.lastChild])$
    replace $node$ with $rangeNode$ in parse tree
```

We've lost the *parent* argument to the semantic action, since it isn't known in LR parses.

Input RE: **`a-e+`**

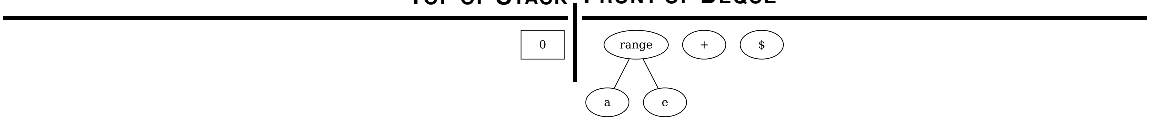
Operation: post LR reduction — after $NUCLEUS$ rule 14 SDT procedure



Input RE: **`a-e+`**

Operation: post LR reduction — after $NUCLEUS$ rule 14 SDT procedure

TOP OF STACK **FRONT OF DEQUE**



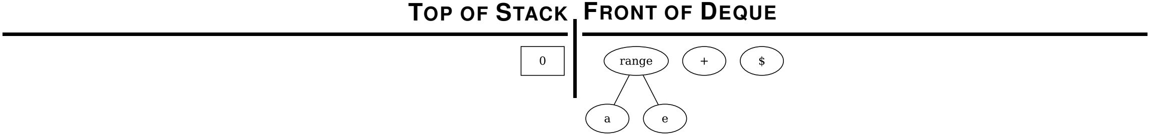
LR_Parse_Failure

`range` $\notin \Sigma_{\$} \cup N$

The LR table does not have a `range` column!

Input RE: **`a-e+`**

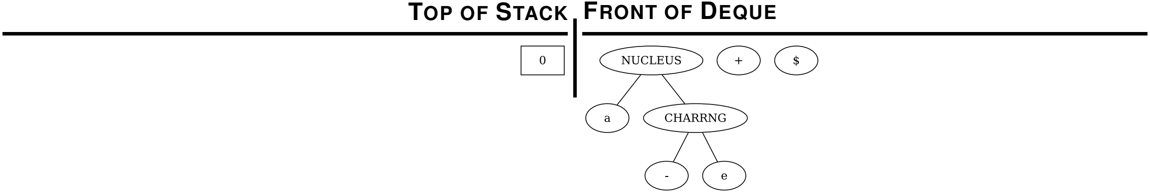
Operation: post LR reduction — after $NUCLEUS$ rule 14 SDT procedure



But we can **delay** the re-typing of $NUCLEUS$ to `range` by “tagging” the node with the appropriate attribute and letting the $ATOM$ **parent change the node type**.

Input RE: **`a-e+`**

Operation: post LR reduction — before $NUCLEUS$ rule 14 SDT procedure

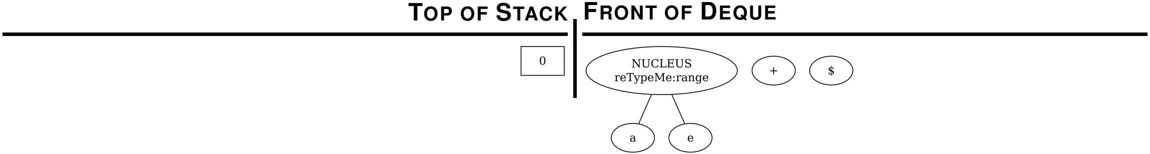


```
NUCLEUS_char_CHARRNG( $node$ )
  $rangeNode \leftarrow node(NUCLEUS, children = [node.firstChild, node.lastChild.lastChild])$
  set attribute $rangeNode.reTypeMe \leftarrow$ range
  replace $node$ with $rangeNode$ in parse tree
```

We postpone the re-typing of $NUCLEUS$ with a node attribute.

Input RE: **`a-e+`**

Operation: post LR reduction — after $NUCLEUS$ rule 14 SDT procedure



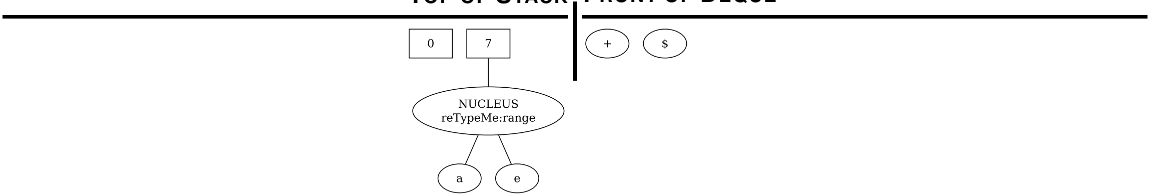
```
NUCLEUS_char_CHARRNG( $node$ )
  $rangeNode \leftarrow node(NUCLEUS, children = [node.firstChild, node.lastChild.lastChild])$
  set attribute $rangeNode.reTypeMe \leftarrow$ range
  replace $node$ with $rangeNode$ in parse tree
```

We postpone the re-typing of $NUCLEUS$ with a node attribute.

