

Reconocimiento Ascendente Bottom Up Parsing



Bottom Up Parsing...

- Son más generales que (deterministic) top down parsing
 - Así como también eficientes
 - Se construye basado en las ideas del top down parsing
- Bottom Up Parsing es el método preferido

Bottom Up Parsing...

- No necesitan gramáticas left-factored
- Volvemos a la gramática original

$$E \rightarrow T + E \mid T$$

$$T \rightarrow \text{int} * T \mid \text{int} \mid (E)$$

- Considere el string

int * int + int

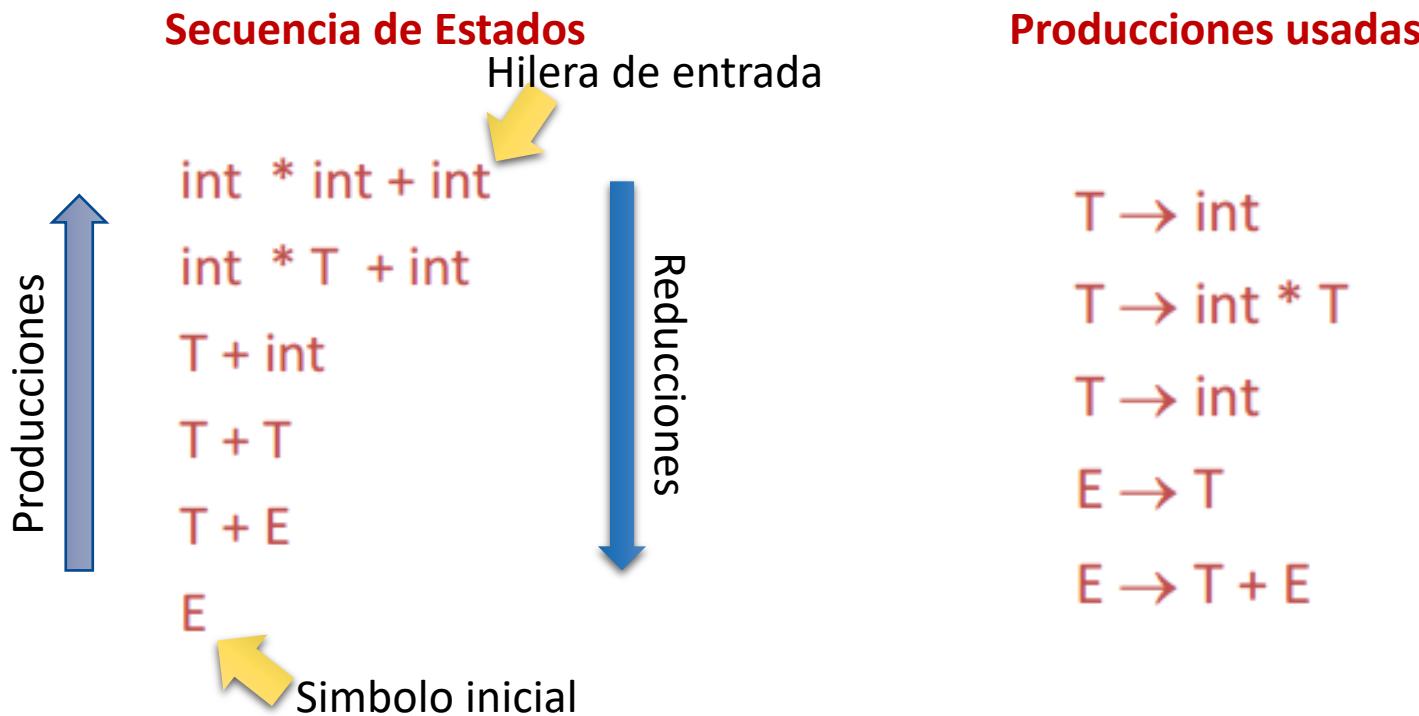
Bottom Up Parsing...

$$E \rightarrow T + E \mid T$$

$$T \rightarrow \text{int} * T \mid \text{int} \mid (E)$$

int * int + int

- Bottom Up parsing **reduce** una hilera al símbolo inicial por medio de producciones “invertidas”



Bottom Up Parsing...

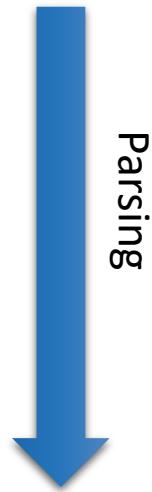
$$E \rightarrow T + E \mid T$$

$$T \rightarrow \text{int} * T \mid \text{int} \mid (E)$$

int * int + int

- Tener en cuenta que las producciones, leen al revés, siguiendo una derivación por la derecha

int * int + int
int * T + int
T + int
T + T
T + E
E



T → int
T → int * T
T → int
E → T
E → T + E

Bottom Up Parsing...

Aspecto Importante:

Un Bottom Up Parser traza una derivación
por la derecha a la inversa

Bottom Up Parsing...

int * int + int

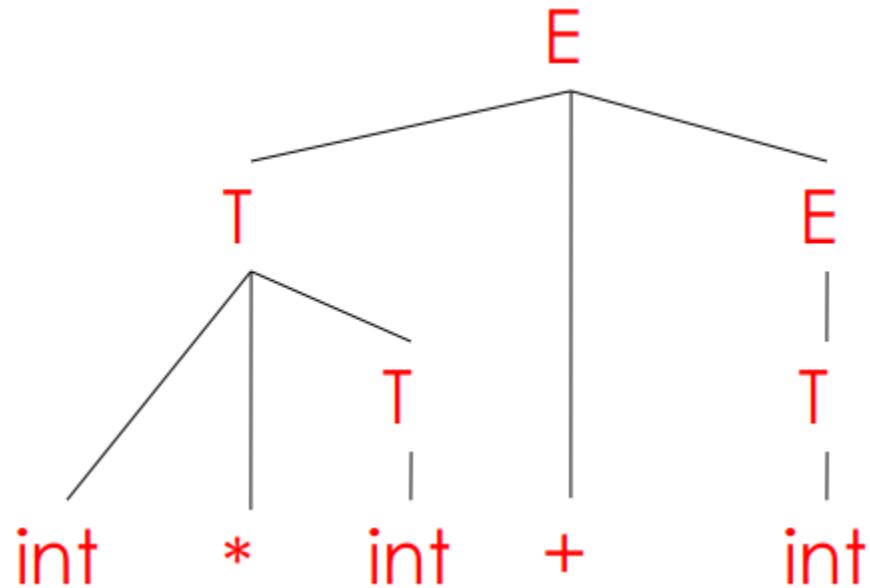
int * T + int

T + int

T + T

T + E

E



Bottom Up Parsing...

int * int + int

int * int + int

Bottom Up Parsing...

int * int + int

int * T + int

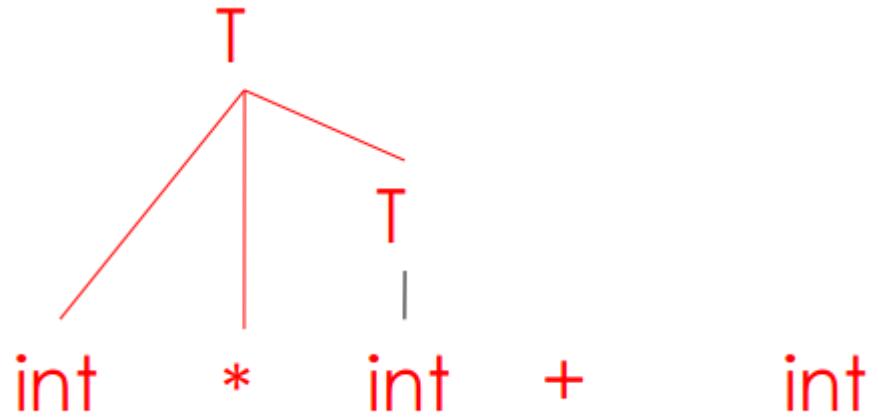
T
|
int * int + int

Bottom Up Parsing...

int * int + int

int * T + int

T + int



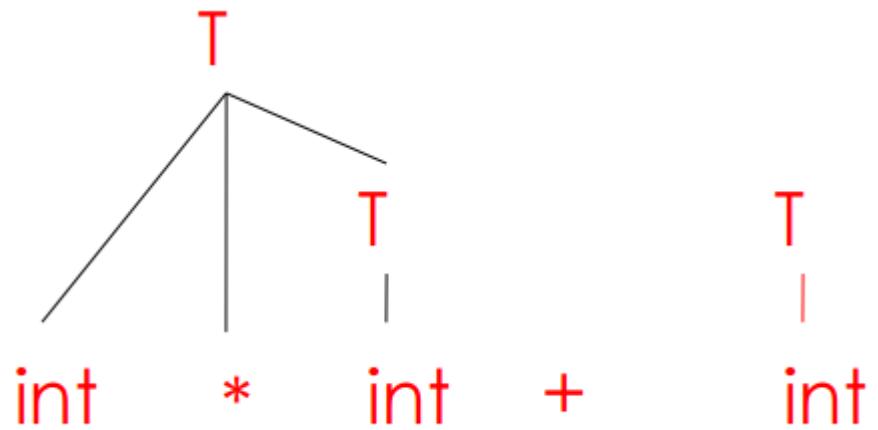
Bottom Up Parsing...

int * int + int

int * T + int

T + int

T + T



Bottom Up Parsing...

int * int + int

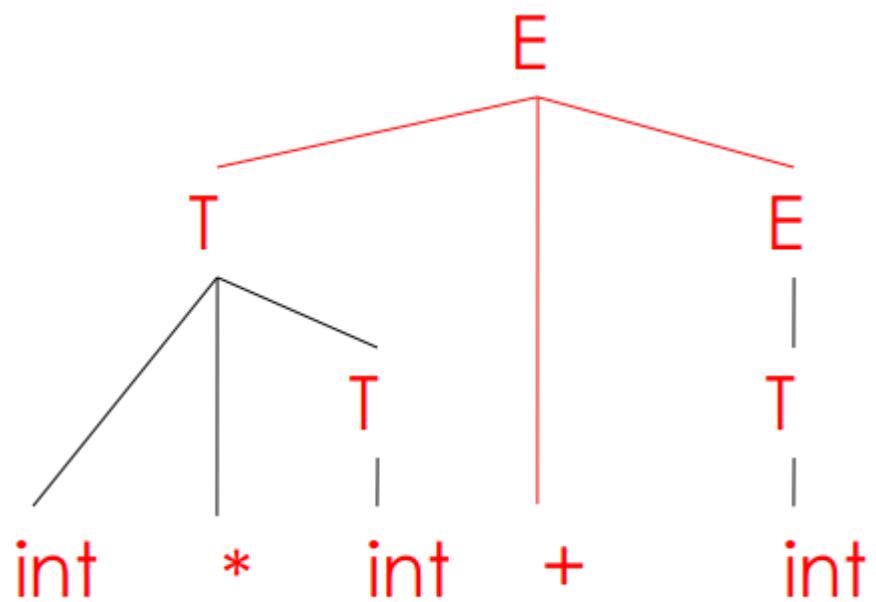
int * T + int

T + int

T + T

T + E

E



Shift-Reduce Parsing

Bottom Up Parsing...

Consecuencias

- Sea $\alpha\beta\omega$ un paso de un parseo bottom up
- Se asume que la siguiente reducción es dada por $X \rightarrow \beta$
- Entonces ω es una hilera de terminales

Así entonces, $\alpha X \omega \rightarrow \alpha \beta \omega$ es un paso en la derivación más hacia la derecha (right most).

Bottom Up Parsing...

- Idea: Separar una cadena en dos sub Hileras
 - Subhilera derecha aún no examinada por los análisis
 - Subhilera izquierda tiene terminales y no terminales
 - El punto de división está marcada por una |

Bottom Up Parsing...

- Bottom Up Parsing usa únicamente dos tipos de acciones

Shift

Reduce

Bottom Up Parsing...

Shift

- Mueve | un lugar a la derecha
 - Desplaza (Shift) un terminal a la cadena de izquierda

$$\text{ABC|xyz} \Rightarrow \text{ABCx|yz}$$

Bottom Up Parsing...

Reduce

- Aplicar una producción inversa en el extremo derecho de la cadena izquierda
 - Si $A \rightarrow xy$ es una producción, entonces

$$Cbxy|ijk \Rightarrow CbA|ijk$$

Bottom Up Parsing...

Ejemplo de movimientos Reduce

int * int | + int
int * T | + int

reduce $T \rightarrow \text{int}$
reduce $T \rightarrow \text{int} * T$

T + int |
T + T |
T + E |
E |

reduce $T \rightarrow \text{int}$
reduce $E \rightarrow T$
reduce $E \rightarrow T + E$

Bottom Up Parsing...

Ejemplo de movimientos Reduce y Shift

int * int + int	shift
int * int + int	shift
int * int + int	shift
int * int + int	reduce T → int
int * T + int	reduce T → int * T
T + int	shift
T + int	shift
T + int	reduce T → int
T + T	reduce E → T
T + E	reduce E → T + E
E	

Bottom Up Parsing...

| int * int + int

↑ int * int + int

Bottom Up Parsing...

| int * int + int

int | * int + int

int * int + int



Bottom Up Parsing...

| int * int + int

int | * int + int

int * | int + int

int * int + int
 ↑

Bottom Up Parsing...

| int * int + int

int | * int + int

int * | int + int

int * int | + int

int * int + int
 ↑

Bottom Up Parsing...

