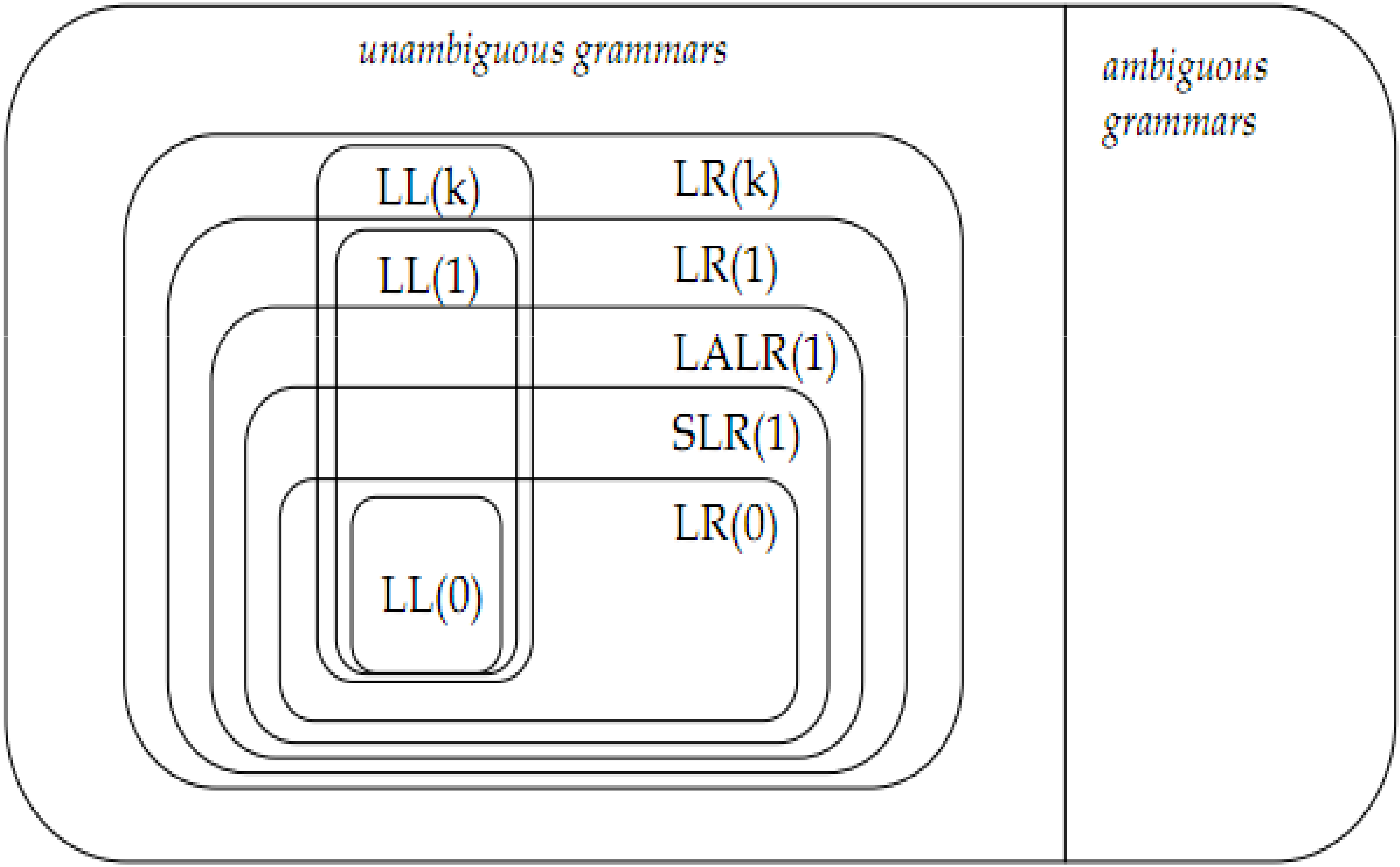
Lecture 10

Syntax Analysis V

Bottom-Up Parsing

Hierarchy of Grammar Class



Viable Prefixes

- α is a viable prefix of a given grammar if:
  - There is a such that αw is a right sentential form
- $\alpha \mid \omega$ is a state of the shift-reduce parser
- As long as the parser has viable prefixes on the stack no parser error has been seen
- The set of viable prefixes is a regular language (not obvious)
- Construct an automaton that accepts viable prefixes

LR(0) Items

- An LR(0) item of a grammar G is a production of G with a special symbol "." at some position of the right side

- • Thus production AXYZ gives four LR(0) items

  A $\rightarrow$ .XYZ

  A $\rightarrow$ X.YZ

  A $\rightarrow$ XY.Z

  A $\rightarrow$ XYZ.