Input RE: **a-e+**

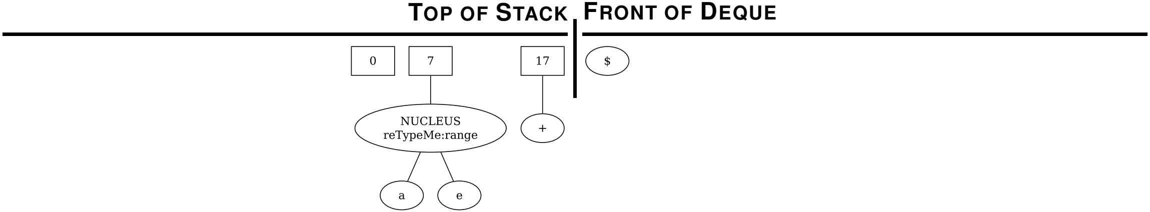
Operation: shift NUCLEUS to stack, goto state 7

Top of Stack **Front of Deque**



Input RE: **`a-e+`**

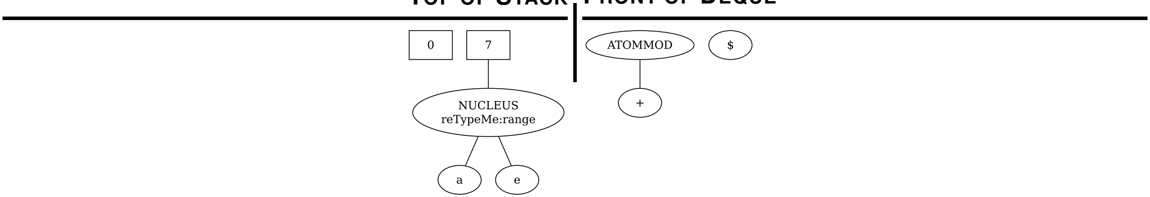
Operation: shift plus to stack, goto state 17



Input RE: **`a-e+`**

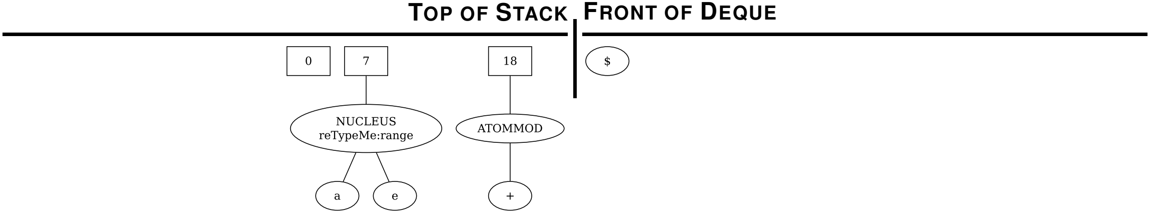
Operation: reduce by rule 11 $ATOMMOD \rightarrow plus$

TOP OF STACK **FRONT OF DEQUE**



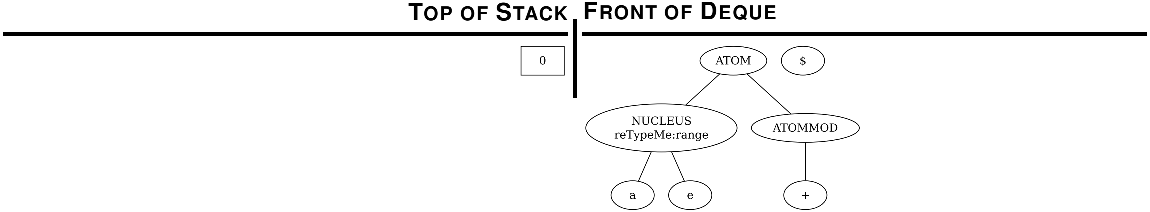
Input RE: **`a-e+`**

Operation: shift ATOMMOD to stack, goto state 18



Input RE: **`a-e+`**

Operation: post LR reduction — before *ATOM* rule 9 SDT procedure

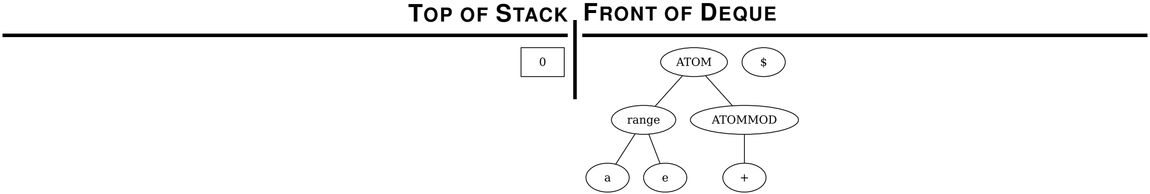


```
ATOM_NUCLEUS_ATOMMOD( $node$ )
  $nucChild \leftarrow node.NUCLEUS$
  if ( $nucChild.reTypeMe$ exists ) then (
    $nucChild.type \leftarrow nucChild.reTypeMe$
    remove $nucChild.reTypeMe$
  )
```

Look for and follow a *reTypeMe* attribute in our *NUCLEUS* child.

Input RE: **a-e+**

Operation: post LR reduction — after $ATOM$ rule 9 SDT procedure



```
ATOM_NUCLEUS_ATOMMOD( $node$ )
  $nucChild \leftarrow node.NUCLEUS$
  if ( $nucChild.reTypeMe$ exists ) then (
    $nucChild.type \leftarrow nucChild.reTypeMe$
    remove $nucChild.reTypeMe$
  )
```

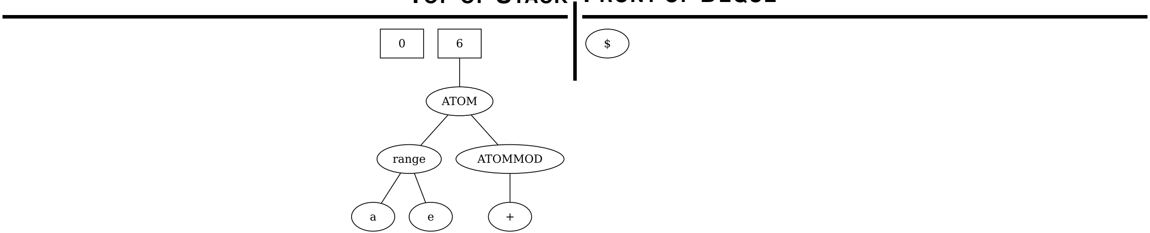
Look for and execute a $reTypeMe$ attribute in our $NUCLEUS$ child.

Yay! it finally works. . .

Input RE: **`a-e+`**

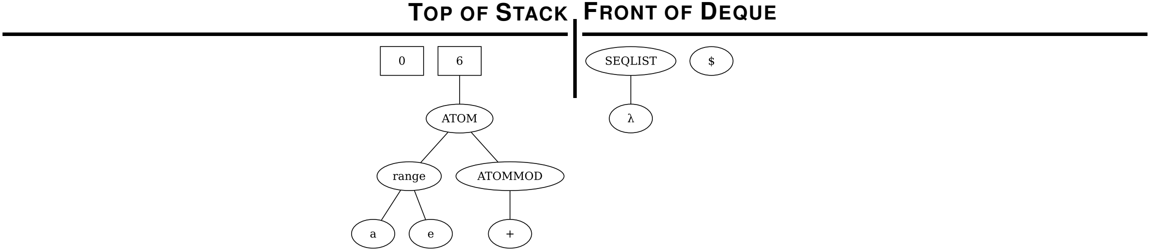
Operation: shift ATOM to stack, goto state 6

TOP OF STACK **FRONT OF DEQUE**



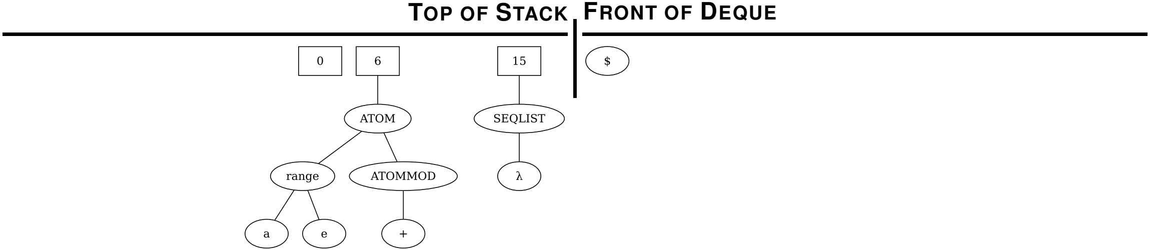
Input RE: **`a-e+`**

Operation: reduce by rule 8 $SEQLIST \rightarrow \lambda$



Input RE: **`a-e+`**

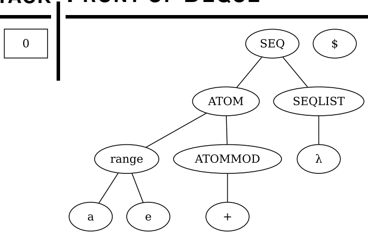
Operation: shift SEQLIST to stack, goto state 15



Input RE: **a-e+**

Operation: reduce by rule 5 $SEQ \rightarrow ATOM\ SEQLIST$

TOP OF STACK **FRONT OF DEQUE**



(There are more λ rules in the parse, but we stop here. . .)