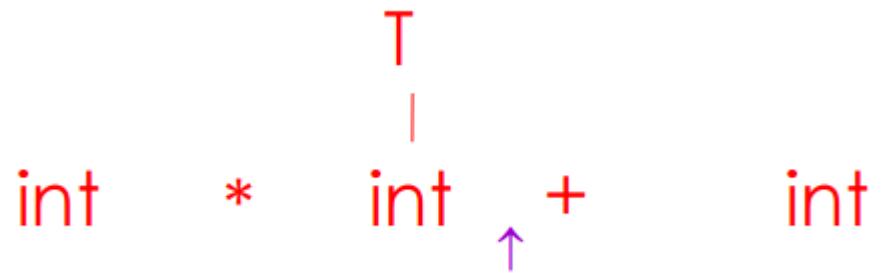
| int * int + int

int | * int + int

int * | int + int

int * int | + int

int * T | + int



Bottom Up Parsing...

| int * int + int

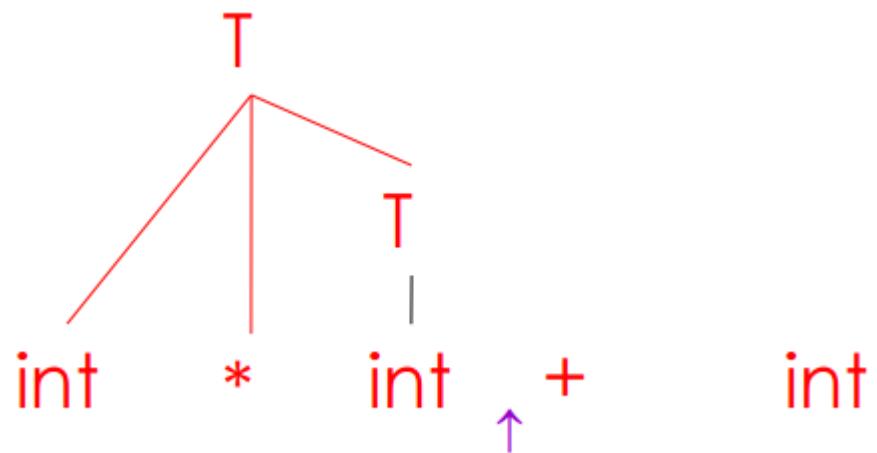
int | * int + int

int * | int + int

int * int | + int

int * T | + int

T | + int



Bottom Up Parsing...

| int * int + int

int | * int + int

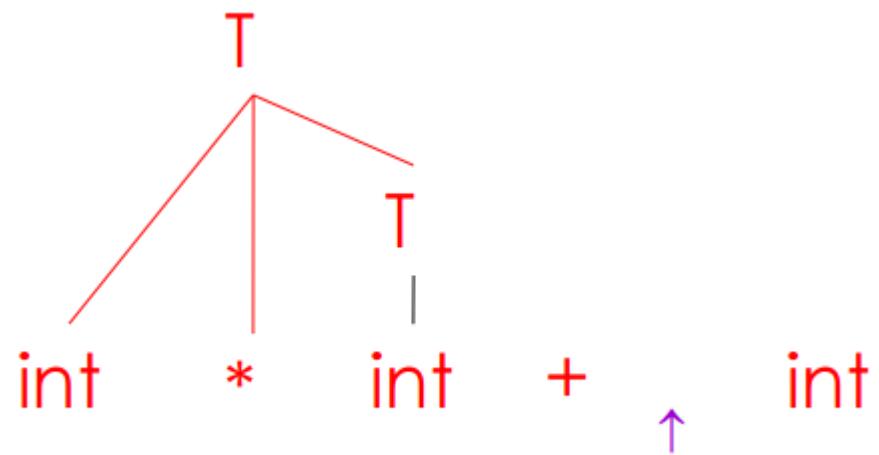
int * | int + int

int * int | + int

int * T | + int

T | + int

T + | int



Bottom Up Parsing...

| int * int + int

int | * int + int

int * | int + int

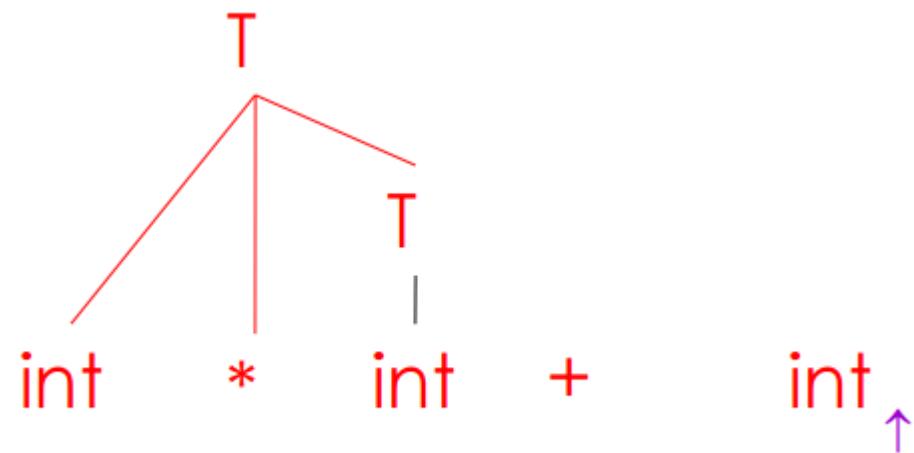
int * int | + int

int * T | + int

T | + int

T + | int

T + int |



Bottom Up Parsing...

| int * int + int

int | * int + int

int * | int + int

int * int | + int

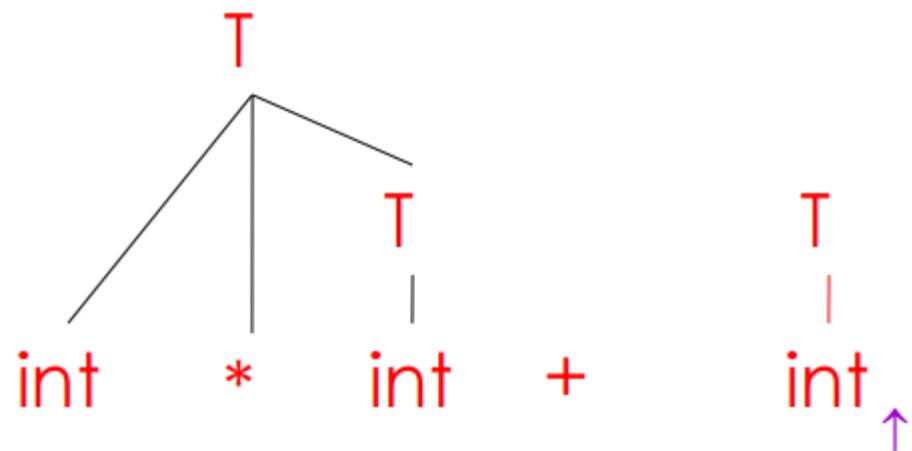
int * T | + int

T | + int

T + | int

T + int |

T + T |



Bottom Up Parsing...

| int * int + int

int | * int + int

int * | int + int

int * int | + int

int * T | + int

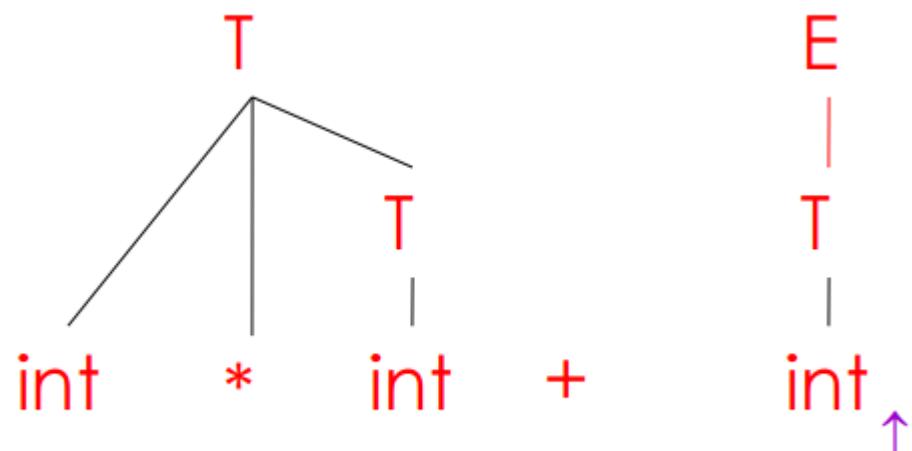
T | + int

T + | int

T + int |

T + T |

T + E |



Bottom Up Parsing...

| int * int + int

int | * int + int

int * | int + int

int * int | + int

int * T | + int

T | + int

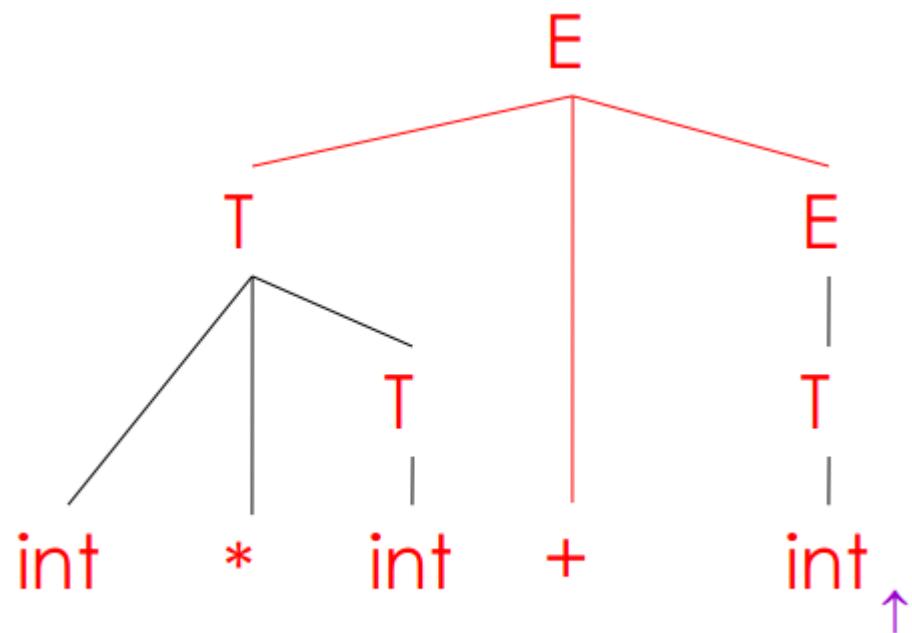
T + | int

T + int |

T + T |

T + E |

E |



Bottom Up Parsing...

| int * int + int

int | * int + int

int * | int + int

int * int | + int

int * T | + int

T | + int

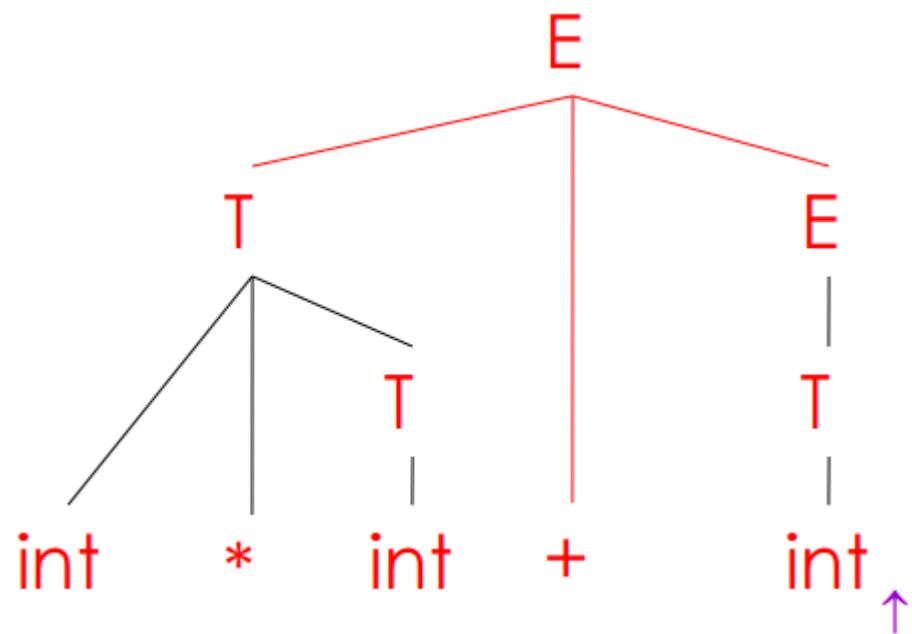
T + | int

T + int |

T + T |

T + E |

E |



Bottom Up Parsing...

- La hilera de la izquierda puede ser implementada por medio de una pila
 - La parte superior de la pila es el |
- Shift empuja un terminal en la pila
- Reduce
 - Símbolos Pops fuera de la pila (producción rhs)
 - Empuje un No Terminal sobre la pila (producción lhs)

Bottom Up Parsing...

- En un estado dado, más de una acción (shift o reduce) puede dar lugar a un análisis sintáctico válido
- Si es legal reducir en dos producciones diferentes, hay un conflicto shift-reduce

Bottom Up Parsing...

- Ejercicio

Dada la siguiente gramática:

$S \rightarrow A B$
 $A \rightarrow A a \mid a$
 $B \rightarrow B b \mid b$
 $S' \rightarrow S$

Valide si la hilera “aaabb” es correcta según dicha gramática.

Pila	Entrada	Acción
\$	aaabb\$	Shift

Bottom Up Parsing...

- Ejercicio - Respuesta

Pila	Entrada	Acción
\$	aaabb\$	Shift
a\$	aabb\$	Reduce A -> a
A\$	aabb\$	Shift
Aa\$	abb\$	Reduce A -> Aa
A\$	abb\$	Shift
Aa\$	bb\$	Reduce A -> Aa
A\$	bb\$	Shift
Ab\$	b\$	Reduce B -> b
AB\$	b\$	Shift
ABb\$	\$	Reduce B -> Bb
AB\$	\$	Reduce S -> AB
S\$	\$	Reduce S' -> S
S'\$	\$	Aceptada

Bottom Up Parsing...

- Ejercicio 2

Dada la siguiente gramática:

$S \rightarrow (S)$

$S' \rightarrow S$

$S \rightarrow ID$

Valide si la hilera “((ID))” es correcta según dicha gramática.

Pila	Entrada	Accion
\$	((ID))\$	Shift

Bottom Up Parsing...

- Ejercicio 2 - Solucion

Pila	Entrada	Accion
\$	((ID))\$	Shift
(\$	(ID))\$	Shift
((\$	ID))\$	Shift
((ID\$))\$	Reduce S -> ID
((S\$))\$	Shift
((S)\$)\$	Reduce S -> (S)
(S\$)\$	Shift
(S)\$	\$	Reduce S -> (S)
S\$	\$	Reduce S' -> S
S'\$	\$	Aceptada

Bottom Up Parsing...

- Ejercicio 3

Dada la gramática:

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$E \rightarrow ID$$

Valide si la hilera “ID * ID + ID” es correcta según dicha gramática.

Bottom Up Parsing...

• Ejercicio 3 – Solucion

Parseo Shift-Reduce No.1

Pila	Entrada	Accion
\$	ID*ID+ID\$	Shift
ID\$	*ID+ID\$	Reduce E -> ID
E\$	*ID+ID\$	Shift
E*\$	ID+ID\$	Shift
E*ID\$	+ID\$	Reduce E -> ID
E*E\$	+ID\$	Reduce E -> E * E
E\$	+ID\$	Shift
E+\$	ID\$	Shift
E+ID\$	\$	Reduce E -> ID
E+E\$	\$	E -> E + E
E\$	\$	Aceptada

Parseo Shift-Reduce No.2

Pila	Entrada	Accion
\$	ID*ID+ID\$	Shift
ID\$	*ID+ID\$	Reduce E -> ID
E\$	*ID+ID\$	Shift
E*\$	ID+ID\$	Shift
E*ID\$	+ID\$	Reduce E -> ID
E*E\$	+ID\$	Shift
E*E+\$	ID\$	Shift
E*E+ID\$	\$	Reduce E -> ID
E*E+E\$	\$	Reduce E -> E + E
E*E\$	\$	Reduce E -> E * E
E\$	\$	Aceptada

Handles

Bottom Up Parsing...