- An item indicates how much of a production has been seen at a point in the process of parsing
  - Symbols on the left of "." are already on the stack **s**
  - Symbols on the right of "." are expected in the input

- The only item for X $\rightarrow$ ε is X $\rightarrow$.

Viable Prefixes and LR(0) Items

- Consider the input: **(int)**

  $E \rightarrow T + E \mid T$

  $T \rightarrow int * T \mid int \mid (E)$

- Then **(E |)** is a state of a shift-reduce parse

- **(E** is a prefix of the rhs of **T → (E)**
  - Will be reduced after the next shift

- Item **T → (E.)** says that so far we have seen **(E** of this production and hope to see **)**

Viable Prefixes and LR(0) Items

- The stack may have many prefixes of rhs's
  - **$\mathbf{Prefix_1}$ $\mathbf{Prefix_2}$ . . . $\mathbf{Prefix_{n-1}}$$\mathbf{Prefix_n}$**

- • Let $Prefix_i$ be a prefix of the rhs of $X_i \rightarrow \alpha_i$
  - **$\mathbf{Prefix_i}$** will eventually reduce to **$\mathbf{X_i}$**
  - The missing part of α_{i-1} starts with X_i
  - i.e. there is a $X_{i-1} \rightarrow Prefix_{i-1}\ X_i\ \beta$ for some β

- Recursively, $Prefix_{k+1} \ldots Prefix_n$ eventually reduces to the missing part of α_k

Viable Prefixes and LR(0) Items

- Consider the string (int * int):
  - (int *|int) is a state of a shift-reduce parse
    - "(" is a prefix of the rhs of T → (E)
    - "ε" is a prefix of the rhs of E T
    - "int *" is a prefix of the rhs of T → int * T

- The "stack of items"
  - T → (.E) says, we have seen "(" of T → (E)
  - E → .T says, we have seen of E → T
  - T → int * .T says, we have seen int * of T int * T

Recognizing Viable Prefixes

- Therefore, we have to build the finite automata that recognizes this sequence of partial rhs's of productions

- Algorithm:

1. Add a dummy production $S' \rightarrow S$ to G
2. The NFA states are the items of G
   - Including the extra production
3. For item $E \rightarrow \alpha.X\beta$ add transition
   - $E \rightarrow \alpha.X\,\beta \rightarrow^{X} E \rightarrow \alpha\, X.\,\beta$
4. For item $E \rightarrow \alpha.X\beta$ and production $X \rightarrow \gamma$ add
   - $E \rightarrow \alpha.X\beta \rightarrow^{\varepsilon} X \rightarrow .\gamma$
5. Every state is an accepting state
6. Start state is $S' \rightarrow .S$

Recognizing Viable Prefixes

- Given the grammar:

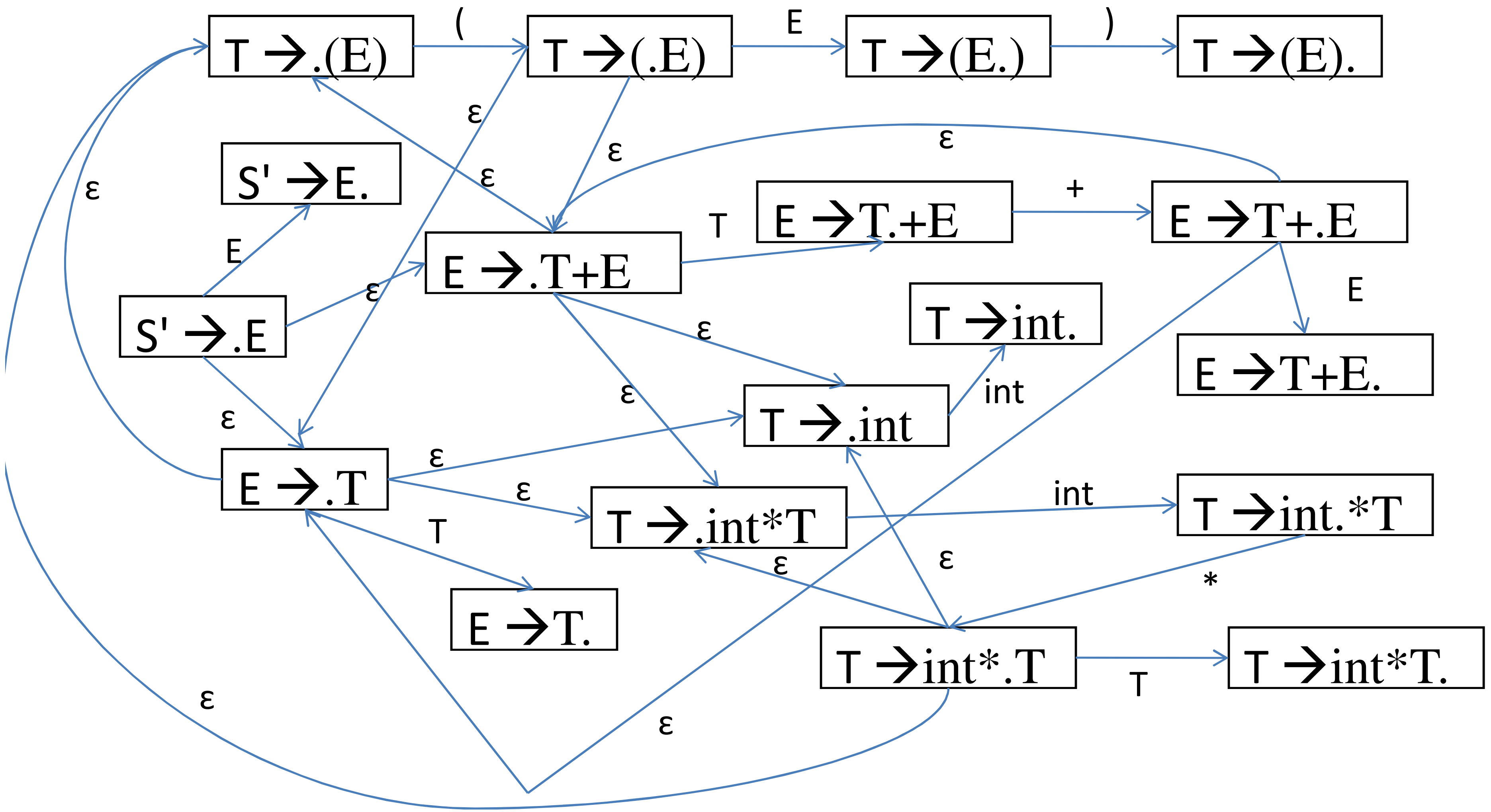
  $S' \rightarrow E$

  $E \rightarrow T + E \mid T$

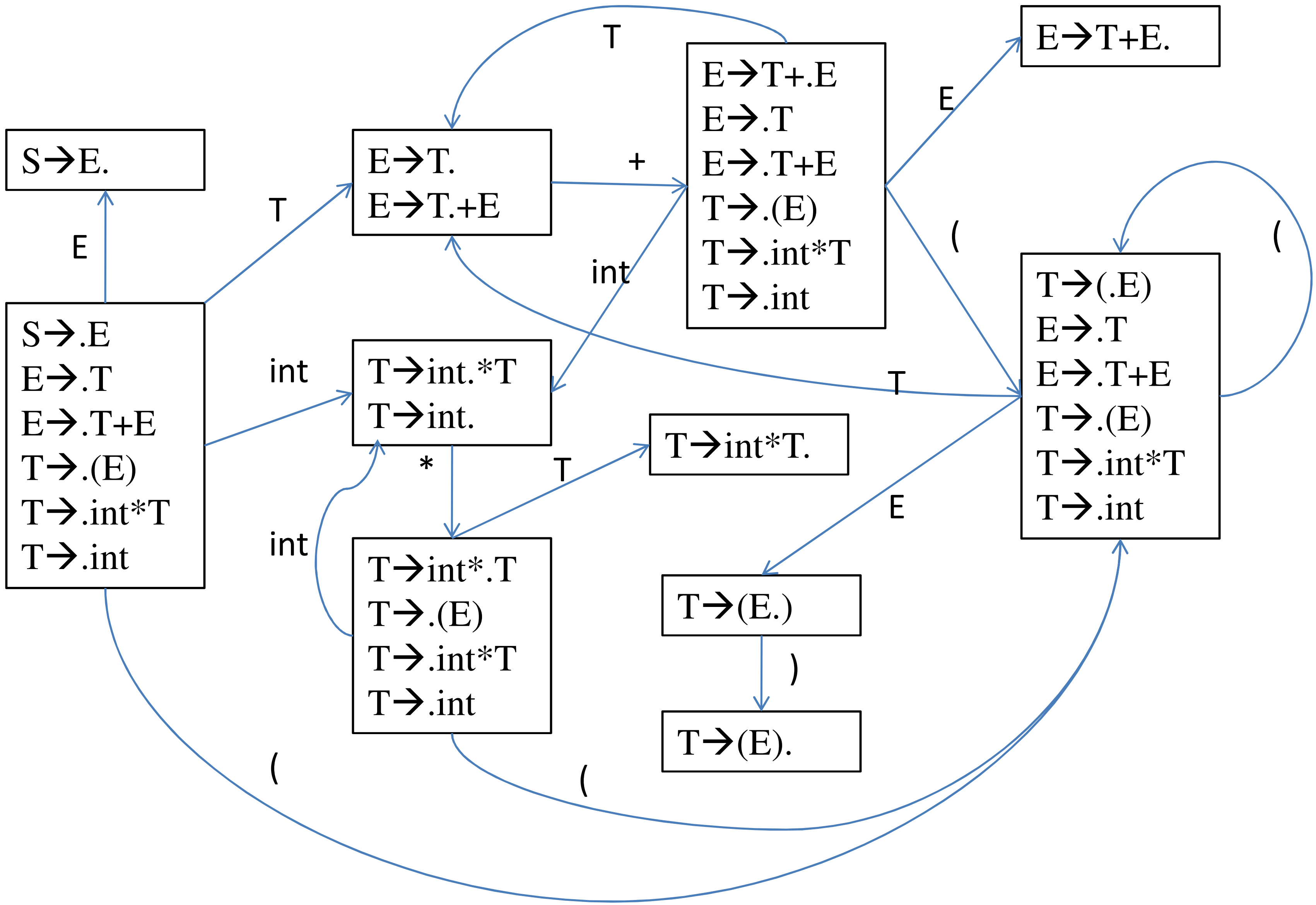
  $T \rightarrow int * T \mid int \mid (E)$

Create an NFA to recognize viable prefixes

Recognizing Viable Prefix



NFA to DFA – Subset Construction



Valid Items

- The states of the DFA are:
  - canonical collections of LR(0) items

- Item $X \rightarrow \beta.\gamma$ is *valid for a viable prefix* $\alpha\beta$ *if*
  - $S' \rightarrow^{*} \alpha X \omega \rightarrow \alpha\beta\gamma\omega$ by a right-most derivation

- After parsing $\alpha\beta$, the valid items are the possible tops of the stack of items