Input RE: **`a-e+`**

Well, that was Ugly : (

Consider what we've just done:

i. We moved all the semantic actions associated with $CHARRNG$ into two other procedures, **one of which, $ATOM$'s semantic action** is two "grammar generations away" from $CHARRNG$.

ii. Who in there right mind would look in the `ATOM_NUCLEUS_ATOMMOD` semantic action thinking "Oh, that's where `range` nodes must be created." No one would.

iii. What I've demonstrated is "spaghetti logic," which is worse than spaghetti code because there are programming tools that can help you figure out spaghetti code...

iv. And the situation becomes worse when more SDT logic is added (we've been working on only one non-terminal to RE expression tree translation!)

v. Don't pursue this method of implementing SDT in LR parses. You've been warned.

Instead, we need to be smarter in our LR parsing. . .

Two Better LR+SDT Approaches

Add an `astStack` A node will have full control over its descendents and itself.

Nodes still don't know their *parent* or left hand siblings at the time of execution.

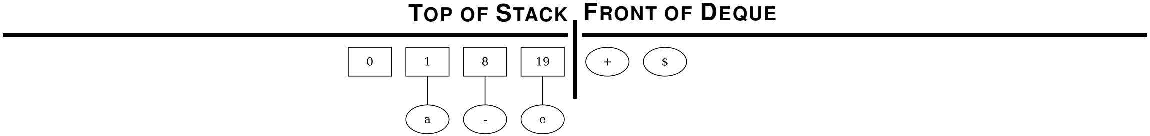
Delay execution Wait until semantic actions have the same execution context as in an LL parse.

Semantic actions can be identical to logic used in LL parses.

Nodes know their *parents*, their left hand siblings, and have full control over their descendents and themselves.

Requires slight modification to tree node structures.

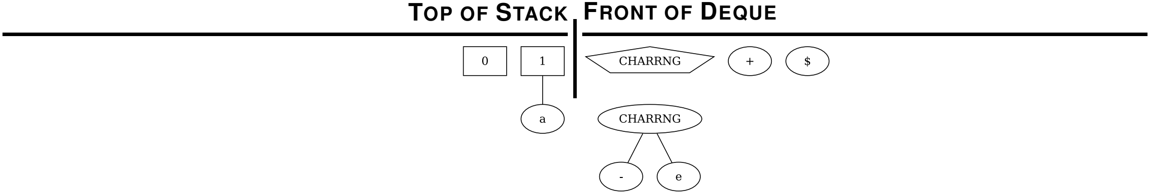
Operation: shift char to stack, goto state 19



Adding an `astStack` to an LR parse.

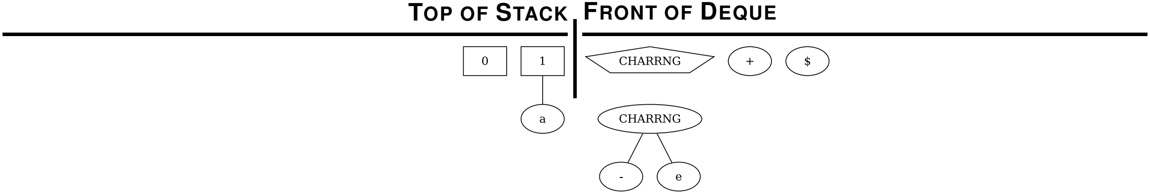
$LRtable[19][+]$ is a reduce action...

Operation: reduce by rule 16 $CHARRNG \rightarrow dash\ char$



You can think of the `astStack` as a separate data structure, or the entries of the stack as pointers or members of the deque elements. There is an element in the `astStack` for each non-terminal at the front of the deque (incidentally, non-terminals appear **only** at the front of the deque).

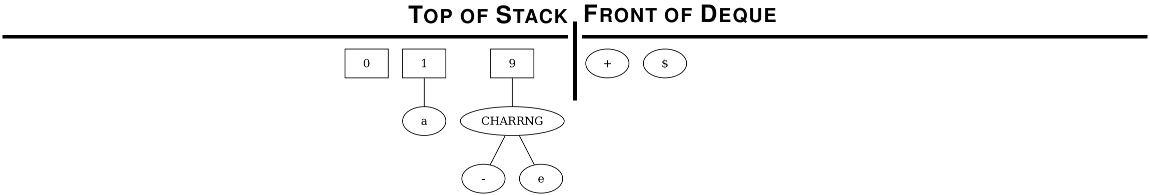
Operation: reduce by rule 16 $CHARRNG \rightarrow dash\ char$



Now the deque holds a **non-terminal placeholder**; not the root of a subtree. The slides draw these special placeholders in **pentagons**.

The placeholders permit the root of the subtree to be a non-grammar symbol (such as `range`). The algorithm uses the **placeholder symbol** as the LR table column entry to look up the next parsing action (shift, reduce).

Operation: shift CHARRNG to stack, goto state 9

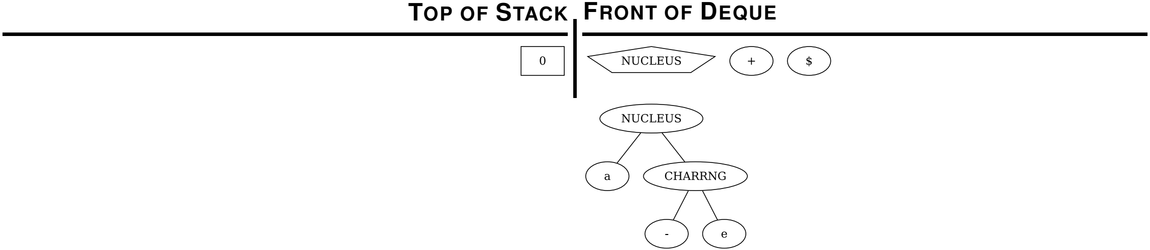


When a **shift** action occurs on a placeholder:

i. Discard the placeholder

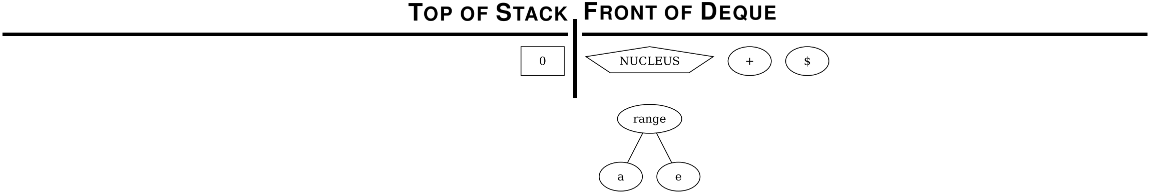
ii. Connect the associated `astStack` element (IOW: pop the `astStack`) to the new state being shifted to the left “knitting needle.”