- Bottom Up parsing usa dos acciones

Shift

solo lee un token de entrada y mueve la barra vertical uno hacia la derecha.

$$ABC|xyz \Rightarrow ABCx|yz$$

Reduce

reemplaza el lado derecho de una producción a la izquierda de la barra vertical por una producción del lado izquierdo.

$$Cbxy|ijk \Rightarrow CbA|ijk$$

Bottom Up Parsing...

- La hilera de la izquierda puede ser implementada por medio de una pila
 - La parte superior de la pila es el |
- Shift empuja un terminal en la pila
- Reduce
 - Símbolos Pops fuera de la pila (producción rhs)
 - Empuje un No Terminal sobre la pila (producción lhs)

Bottom Up Parsing...

- Cómo decidir cuando usar Shift o Reduce?
- Ejemplo de gramática

$E \rightarrow T + E \mid T$

$T \rightarrow \text{int} * T \mid \text{int} \mid (E)$

- Considere el paso $\text{int} \mid * \text{int} + \text{int}$
 - Se puede reducir por $T \rightarrow \text{int}$ dando $T \mid * \text{int} + \text{int}$
 - Un error fatal !!!
 - No hay manera de reducir el símbolo inicial E

Bottom Up Parsing...

- Intuición: Se desea reducir solamente si el resultado puede aún ser reducido hasta el símbolo inicial.
- Asuma la derivación más a la derecha (rightmost)

$$S \xrightarrow{*} \alpha X \omega \rightarrow \alpha \beta \omega$$

←
Reducción

- Entonces $\alpha\beta$ es un handle de $\alpha\beta\omega$
- Los handlers vienen a formalizar la intuición
 - Un Handler es una reducción que permite además reducciones de regreso al símbolo inicial.

Bottom Up Parsing...

Aspecto Importante:

En el parser shift-reduce, handlers sólo aparecen en la parte superior de la pila, nunca dentro

Bottom Up Parsing...

- Inducción informal sobre # de movimientos reduce
- Inicialmente verdadero, pila se encuentra vacía
- Inmediatamente después de la reducción de un handler
 - NoTerminal más a la derecha sobre la cima de la pila
 - Siguiente handler debe estar a la derecha del NoTerminal de más a la derecha, debido a que es una derivación rightmost.
 - Secuencia de movimientos Shift hasta alcanzar al siguiente handler

Bottom Up Parsing...

- En el parseo shift-reduce, handlers siempre aparecen en la cima de la pila.
- Handlers nunca se encuentran a la izquierda de un NoTerminal más a la derecha
 - Por tanto, movimientos shift-reduce son suficientes; el | nunca necesita moverse a la izquierda
- Algoritmos de bottom up se encuentran basados en el reconocimiento de handlers.

Bottom Up Parsing...

- Malas Noticias
 - No hay algoritmos conocidos que sean eficientes para identificar Handles
- Buenas Noticias
 - Existen buenos heurísticos para adivinar handles
 - En algunas gramáticas libres de contexto, los heurísticos siempre adivinan de forma correcta.

Bottom Up Parsing...

- No es obvio la forma de detectar handles
- En cada paso, el parseador observa únicamente la pila, no la entrada completa; inicia con esto ...

- α es un prefijo viable si hay un ω tal que $\alpha|\omega$ es un estado de un parseador shift-reduce.
-
- pila Resto de la entrada
↓ ↓
 $\alpha | \omega$

Bottom Up Parsing...

- Un prefijo viable no se extiende más allá del extremo derecho del handle
- Es un prefijo variable debido a que es un prefijo del handle.
- Mientras un parseador tiene prefijos viables en la pila ningún error de parseo se ha detectado.

Bottom Up Parsing...

Aspecto Importante:

Para cualquier gramática, un conjunto de prefijos viables es un lenguaje regular.

Bottom Up Parsing...

- Un ítem es una producción con un “.” en algún lado del rhs (right hand side).
- Los ítems para $T \rightarrow (E)$ son
 - $T \rightarrow .(E)$
 - $T \rightarrow (.E)$
 - $T \rightarrow (E.)$
 - $T \rightarrow (E).$

Bottom Up Parsing...

- El único ítem para $X \rightarrow \varepsilon$ es $X \rightarrow .$
- Los ítems son frecuentemente llamados “ítems LR(0)”

Bottom Up Parsing...

- Considere la entrada (int)

$$\begin{aligned} E &\rightarrow T + E \mid T \\ T &\rightarrow \text{int} * T \mid \text{int} \mid (E) \end{aligned}$$

- Entonces (E|) es un estado de un parser shift-reduce
 - (E es un prefijo del rhs de $T \rightarrow (E)$
 - Será reducido luego del siguiente shift
 - El ítem $T \rightarrow (E.)$ indica qué tan largo se ha visto (E de esta producción y la esperanza de ver)

Bottom Up Parsing...

- La pila puede tener muchos prefijos de rhs's

Prefix₁ Prefix₂ . . . Prefix_{n-1} Prefix_n

- Sea Prefix_i un prefijo de rhs de $X_i \rightarrow \alpha_i$
 - Prefix_i finalmente reducirá X_i
 - La parte perdida de α_{i-1} inicia con X_i
 - Esto es, hay un $X_{i-1} \rightarrow \text{Prefix}_{i-1} X_i \beta$ para algún β
- De forma recursiva, Prefix_{k+1}...Prefix_n finalmente reducirá a la parte perdida α_k

Bottom Up Parsing...

- Considere el string $(\text{int} * \text{int})$:
 - $(\text{int} * | \text{int})$ es un estado de un parser shift-reduce
- “ $($ ” es un prefijo del rhs de $T \rightarrow (E)$
- “ ϵ ” es un prefijo del rhs de $E \rightarrow T$
- “ $\text{int} *$ ” es un prefijo del rhs de $T \rightarrow \text{int} * T$

Bottom Up Parsing...

- La “pila de ítems”

$T \rightarrow (.E)$

$E \rightarrow .T$

$T \rightarrow \text{int} * .T$

- Dice que

- Ha visto “ $(.$ ” de $T \rightarrow (E)$
- Ha visto ϵ de $E \rightarrow T$
- Ha visto $\text{int} *$ de $T \rightarrow \text{int} * T$

Bottom Up Parsing...

- Idea: Reconocer prefijos viables
 - Reconocer una secuencia de rhs's parciales de producciones, donde
 - Cada rhs parcial puede finalmente reducir a la parte del sufijo perdido de su predecesor.

Bottom Up Parsing...

- “Analizador Sintáctico Ascendente por desplazamiento/reducción”
- Usa una pila y un puntero a la cadena de entrada
- Shift (Desplazar): pone un símbolo de la entrada en el tope de la pila
- Reduce(Reducir) : desapila algunos símbolos que son el cuerpo de una producción y los reemplaza por la cabeza

Bottom Up Parsing...

- “Handle” de una cadena es una **subcadena** que coincide con el lado derecho de una producción, y cuya **reducción** al no terminal de la izquierda representa un paso en la derivación por la derecha (en forma inversa) .

Si

$$S^* \Rightarrow \alpha Aw^* \Rightarrow \alpha\beta w, \text{ entonces } A \rightarrow \beta$$

es un handle de $\alpha\beta w$ (la cadena w contiene símbolos terminales solamente)

Bottom Up Parsing...

- “Ítems LR(0)” de una gramática G, es una regla de producción de G, con un punto en alguna posición del lado derecho
- Por ejemplo, $E \rightarrow E + T$, genera los ítems:

$$E \rightarrow \bullet E + T$$
$$E \rightarrow E \bullet + T$$
$$E \rightarrow E + \bullet T$$
$$E \rightarrow E + T \bullet$$

$A \rightarrow \epsilon$, genera el ítem: $A \rightarrow \bullet$

Bottom Up Parsing...

- Un ítem no determina el estado de un parser
- Para determinar el estado de un parser se consideran conjuntos de ítems
- Los conjuntos de ítems serán los estados del autómata determinista LR(0)

Bottom Up Parsing...

- “Autómata LR(0)”
- Los estados son los elementos de la colección canónica LR(0)
- Las transiciones son los valores de la función GoTo
- Estado inicial: $\{[S' \rightarrow S]\}$
- Todos los estados son de aceptación
- Este autómata ayuda a decidir si se debe hacer shift o reduce cuando ambas opciones son posibles

Bottom Up Parsing...

- “Gramática Aumentada”
- Si G es una gramática con símbolo inicial S , entonces, G' - la gramática G aumentada- es G con un nuevo símbolo de inicio S' , tal que $S' \rightarrow S$
- Este nuevo símbolo (S') se crea para indicar al analizador que detenga el proceso y acepte la entrada (la cadena se acepta cuando se hace el reduce por $S' \rightarrow S$)

Bottom Up Parsing...

- “Prefijo Viable”
- Son todos los prefijos de una forma sentencial que no van más allá del pivote
- El pivote es la subcadena que vamos a tomar para reducir y que forma parte del camino correcto.
- Prefijos de una forma sentencial derecha que pueden aparecer en la pila de un parser shift-reduce, porque no sobrepasan el handle ubicado mas a la derecha.

Bottom Up Parsing...

- “Prefijo Viable”
- La base de la construcción de las tablas de decisión LR es encontrar todos los prefijos viables de cualquier subcadena del lenguaje
- Si la pila contiene un prefijo viable, se puede extender
 - Hacer shift, si mantiene un prefijo viable.
 - Sino: reduce, si hay un handle en el tope
 - Sino: error

Bottom Up Parsing...

Reconociendo Prefijos viables

Bottom Up Parsing...

1. Agregue una producción $S' \rightarrow S$ a G
2. Los estados del NFA (*que reconoce los prefijos viables*) son los ítems de G
 - Incluye la producción extra
3. Para el ítem $E \rightarrow \alpha.X\beta$ agregue la transición
$$E \rightarrow \alpha.X\beta \xrightarrow{X} E \rightarrow \alpha X.\beta$$
4. Para el ítem $E \rightarrow \alpha.X\beta$ y producción $X \rightarrow \gamma$ agregue
$$E \rightarrow \alpha.X\beta \xrightarrow{\epsilon} X \rightarrow .\gamma$$

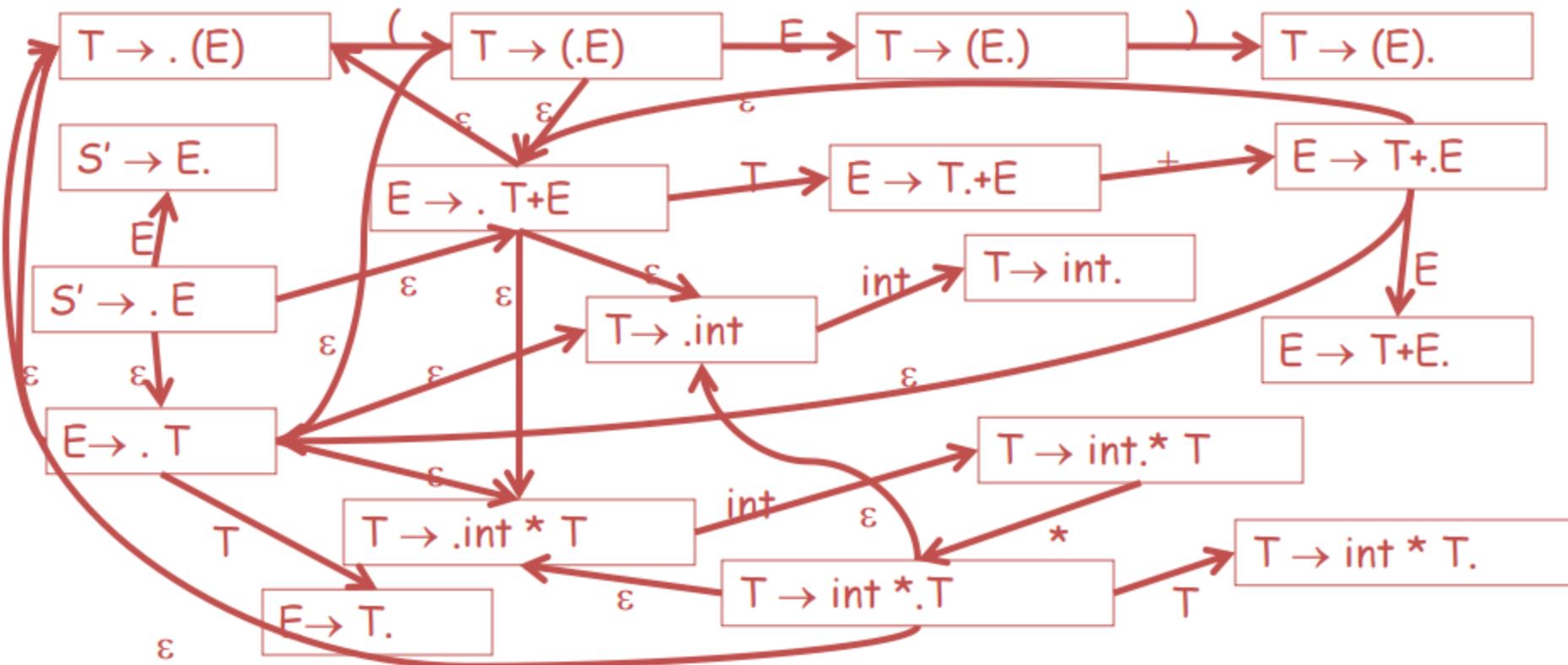
Bottom Up Parsing...

5. Cada estado es un estado de aceptación
6. El estado inicial es $S' \rightarrow .S$

Bottom Up Parsing...

$$\begin{aligned} S' &\rightarrow E \\ E &\rightarrow T + E \mid T \\ T &\rightarrow \text{int} * T \mid \text{int} \mid (E) \end{aligned}$$

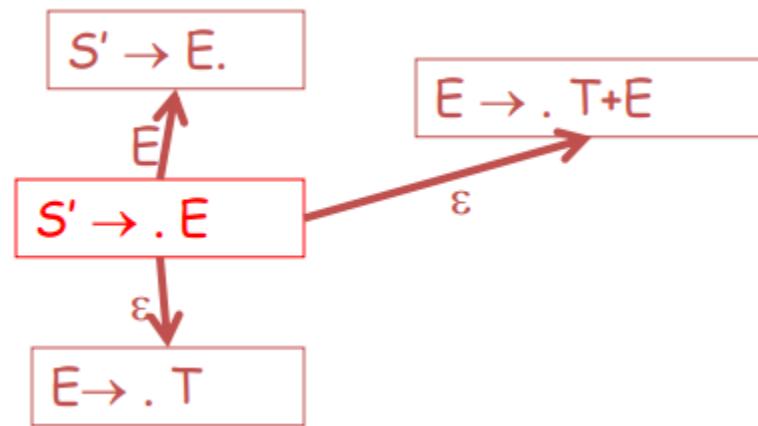
Bottom Up Parsing...