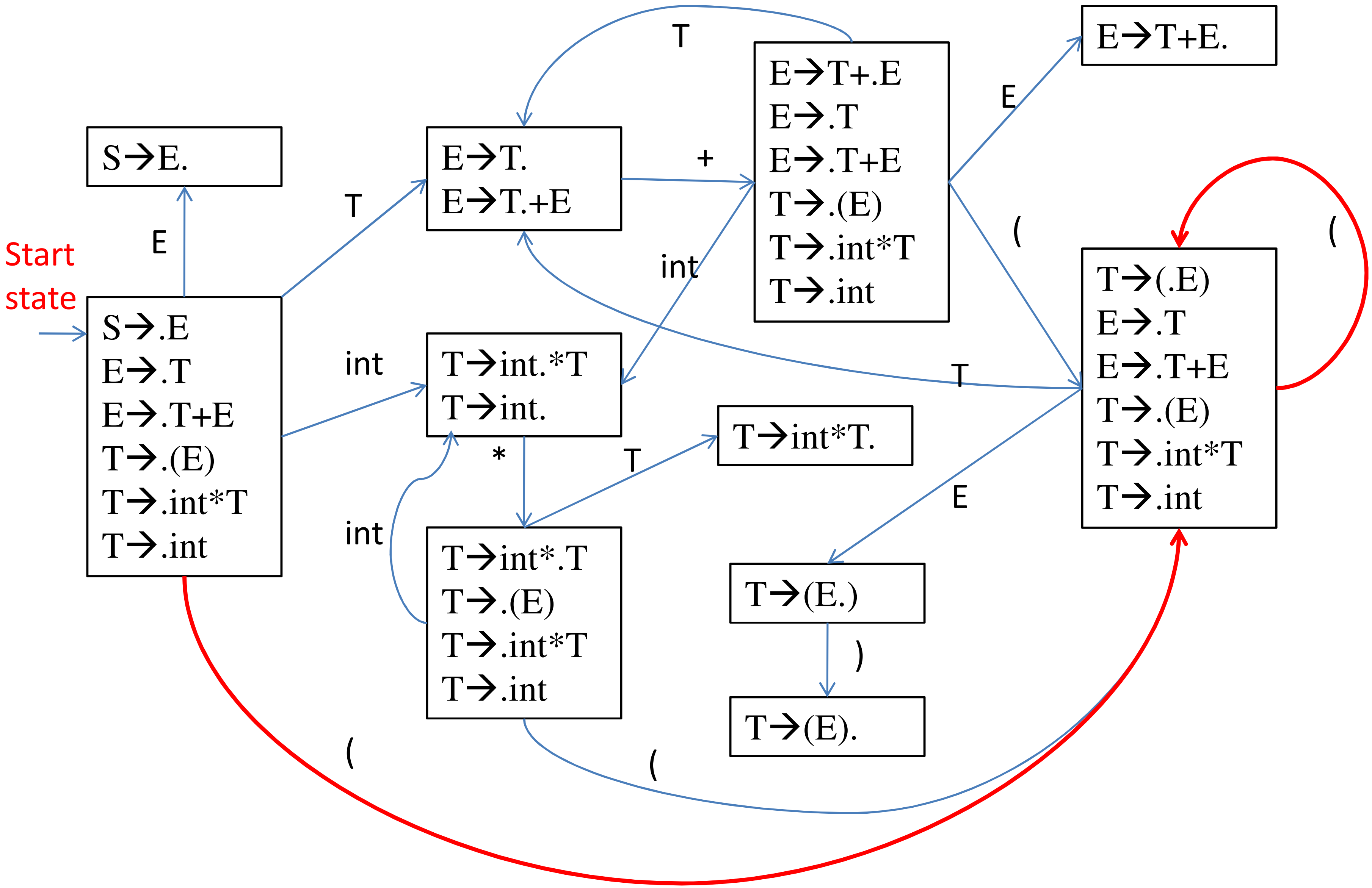
- An item **I** is valid for a viable prefix α if the DFA recognizing viable prefixes terminates on input α in a state S containing I

Valid Items

- The items in S describe what the top of the item stack might be after reading input α

- An item is often valid for many prefixes
  - Example: The item $T \rightarrow (.E)$ is valid for prefixes
    - (, ((, (((, ((((, . . .

NFA to DFA – Subset Construction



LR(0) Parsing

- Assume
  - stack contains α
  - next input token is t
  - DFA on input α terminates in state s