Operation: post LR reduction — before $NUCLEUS$ rule 14 SDT procedure



```
NUCLEUS_char_CHARRNG( $node$ )
  $rangeNode \leftarrow node(\texttt{range}, children = [node.firstChild, node.lastChild.lastChild])$
  replace $node$ with $rangeNode$ in parse tree
```

Operation: post LR reduction — after $NUCLEUS$ rule 14 SDT procedure

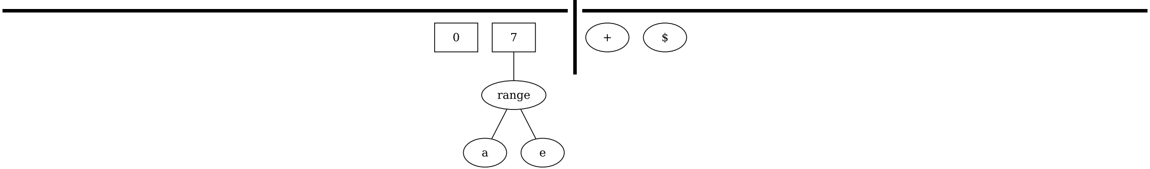


```
NUCLEUS_char_CHARRNG( $node$ )
  $rangeNode \leftarrow node(\texttt{range}, children = [node.firstChild, node.lastChild.lastChild])$
  replace $node$ with $rangeNode$ in parse tree
```

Now the `range` node root of the subtree doesn't interfere with LR table column look up :)

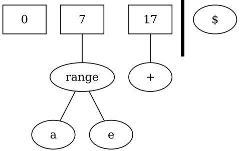
Operation: shift range to stack, goto state 7

TOP OF STACK **FRONT OF DEQUE**

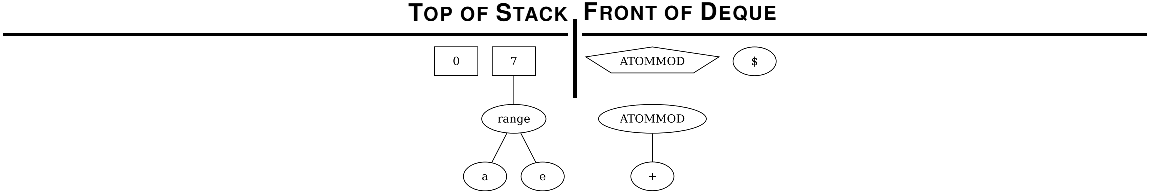


Operation: shift plus to stack, goto state 17

TOP OF STACK **FRONT OF DEQUE**

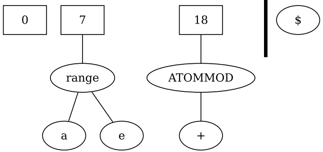


Operation: reduce by rule 11 $ATOMMOD \rightarrow plus$



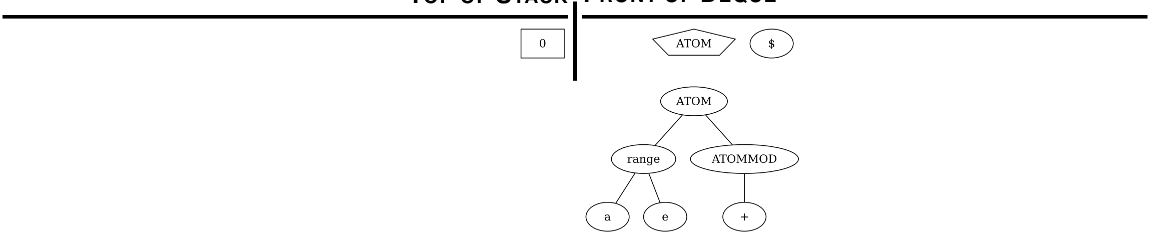
Operation: shift ATOMMOD to stack, goto state 18

TOP OF STACK **FRONT OF DEQUE**



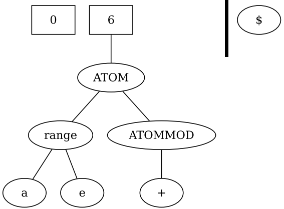
Operation: reduce by rule 9 $ATOM \to NUCLEUS\,ATOMMOD$

TOP OF STACK **FRONT OF DEQUE**

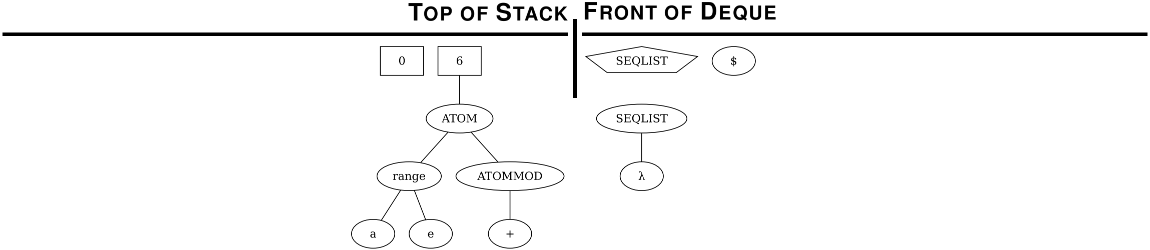


Operation: shift ATOM to stack, goto state 6

TOP OF STACK **FRONT OF DEQUE**

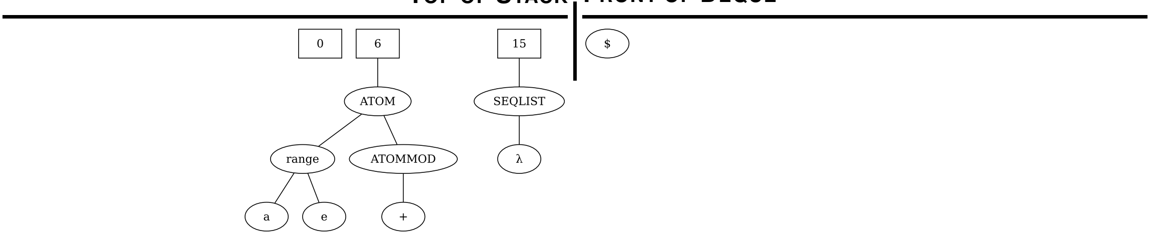


Operation: reduce by rule 8 $SEQLIST \rightarrow \lambda$



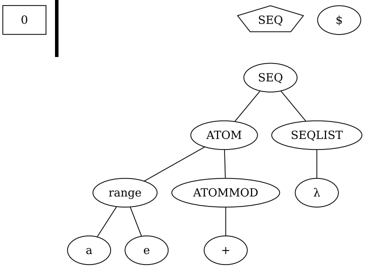
Operation: shift SEQLIST to stack, goto state 15

TOP OF STACK **FRONT OF DEQUE**

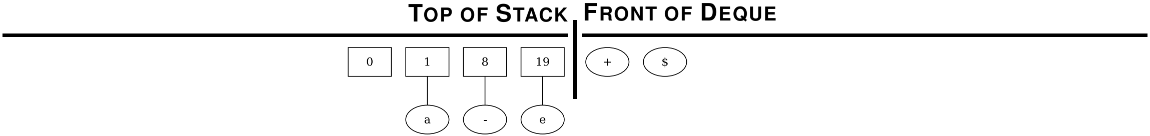


Operation: reduce by rule 5 $SEQ \rightarrow ATOM\ SEQLIST$

Top of Stack **Front of Deque**



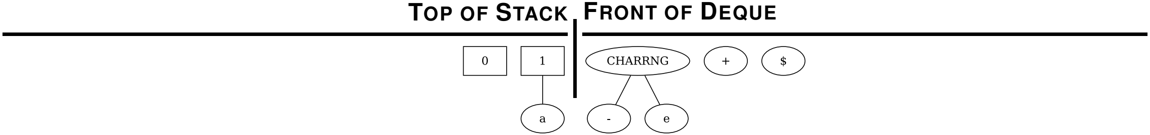
Operation: shift char to stack, goto state 19



Delayed execution in an LR Parse.