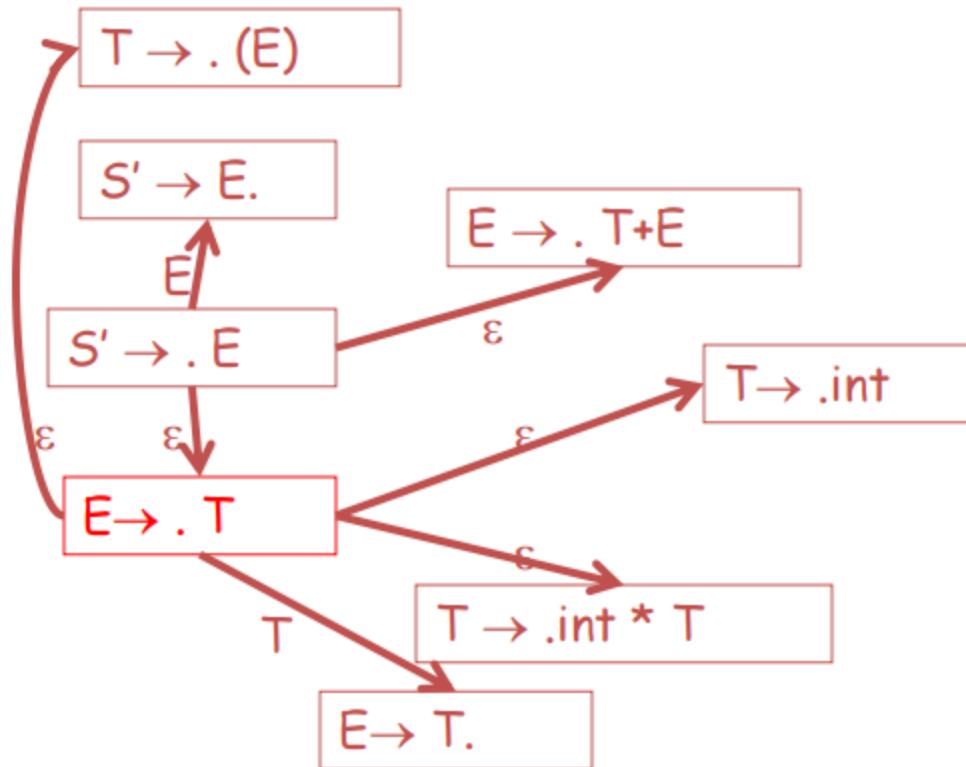
Bottom Up Parsing...

$$S' \rightarrow . E$$

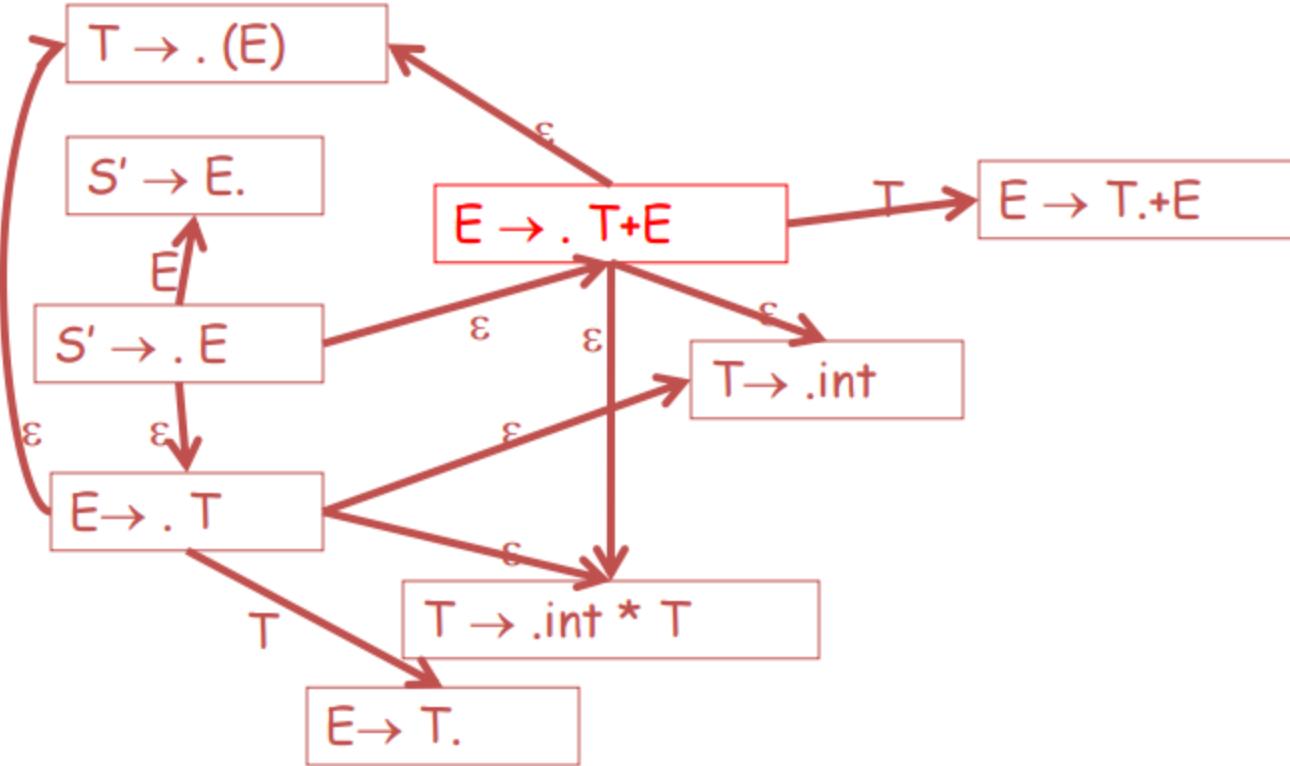
Bottom Up Parsing...



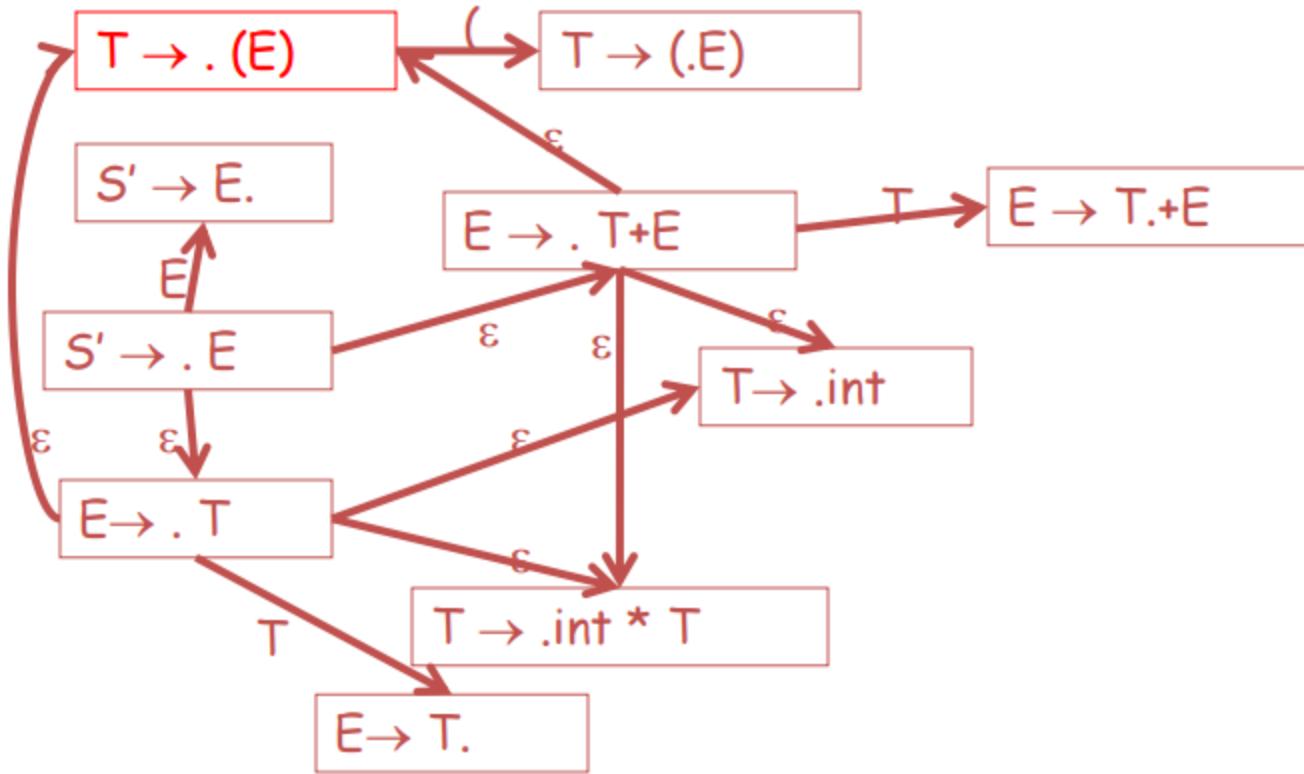
Bottom Up Parsing...



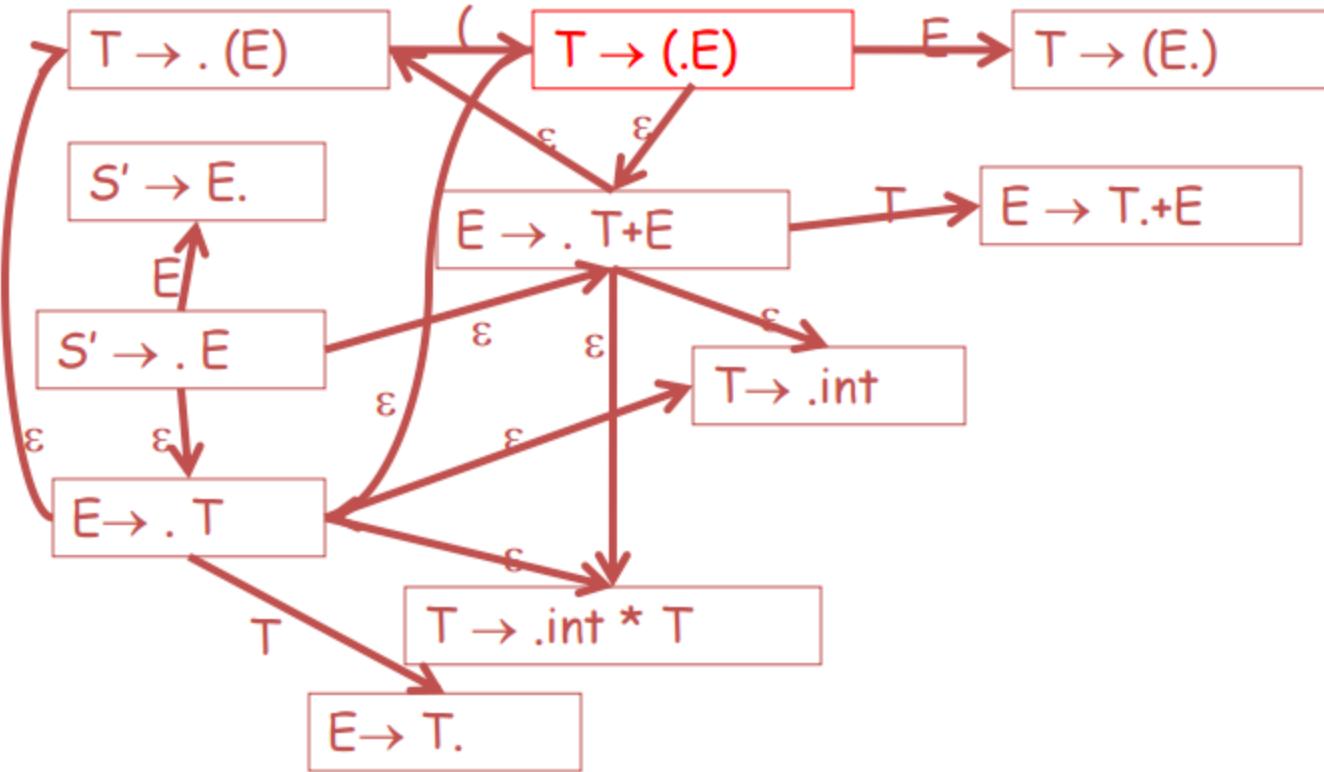
Bottom Up Parsing...