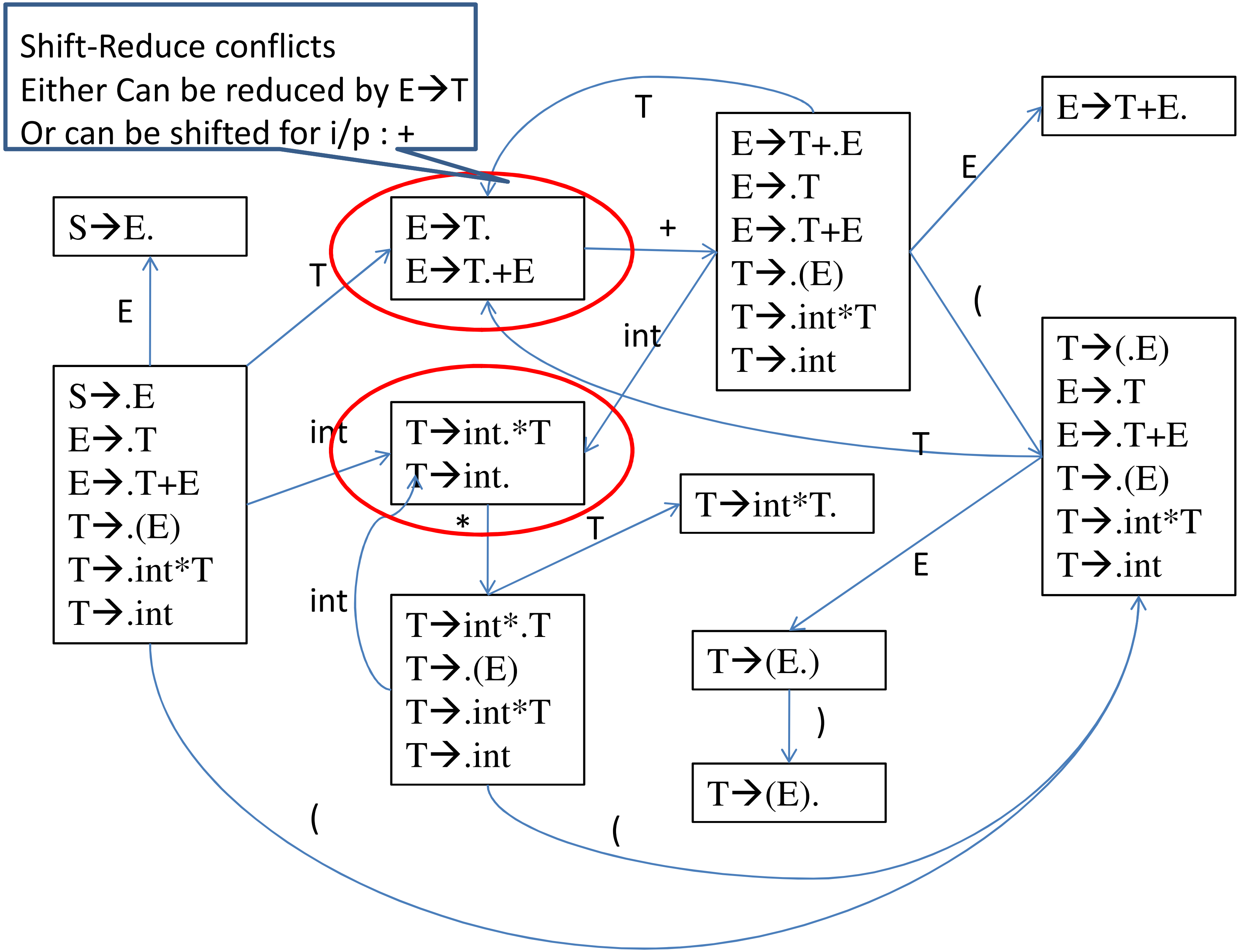
- Reduce by $X \rightarrow \beta$ if
  - s contains item $X \rightarrow \beta.$

- Shift if
  - s contains item $X \rightarrow \beta.t\omega$
  - Equivalent to saying s has a transition labeled t

LR(0) Parsing - Conflicts

- LR(0) has a reduce/reduce conflict if:
  - Any state has two reduce items:
  - $X \rightarrow \beta.$ and $Y \rightarrow \omega.$
- LR(0) has a shift/reduce conflict if:
  - Any state has a reduce item and a shift item:
  - $X \rightarrow \beta.$ **and** $Y \rightarrow \omega.t\delta$
- SLR improves on LR(0) shift/reduce heuristics
  - Fewer states have conflicts

Shift Reduce Conflicts



Countermeasure

- SLR = Simple LR
- Improve on LR(0) shift/reduce heuristics
  - Fewer states have conflicts

SLR Parsing

- Assume
  - stack contains α
  - next input token is **t**
  - DFA on input α terminates in state **s**

- Reduce by $X \rightarrow \beta$ if
  - s contains item $X \rightarrow \beta.$
  - $t \in$ Follow (X)