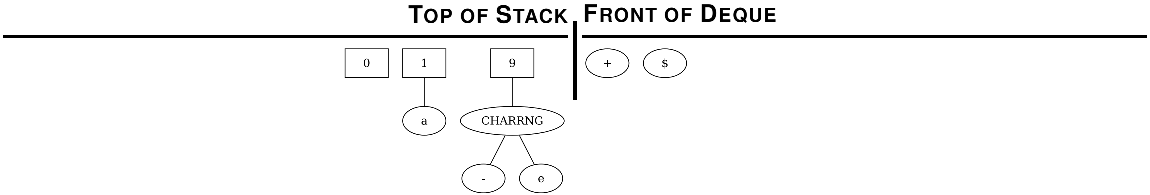
$LRtable[19][+]$ is a reduce action...

Operation: reduce by rule 16 $CHARRNG \to dash\ char$

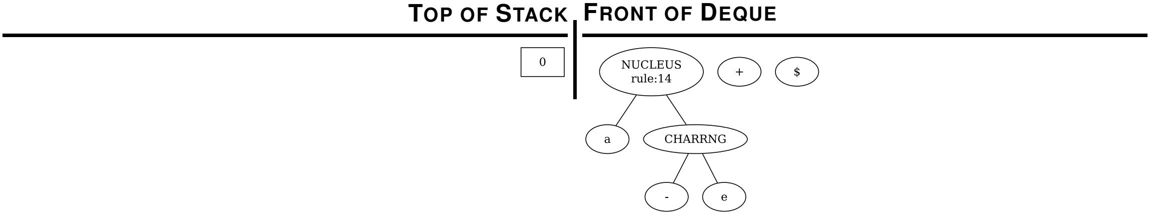


No $CHARRNG$ semantic action, we use the $NUCLEUS$ reduction for the semantic action. . .

Operation: shift CHARRNG to stack, goto state 9



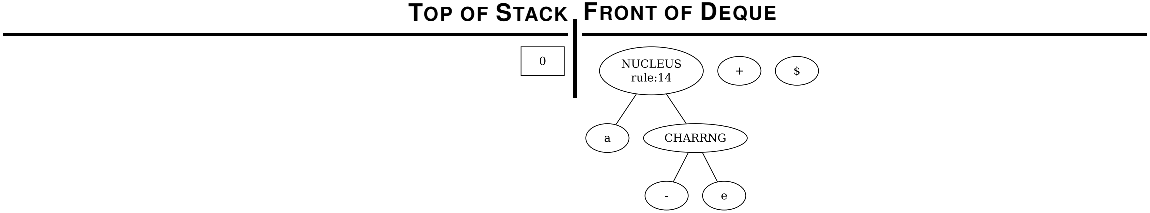
Operation: reduce by rule 14 $NUCLEUS \rightarrow char\, CHARRNG$



We've just reduced to a $NUCLEUS$ node, but **we don't execute the semantic action** for $NUCLEUS$ yet!

(If we did, this implementation would be pretty poorly named.)

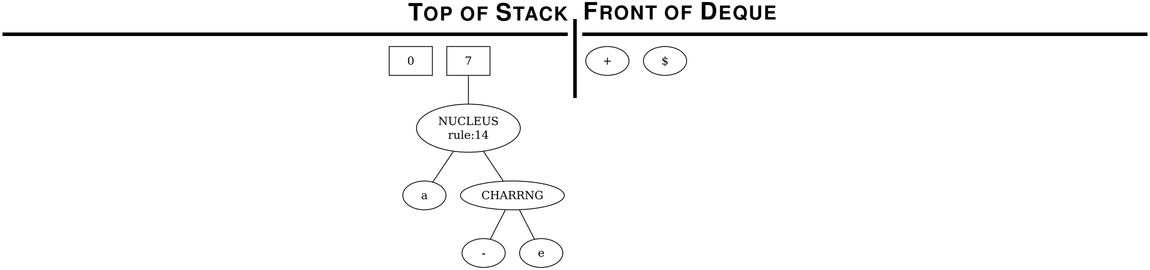
Operation: reduce by rule 14 $NUCLEUS \rightarrow char\, CHARRNG$



We've just reduced to a $NUCLEUS$ node, but **we don't execute the semantic action** for $NUCLEUS$ yet!

Instead we will tag the $NUCLEUS$ node with the production rule number (or a pointer to this production rule's semantic action). This is the `rule:14` attribute...

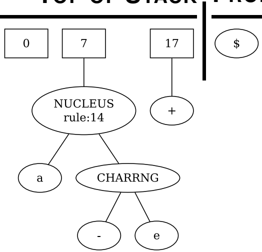
Operation: shift NUCLEUS to stack, goto state 7



. . . and we blissfully continue on with the parse.