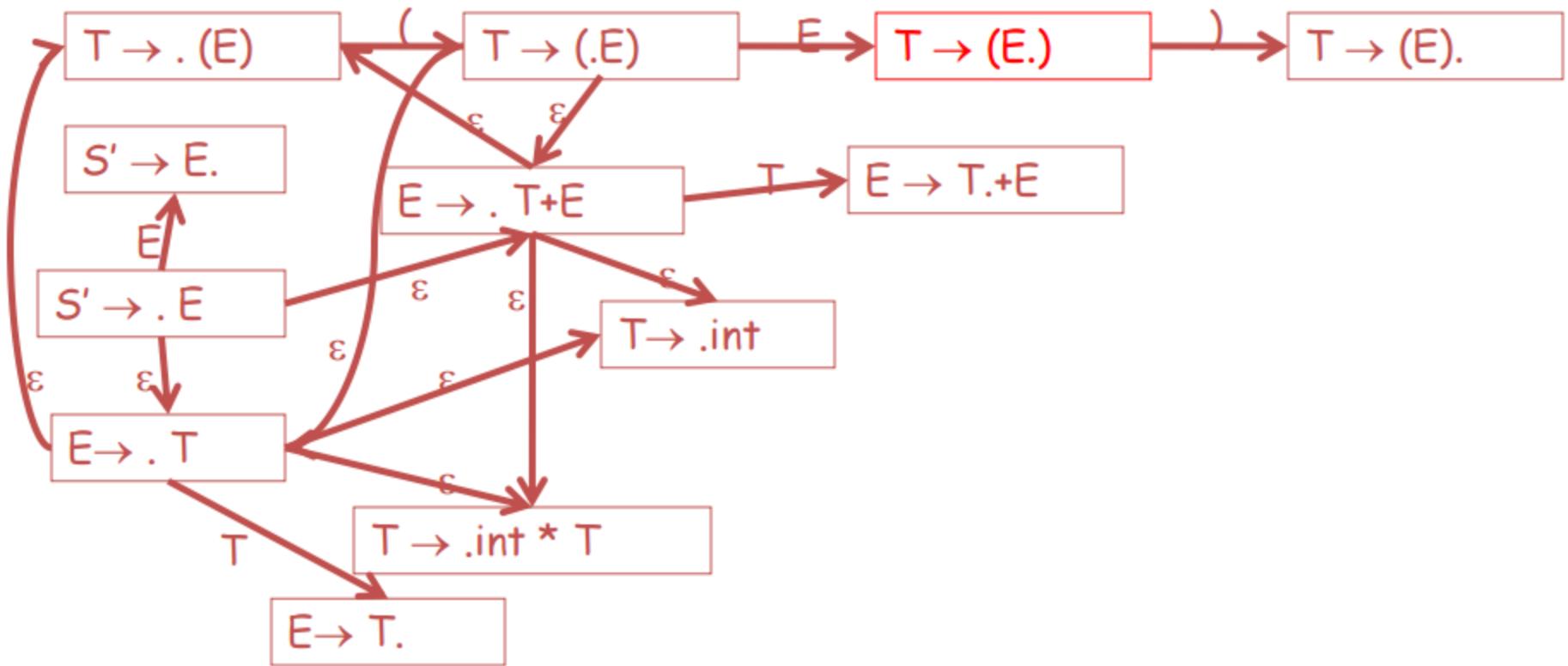
Bottom Up Parsing...



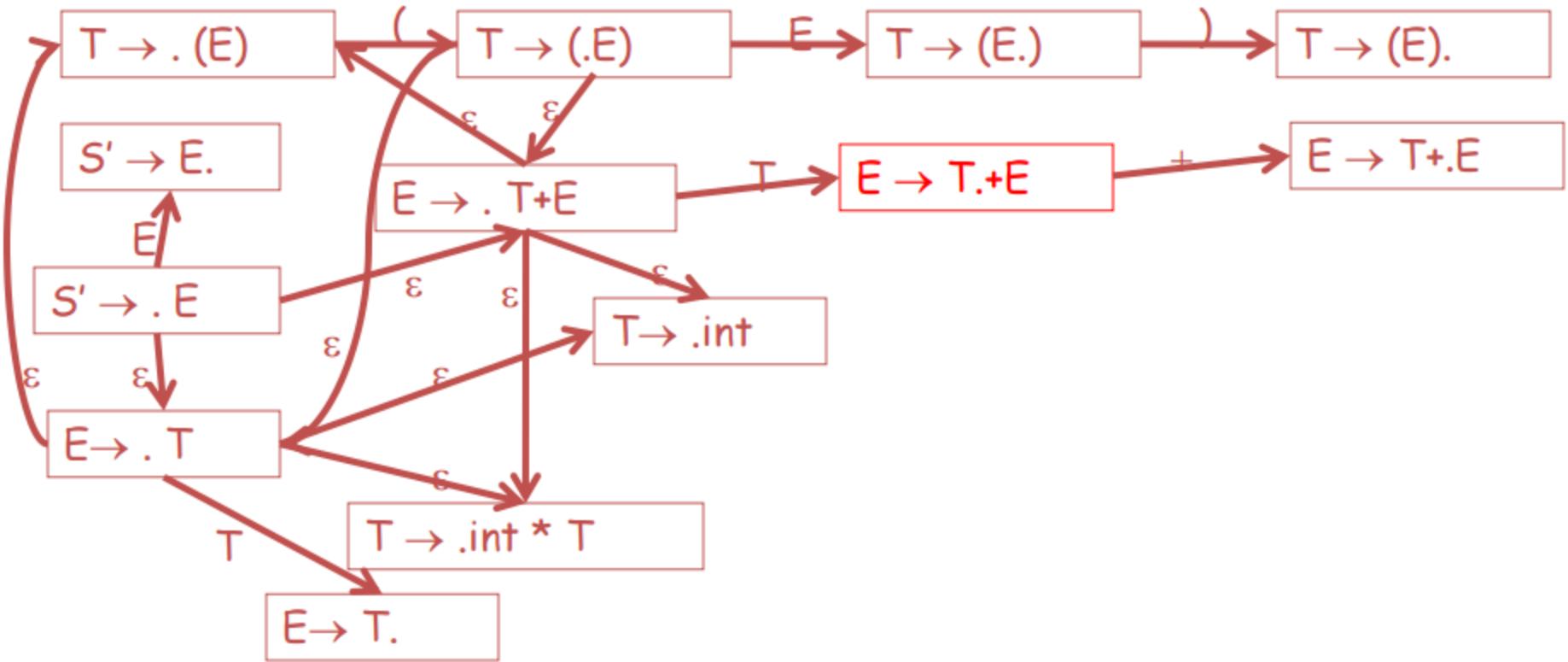
Bottom Up Parsing...



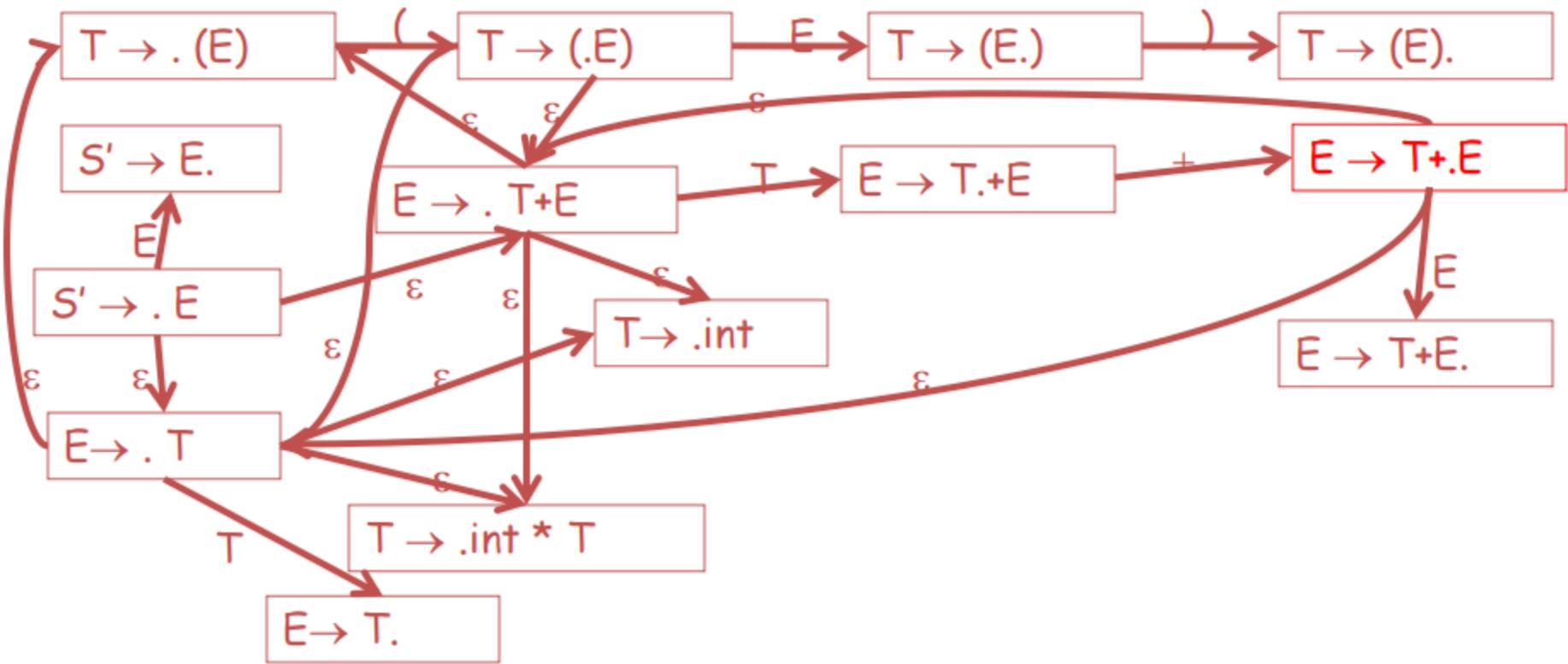
Bottom Up Parsing...



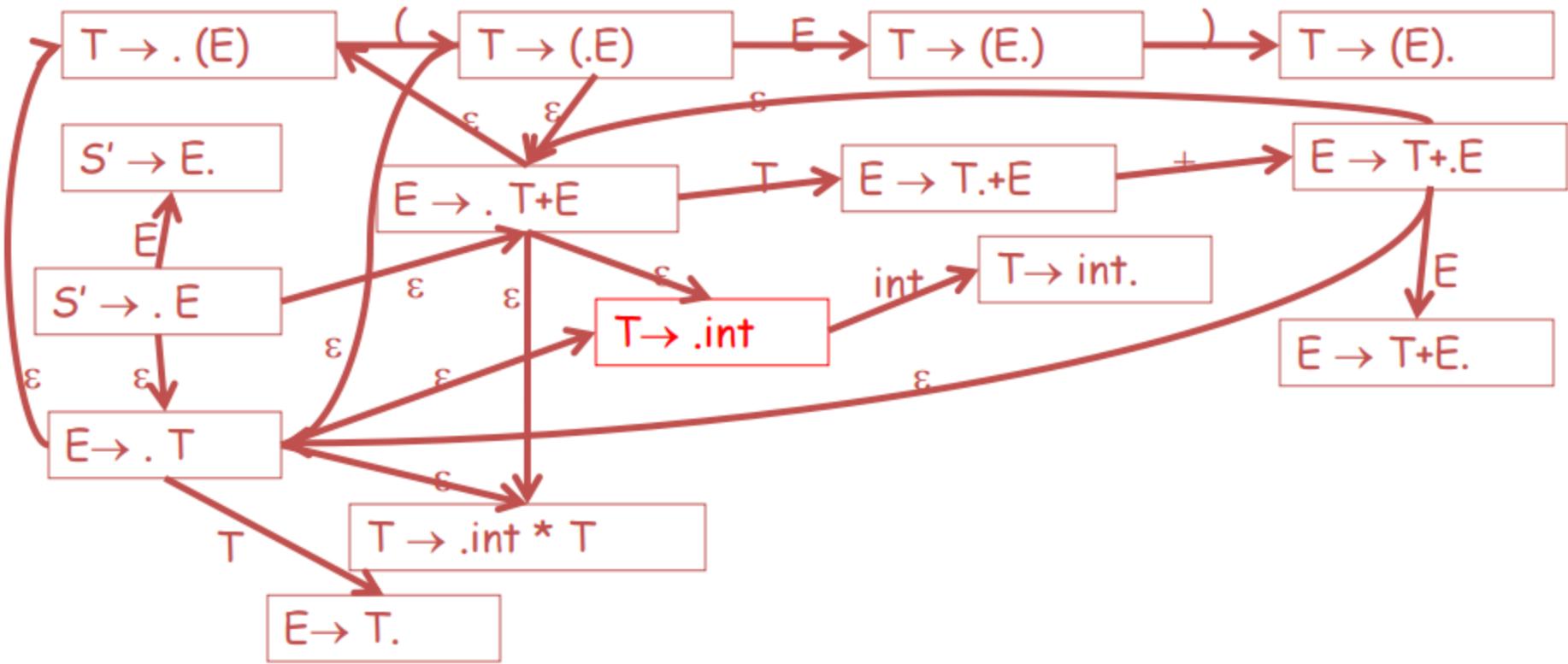
Bottom Up Parsing...



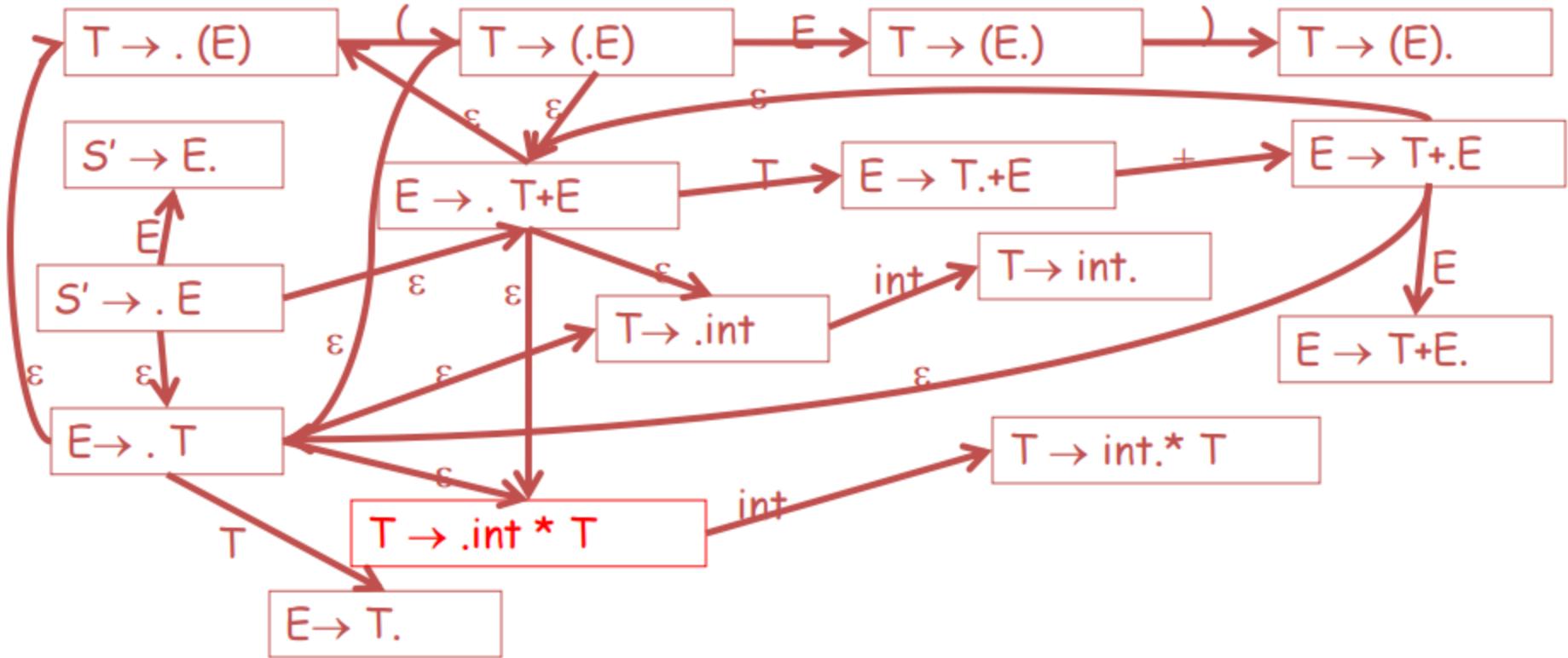
Bottom Up Parsing...