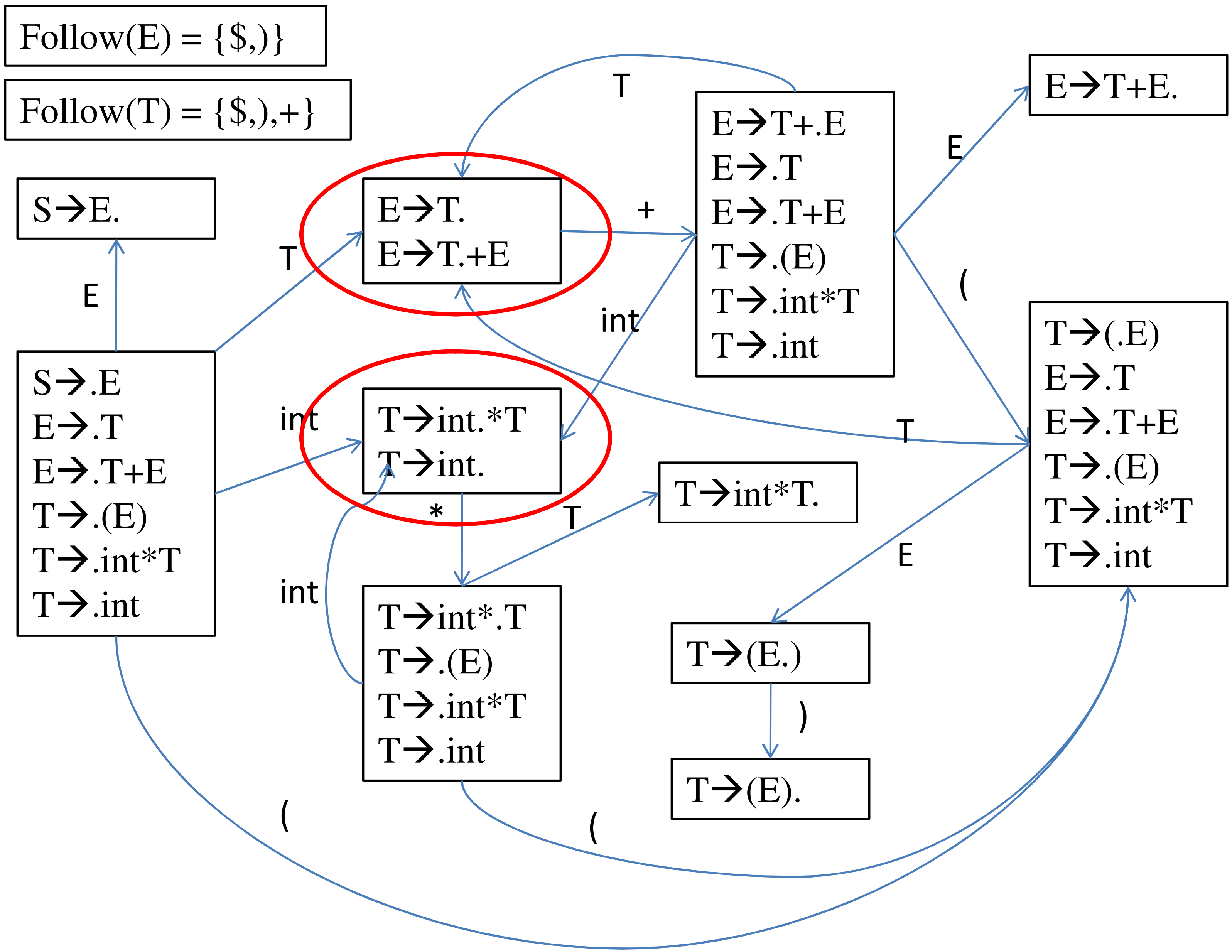
- Shift if
  - s contains item $X \rightarrow \beta.t\omega$
  - Equivalent to saying s has a transition labeled t

SLR Parsing

- If there are conflicts under these rules, the grammar is not SLR

- The rules amount to a heuristic for detecting handle
  - The SLR grammars are those where the heuristics detect exactly the handle

Shift Reduce Conflicts



SLR Parsing Algorithm

1. Let M be DFA for viable prefixes of G
2. Let $|x_1 \ldots x_n\$$ be initial configuration
3. Repeat until configuration is **S|$**
   1. Let $\alpha|\omega$ be current configuration
   2. Run M on current stack α
   3. If M rejects α, report parsing error
      1. Stack α is not a viable prefix
   4. If M accepts α with items I, let u be next input
      1. Shift if $X \rightarrow \beta.u\gamma \in I$
      2. Reduce if $X \rightarrow \beta. \in I$ and $u \in \text{Follow}(X)$
      3. Report parsing error if neither applies

SLR Parsing - Improvements

- Note that Step 3.3 is redundant
- If there is a conflict in the last step, grammar is not SLR(k)

- Lots of grammars are not SLR
  - Including all ambiguous grammars

- We can parse more grammars by using precedence declarations
  - Instructions for resolving conflicts