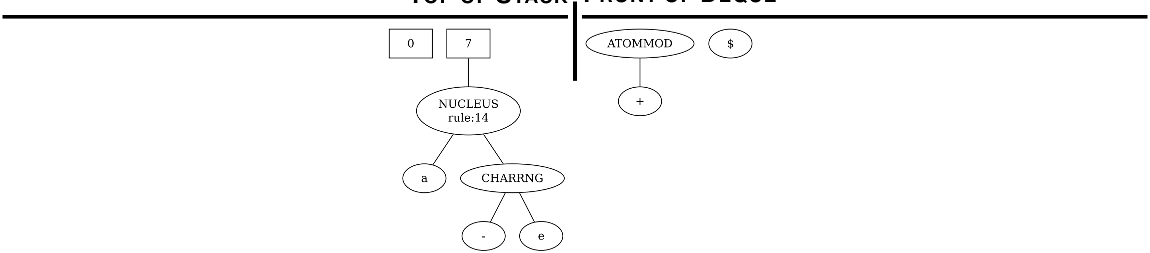
Operation: shift plus to stack, goto state 17

TOP OF STACK **FRONT OF DEQUE**



Operation: reduce by rule 11 $ATOMMOD \to plus$

TOP OF STACK **FRONT OF DEQUE**

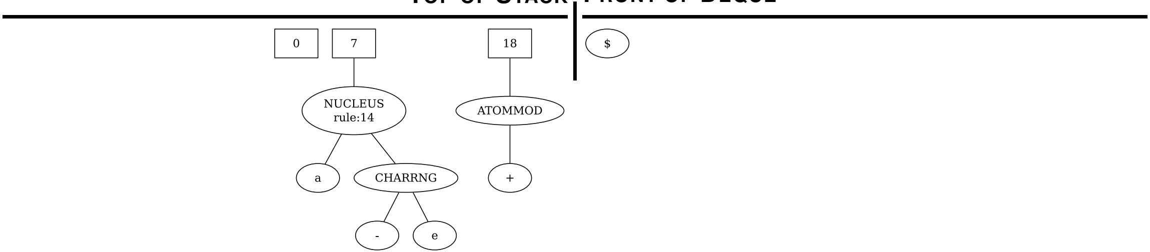


If $ATOMMOD$ **had a semantic action**, we would have tagged the node in this step.

But in this example it doesn't.

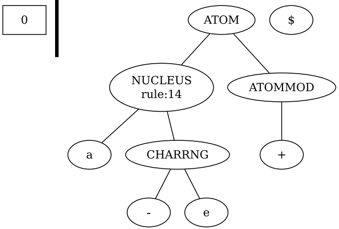
Operation: shift ATOMMOD to stack, goto state 18

TOP OF STACK **FRONT OF DEQUE**



Operation: post LR reduction — before $NUCLEUS$ rule 14 SDT procedure

TOP OF STACK **FRONT OF DEQUE**



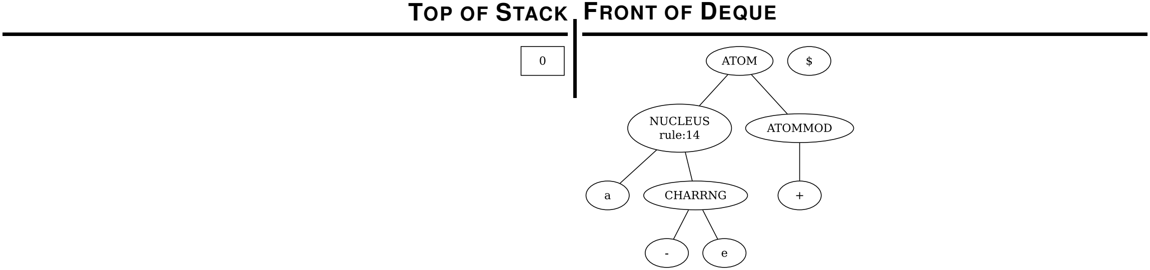
OK! Time for some (semantic) action!

Any time a reduction is performed, inspect each immediate child of the root node.

If the child has been “tagged” with a semantic action, execute it now.

```
foreach ( $child$ in $parent.children$ from left to right ) do (
  if ( $child.rule$ exists ) then (
    $semAction \leftarrow$ semantic action for $child.rule$
    call $semAction(parent, child)$
  )
)
```

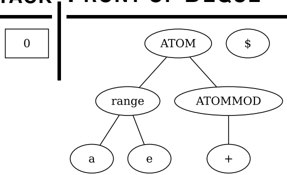
Operation: post LR reduction — before $NUCLEUS$ rule 14 SDT procedure



```
NUCLEUS_char_CHARRNG( $parent$, $node$ )
  $rangeNode \leftarrow node(\texttt{range}, children = [node.firstChild, node.lastChild.lastChild])$
  replace $node$ with $rangeNode$ in $parent$
```

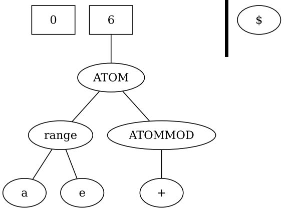
Operation: post LR reduction — after $NUCLEUS$ rule 14 SDT procedure

TOP OF STACK **FRONT OF DEQUE**



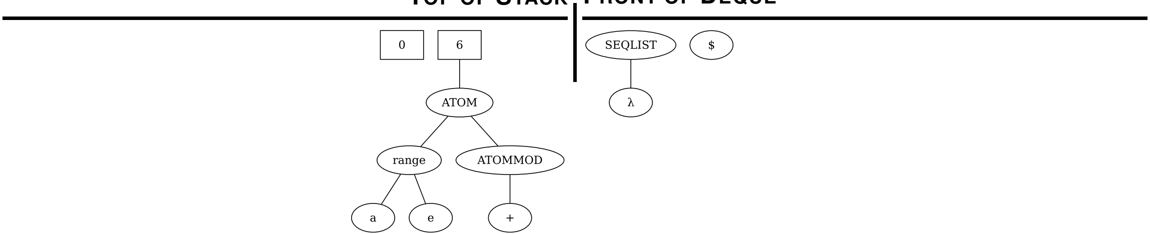
Operation: shift ATOM to stack, goto state 6

TOP OF STACK **FRONT OF DEQUE**



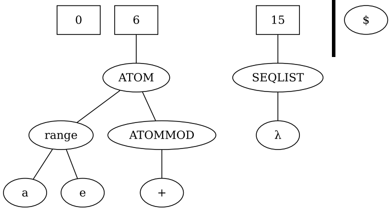
Operation: reduce by rule 8 $SEQLIST \rightarrow \lambda$

Top of Stack **Front of Deque**



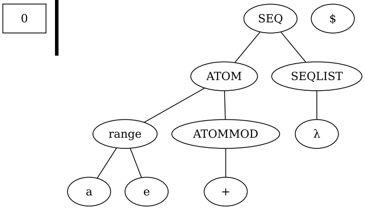
Operation: shift SEQLIST to stack, goto state 15

TOP OF STACK **FRONT OF DEQUE**

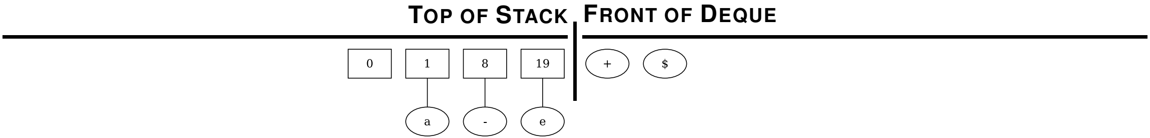


Operation: reduce by rule 5 $SEQ \rightarrow ATOM\ SEQLIST$

Top of Stack **Front of Deque**



Operation: shift char to stack, goto state 19

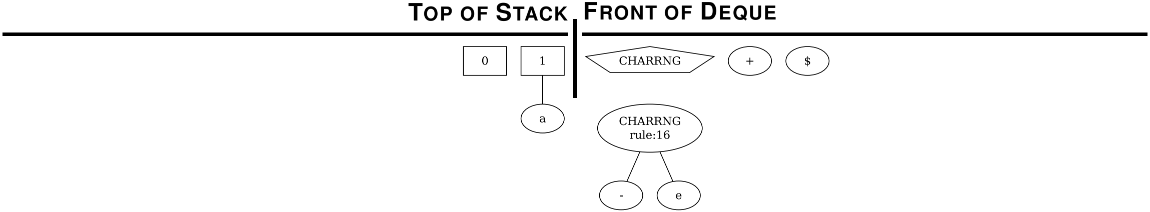


Too Better LR+SDT Approach

It’s no joke! Combine the two techniques, this allows us to put the logic for $CHARRNG$ **entirely** at the $CHARRNG$ semantic action (where it belongs, IMHO).