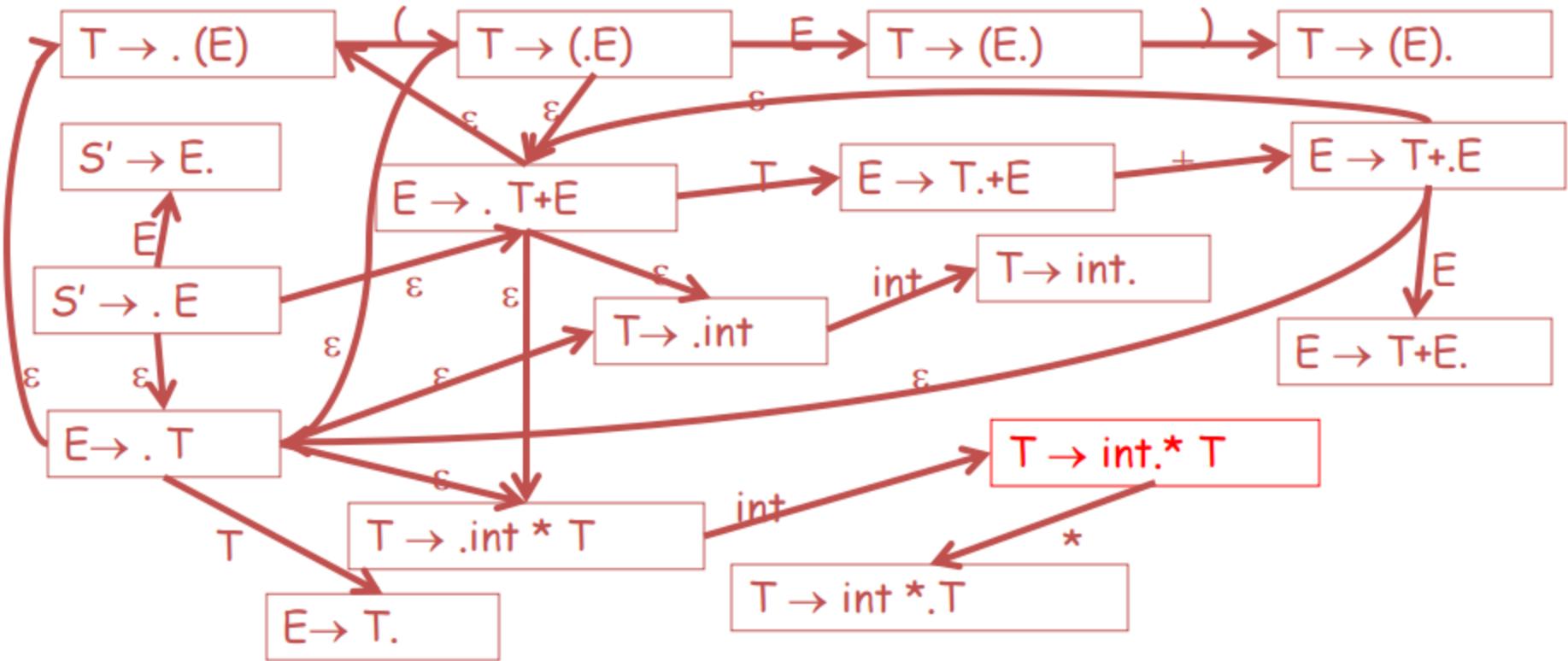
Bottom Up Parsing...



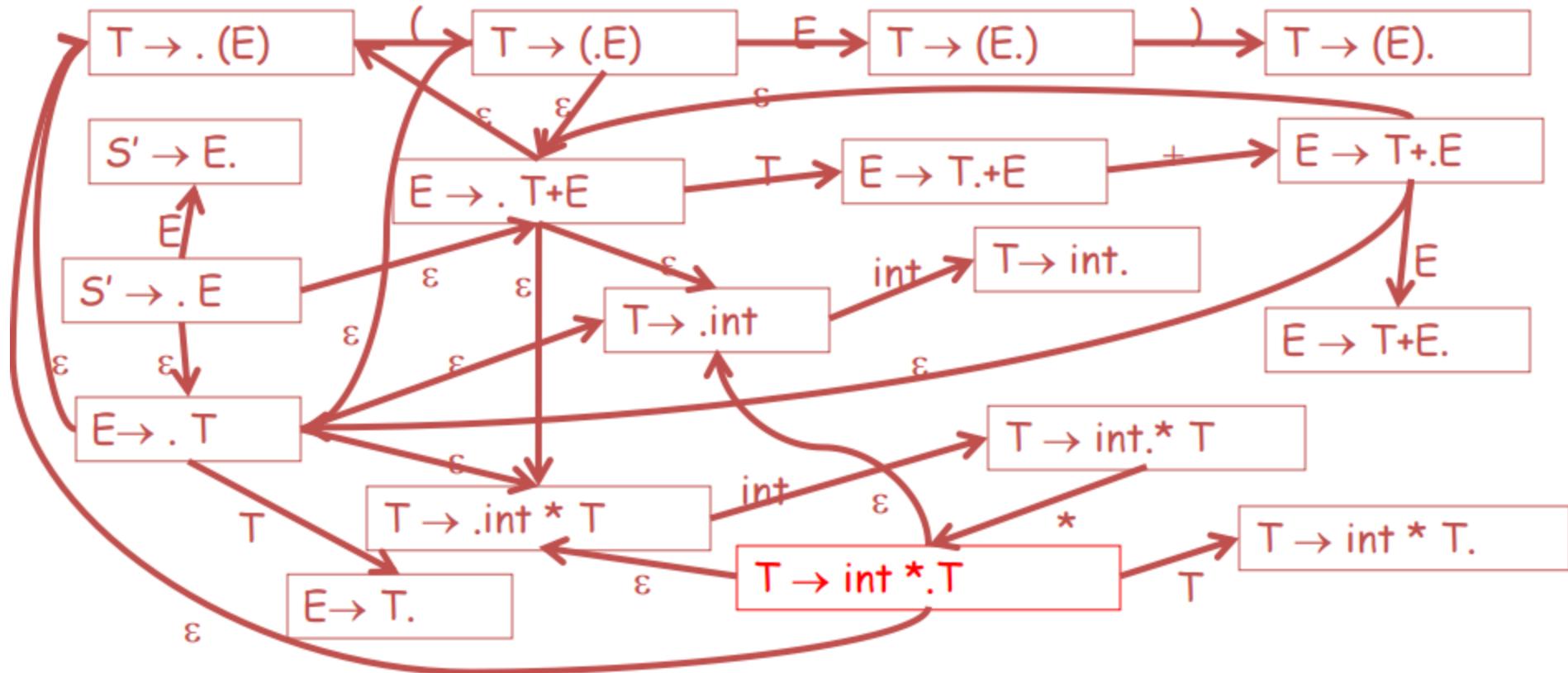
Bottom Up Parsing...



Bottom Up Parsing...



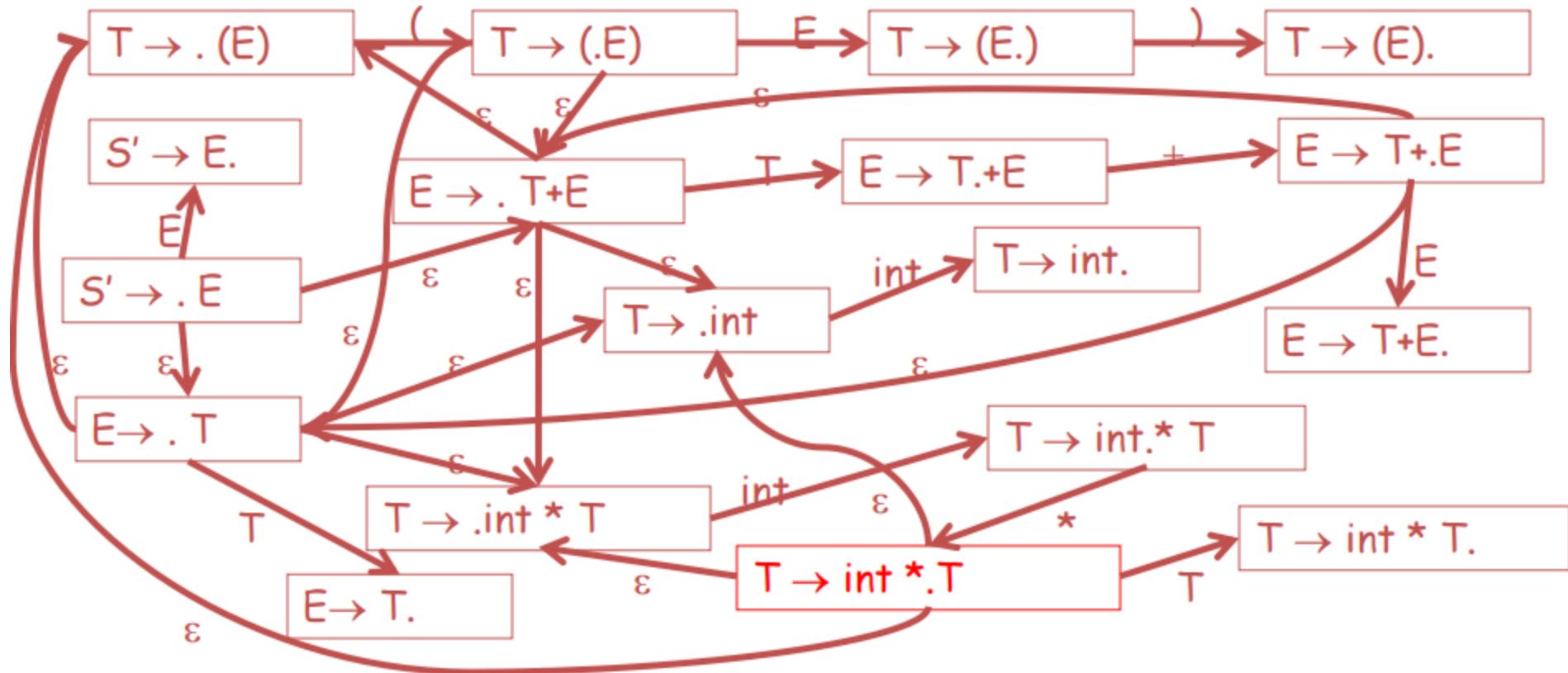
Bottom Up Parsing...



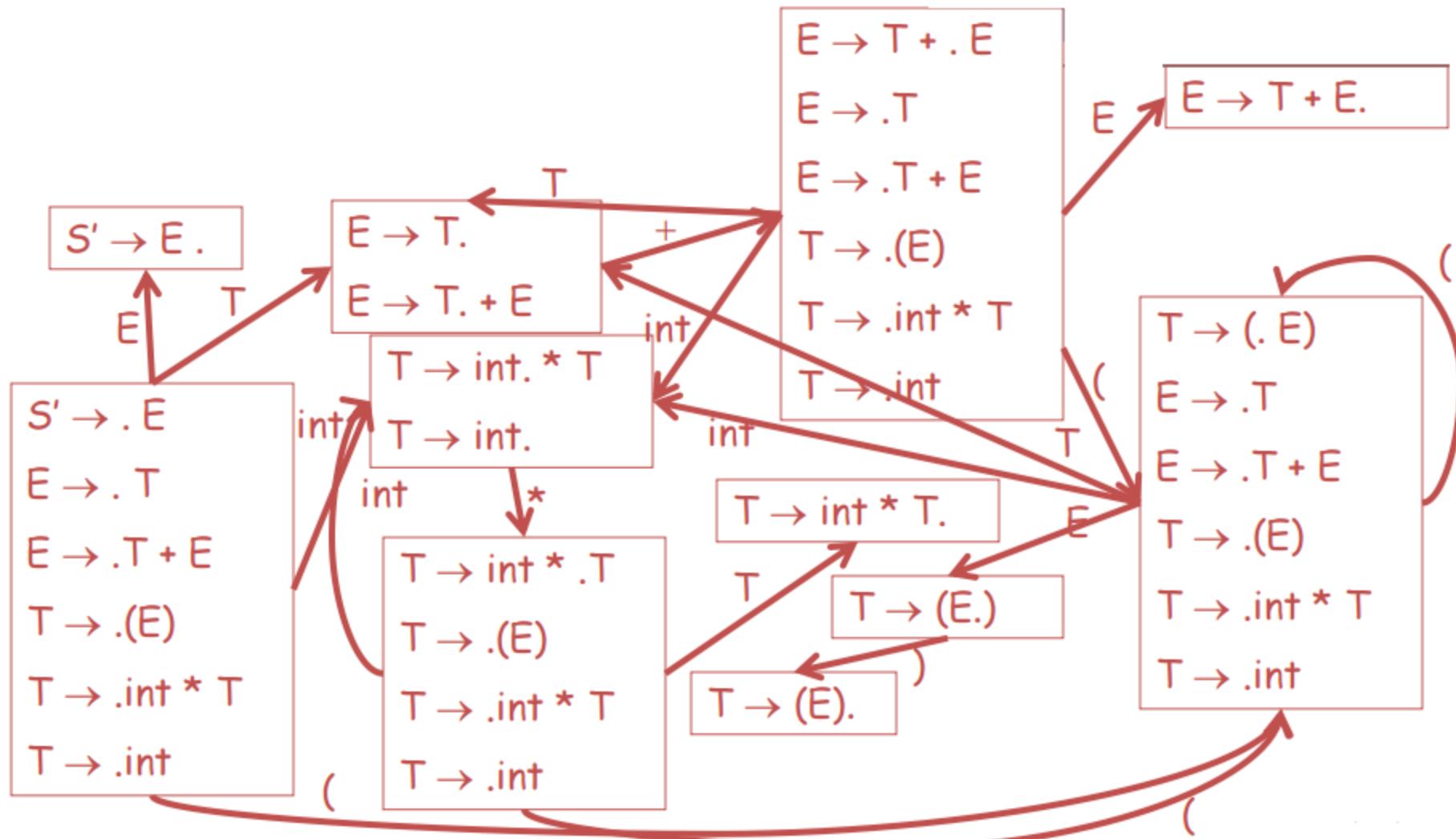
Bottom Up Parsing...

Items Validos

Bottom Up Parsing...



Bottom Up Parsing...



Bottom Up Parsing...