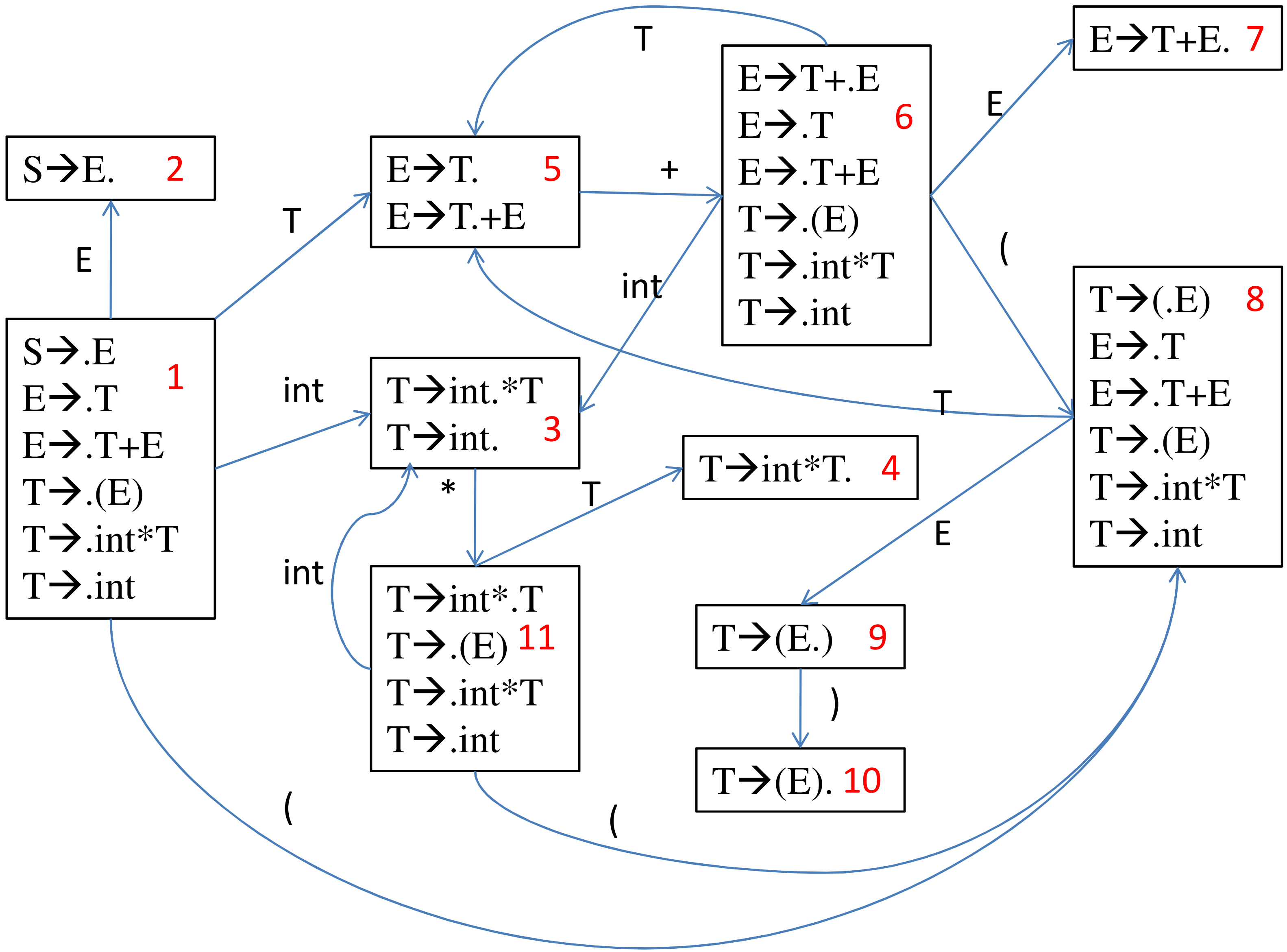
SLR Parsing

- Consider the ambiguous grammar:
  - **E → E + E | E * E | (E) | int**

- The DFA for this grammar contains a state with the following items:
  - **E → E * E. and E → E. + E**
  - There is a shift/reduce conflict

- Declaring "* has higher precedence than +" resolves this conflict in favor of reducing

SLR Parsing Example



SLR Parsing Example

- Parse the token stream: int * int$

Configuration	*DFA Halt State*	*Action*
|int * int$	1	Shift
int | * int$	3 * $\notin$ Follow(T)	Shift
int * | int$	11	Shift
int * int |$	3 $ $\in$ Follow(T)	Reduce. T → int
int * T |$	4 $ $\in$ Follow(T)	Reduce. T → int * T
T |$	5 $ $\in$ Follow(T)	Reduce. E → T
E |$	Accept	

To be continued...