$LRtable[19][+]$ is a reduce action. . .

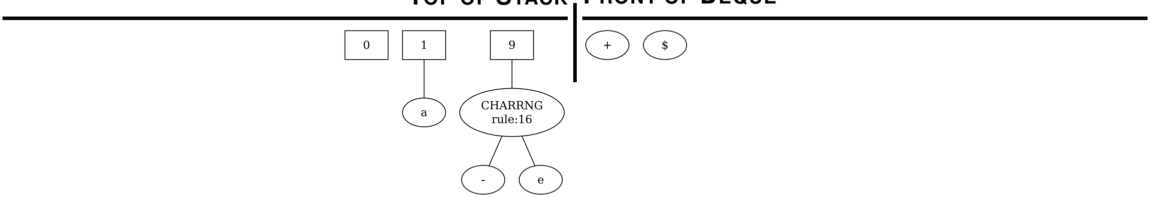
Operation: reduce by rule 16 $CHARRNG \rightarrow dash\ char$



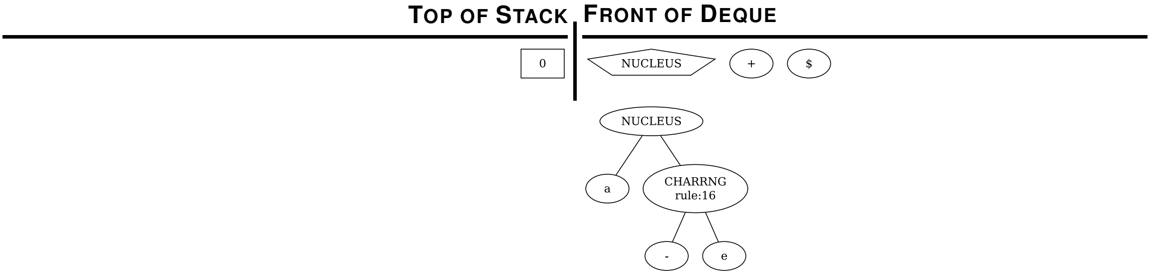
Tag the *CHARRNG* node with the production rule number (or a pointer to this production rule's semantic action). This is the `rule:16` attribute...

Operation: shift CHARRNG to stack, goto state 9

TOP OF STACK **FRONT OF DEQUE**



Operation: post LR reduction — before *CHARRNG* rule 16 SDT procedure



TOP OF STACK **FRONT OF DEQUE**

0

NUCLEUS

+

$

NUCLEUS

a

CHARRNG
rule:16

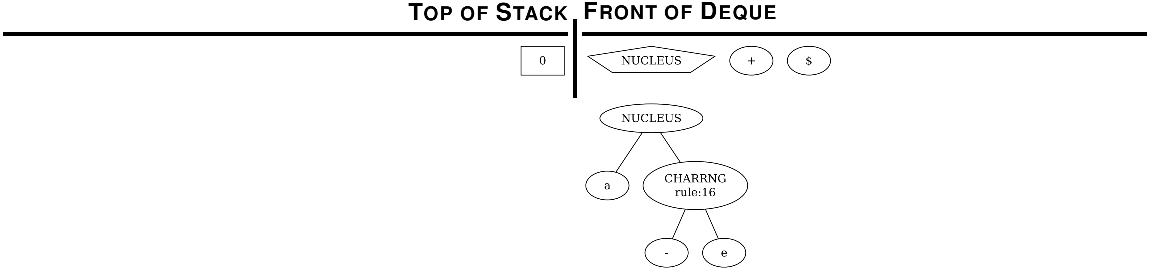
-

e

Any time a reduction is performed, inspect each immediate child of the root node.
If the child has been “tagged” with a semantic action, execute it now.