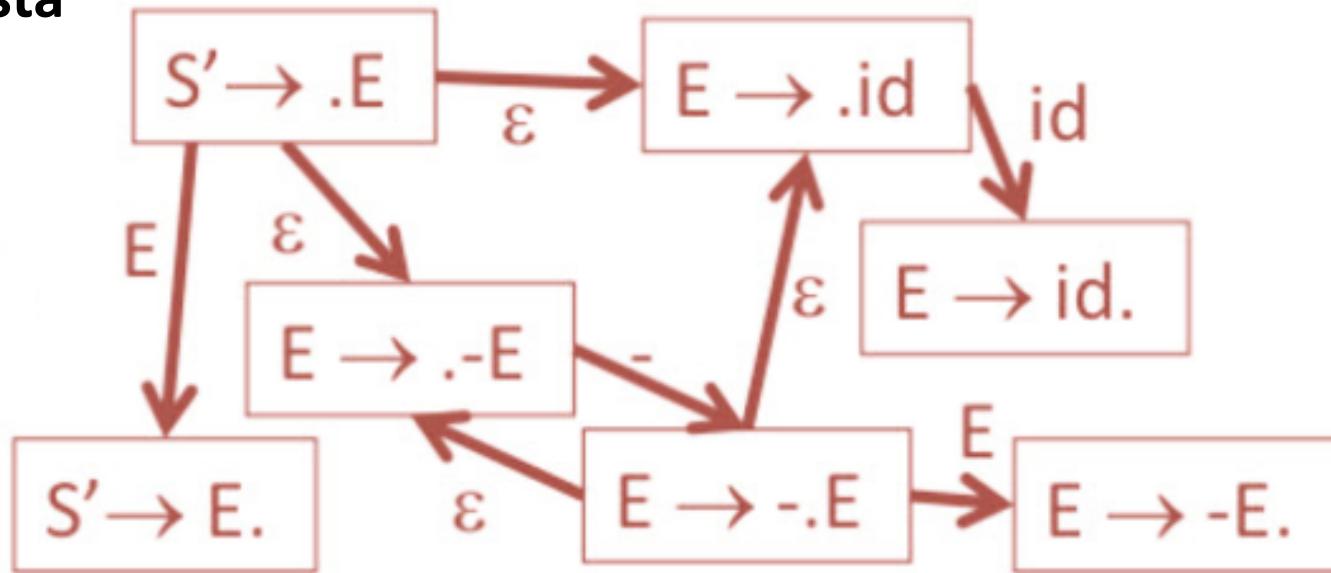
Implemente el NFA de la siguiente gramática dada:

$$S' \rightarrow E \quad E \rightarrow -E \mid id$$

Bottom Up Parsing...

Implemente el NFA de la siguiente gramática dada:

Respuesta



SLR Parsing

SLR Parsing...

- SLR Parsing: Asume
 - La pila contiene α
 - La próxima entrada es t
 - DFA sobre la entrada α termina en estado s
- Hay reducción por $X \rightarrow \beta$ si
 - s contiene el ítem $X \rightarrow \beta$.
- Hay un Shift si
 - s contiene el ítem $X \rightarrow \beta.t\omega$
 - Equivalente a decir que s tiene una transición etiquetada t

SLR Parsing...

- LR(0) tiene un conflicto reduce/reduce si:
 - Cualquier estado tiene dos ítems a reducir
 - $X \rightarrow \beta.$ y $Y \rightarrow \omega.$
- LR(0) tiene un conflicto shift/reduce si:
 - Cualquier estado tiene un ítem a reducir y un ítem a mover (shift)
 - $X \rightarrow \beta.$ y $Y \rightarrow \omega.t\delta$

SLR Parsing...

- SLR = “Simple LR”
- SLR mejora los heuriticos shift/reduce LR(0)
 - Pocos estados tienen conflictos

SLR Parsing...

- Asume
 - La pila contiene α
 - La próxima entrada es t
 - DFA sobre la entrada α termina en estado s
- Hay reducción por $X \rightarrow \beta$ si
 - s contiene el ítem $X \rightarrow \beta$.
 - $t \in \text{Follow}(X)$
- Hay un Shift si
 - s contiene el ítem $X \rightarrow \beta.t\omega$

SLR Parsing...

- Si hay conflictos bajo esas reglas,
la gramática no es SLR

SLR Parsing...

- Considere la siguiente gramática ambigua

$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{int}$$

- El DFA de esa gramática contiene un estado con los siguientes ítems:

$$E \rightarrow E * E . \quad E \rightarrow E . + E$$

- Declarando “* tiene una precedencia mayor a +” resuelva el conflicto a favor de la reducción

SLR Parsing...

- El término "declaración precedente" es engañoso
- Definen resolución de conflictos, lo cual no es exactamente lo mismo.

SLR Parsing...

1. Sea M un DFA para los prefijos viables de G
2. Sea $|x_1 \dots x_n \$|$ configuración inicial
3. Repita lo siguiente hasta encontrar la configuración $S | \$$
 - - Sea $\alpha | \omega$ la actual configuración
 - - Aplique en M el actual apilado α
 - - Si M rechaza α , presente un error de parseo
 - El apilado α no es un prefijo viable
 - - Si M acepta α con items I , sea a la siguiente entrada
 - Haga un shift si $X \rightarrow \beta. a \gamma \in I$
 - Haga un reduce si $X \rightarrow \beta. \in I$ y $a \in \text{Follow}(X)$
 - Reporte un error de parseo en caso contrario

SLR Parsing...

- Si hay un conflicto en el último paso, la gramática no es SLR(k)
- K es la cantidad de lookahead
 - en la práctica $k=1$

SLR Parsing...

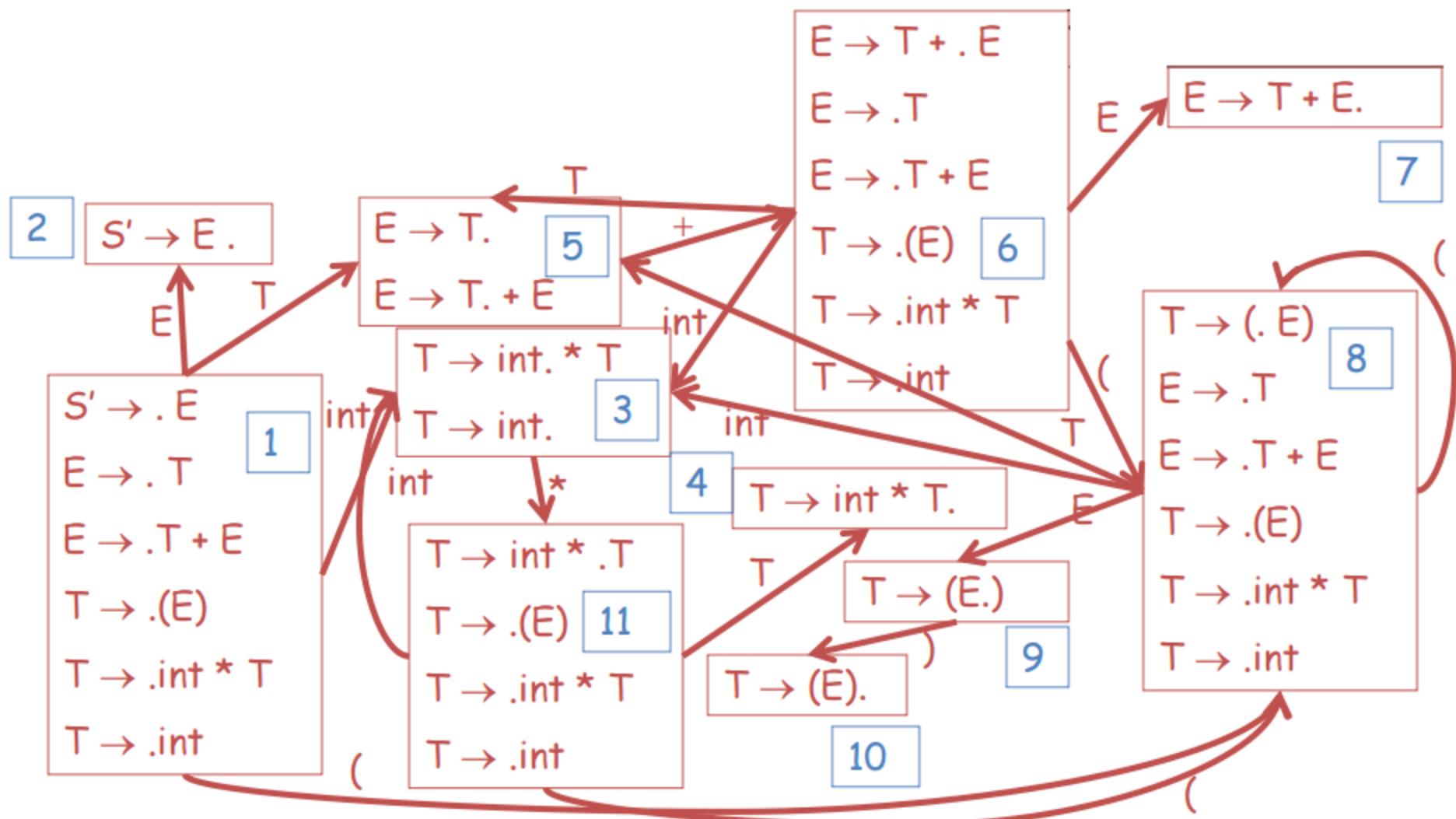
$S' \rightarrow E$

$E \rightarrow T + E \mid T$

$T \rightarrow \text{int} * T \mid \text{int} \mid (E)$

Follow T: {+, \$, int, ()}

SLR Parsing...

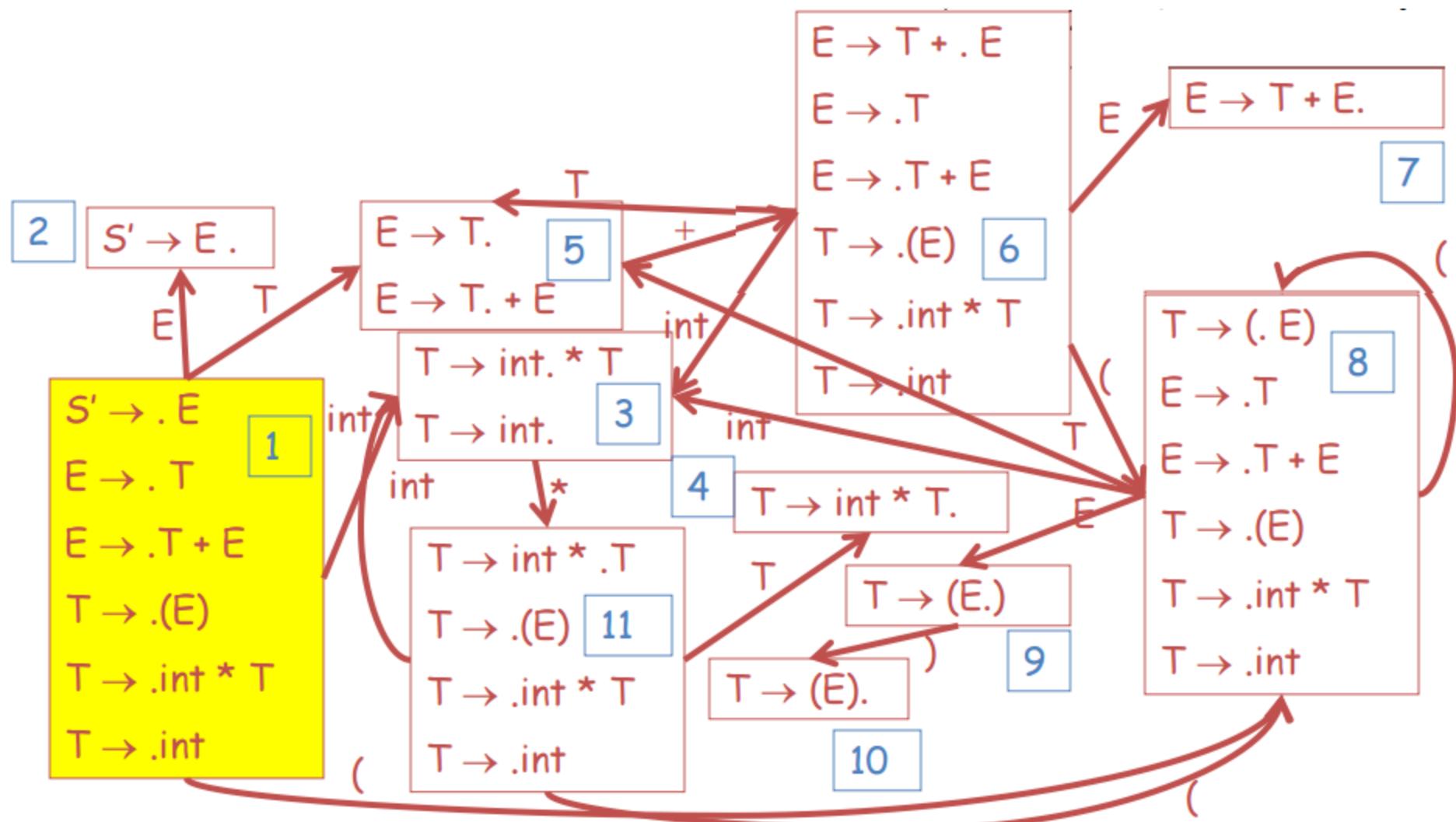


SLR Parsing...