```
foreach ( child in parent.children from left to right ) do (
  if ( child.rule exists ) then (
    semAction ← semantic action for child.rule
    call semAction(parent, child)
  )
)
```

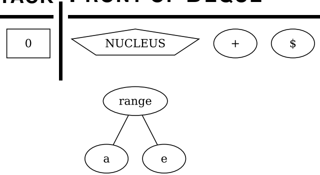
Operation: post LR reduction — before $CHARRNG$ rule 16 SDT procedure



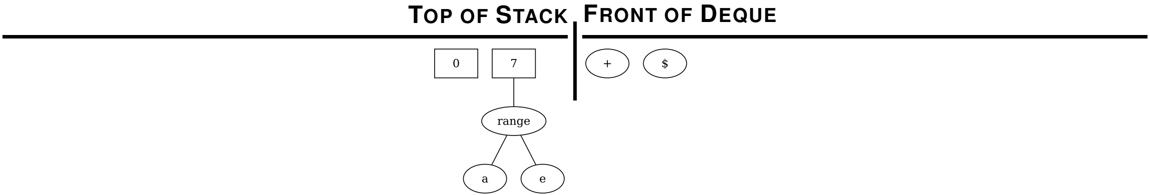
```
CHARRNG_dash_char( $parent$, $node$ )
  $rangeNode \leftarrow node(\texttt{range}, children = [parent.firstChild, node.lastChild])$
  replace $parent$ with $rangeNode$ in parse tree
```

Operation: post LR reduction — after *CHARRNG* rule 16 SDT procedure

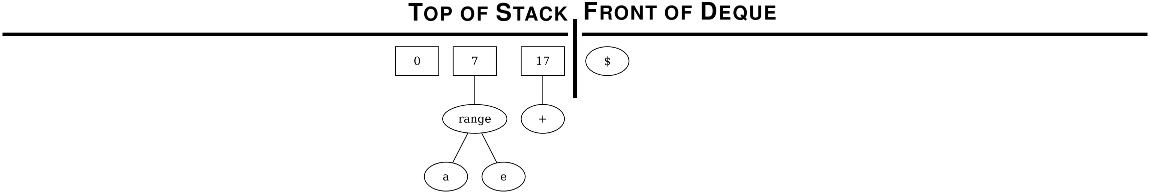
TOP OF STACK **FRONT OF DEQUE**



Operation: shift range to stack, goto state 7



Operation: shift plus to stack, goto state 17

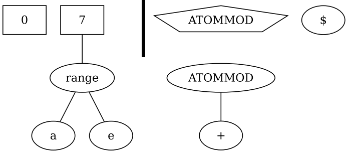


TOP OF STACK

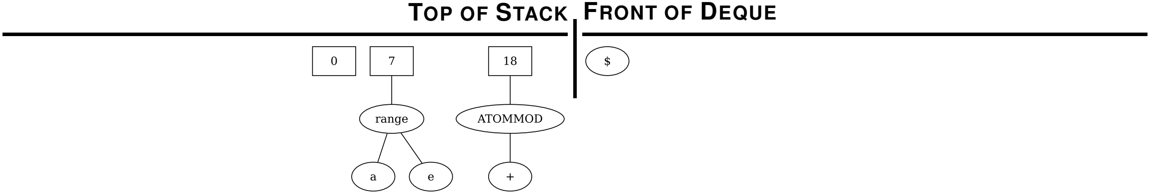
FRONT OF DEQUE

Operation: reduce by rule 11 $ATOMMOD \to plus$

Top of Stack **Front of Deque**

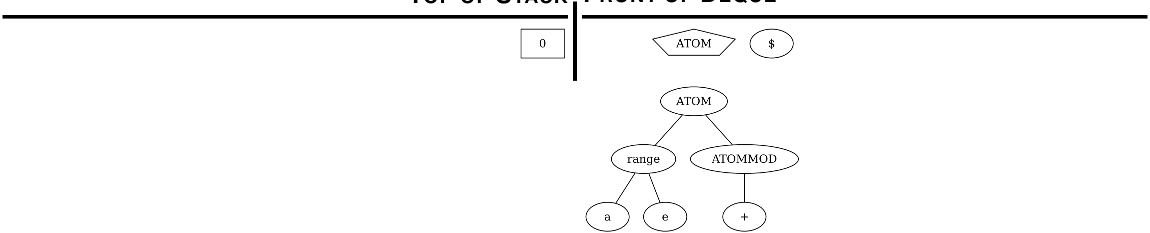


Operation: shift ATOMMOD to stack, goto state 18



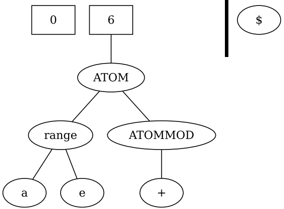
Operation: reduce by rule 9 $ATOM \rightarrow NUCLEUS\, ATOMMOD$

Top of Stack | **Front of Deque**



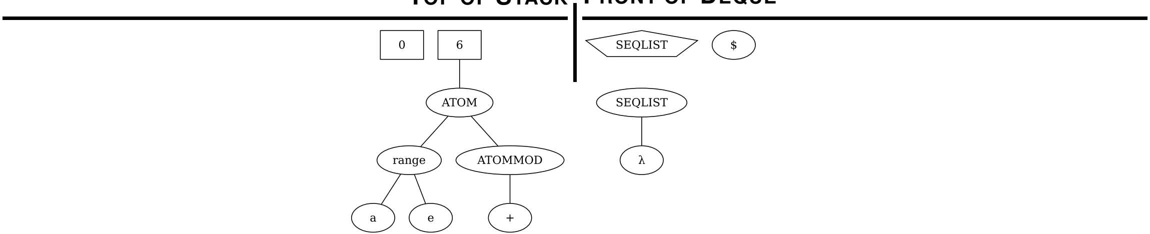
Operation: shift ATOM to stack, goto state 6

TOP OF STACK **FRONT OF DEQUE**



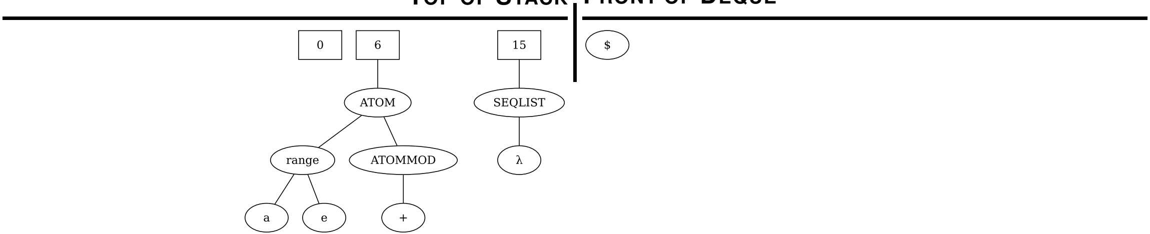
Operation: reduce by rule 8 $SEQLIST \rightarrow \lambda$

TOP OF STACK **FRONT OF DEQUE**



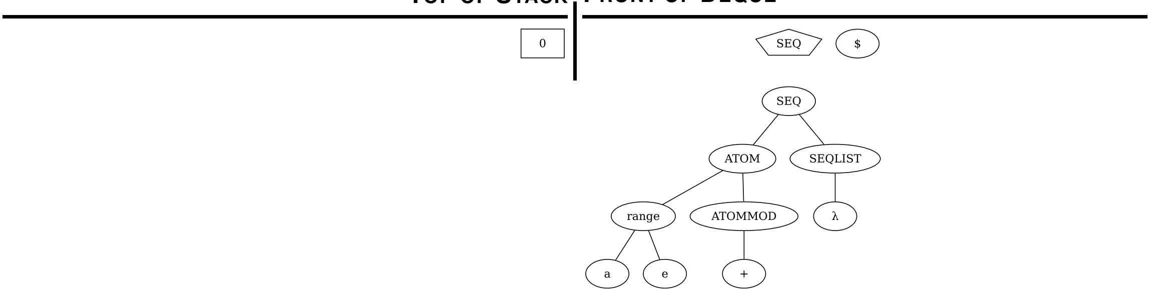
Operation: shift SEQLIST to stack, goto state 15

TOP OF STACK **FRONT OF DEQUE**



Operation: reduce by rule 5 $SEQ \rightarrow ATOM\ SEQLIST$

TOP OF STACK **FRONT OF DEQUE**



fini