Configuración	Estado	Acción
---------------	--------	--------

| int * int\$

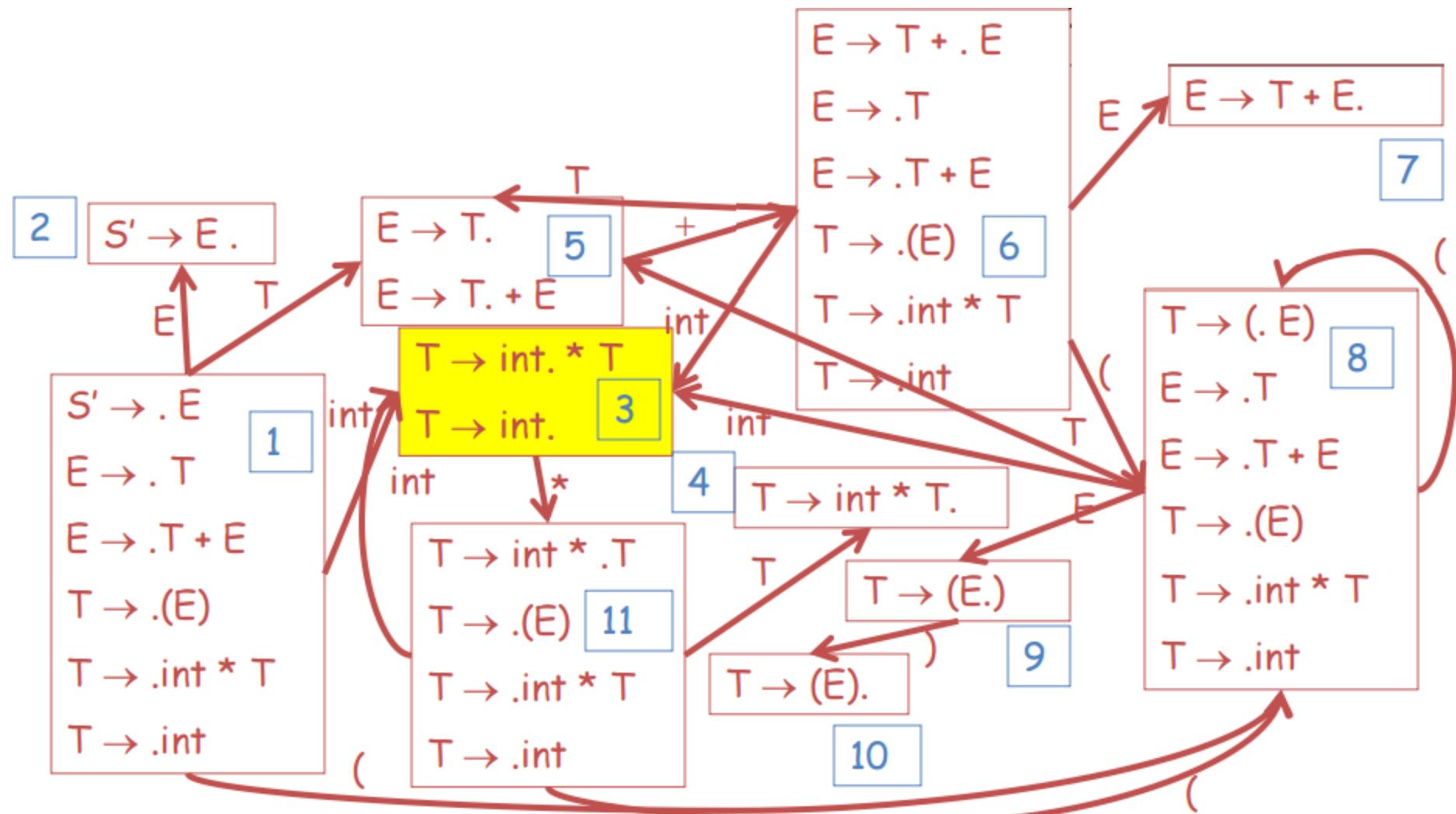
SLR Parsing...



SLR Parsing...

Configuración	Estado	Acción
int * int\$	1	shift
int * int\$		

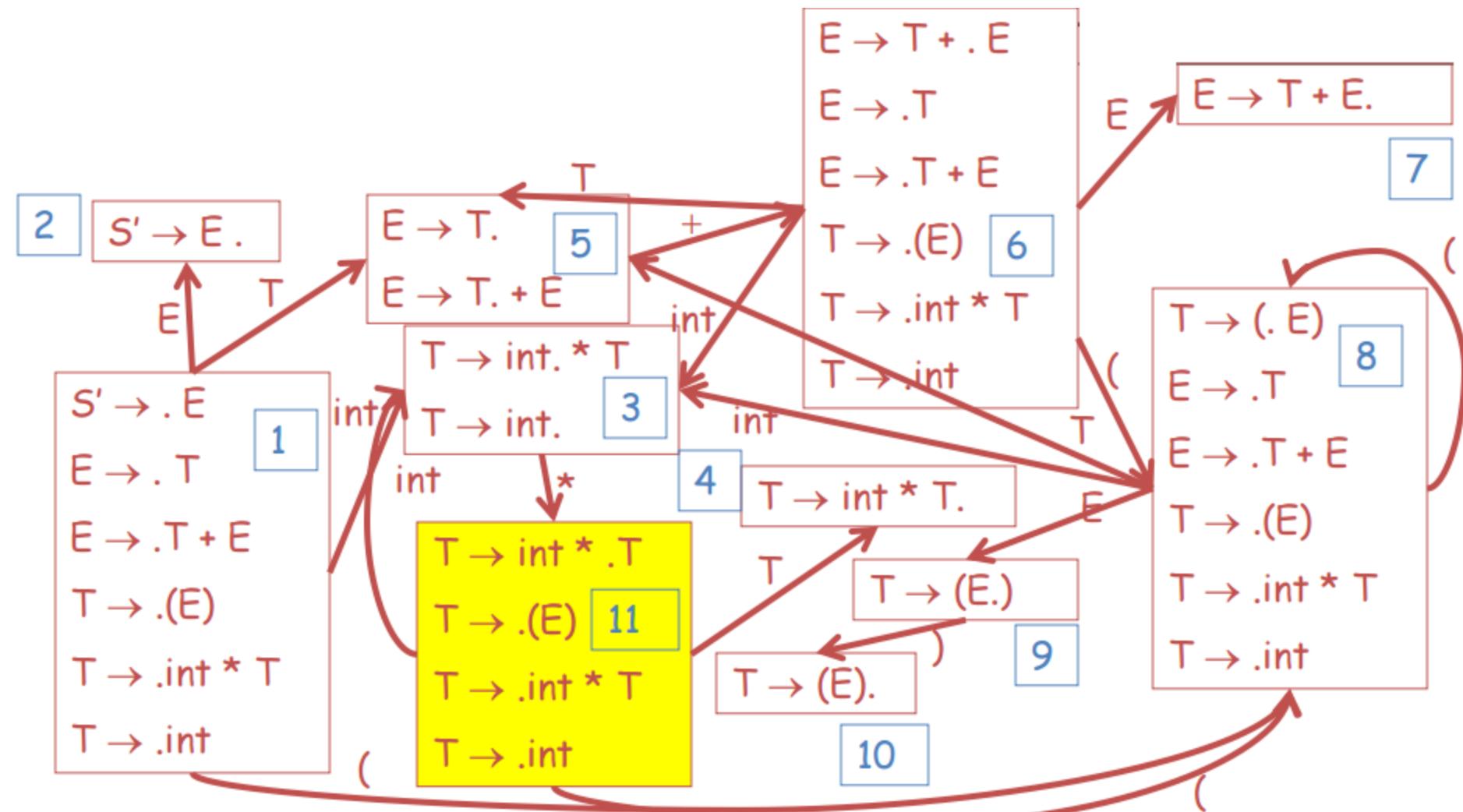
SLR Parsing...



SLR Parsing...

Configuración	Estado	Acción
int * int\$	1	shift
int * int\$	3 * not in Follow(T)	shift
int * int\$		

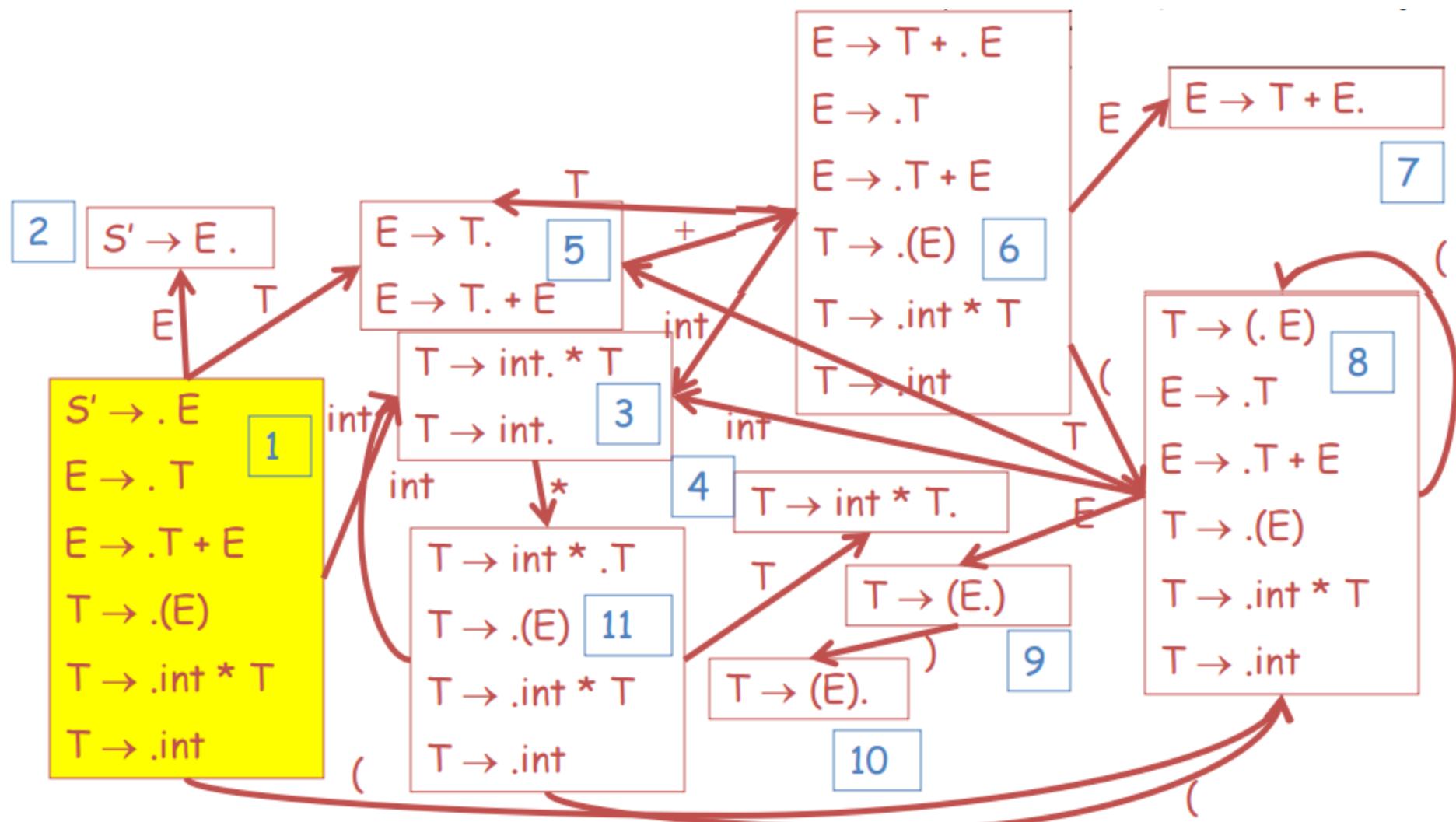
SLR Parsing...



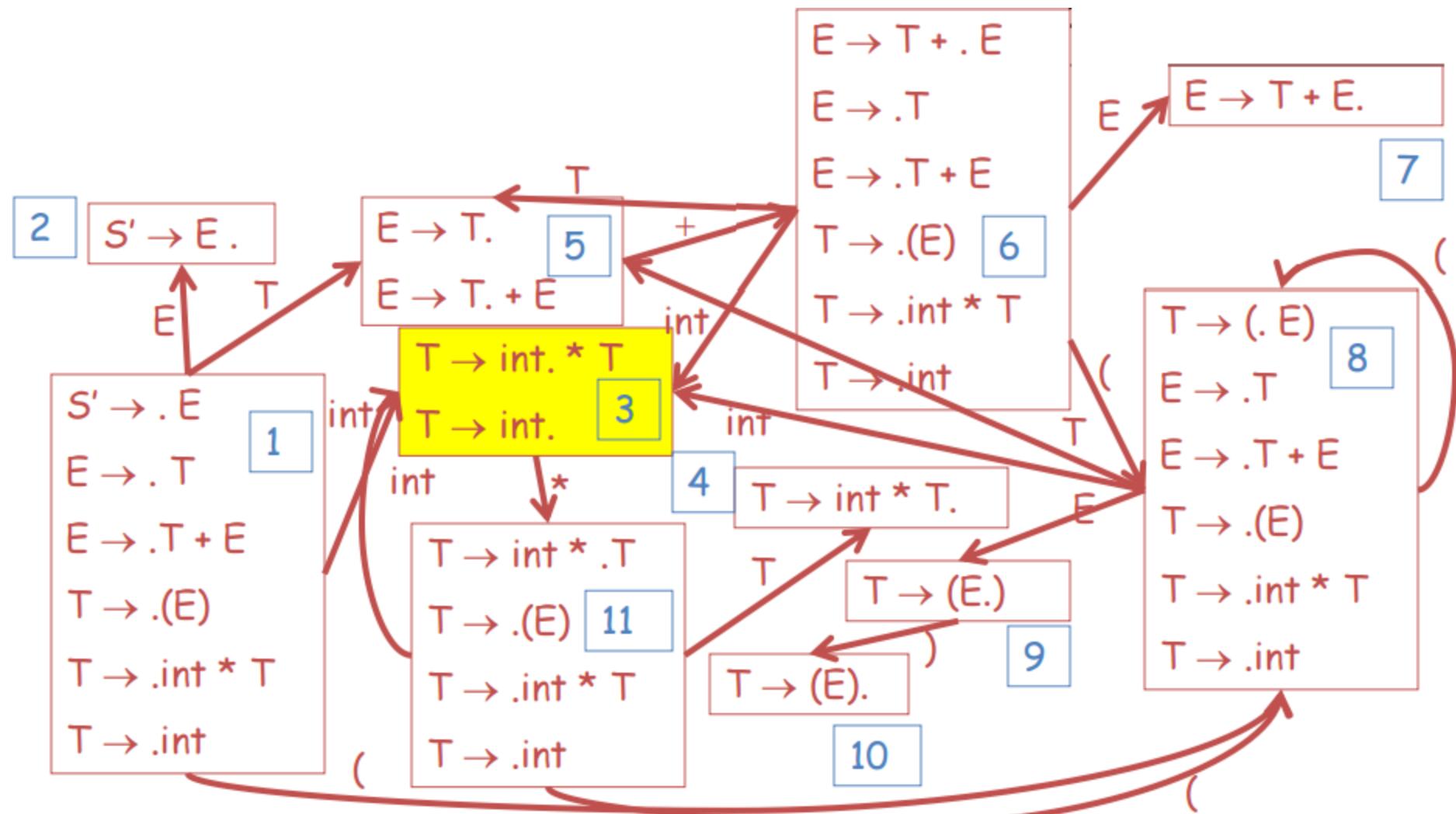
SLR Parsing...

Configuración	Estado	Acción
int * int\$	1	shift
int * int\$	3 * not in Follow(T)	shift
int * int\$	11	shift
int * int \$		

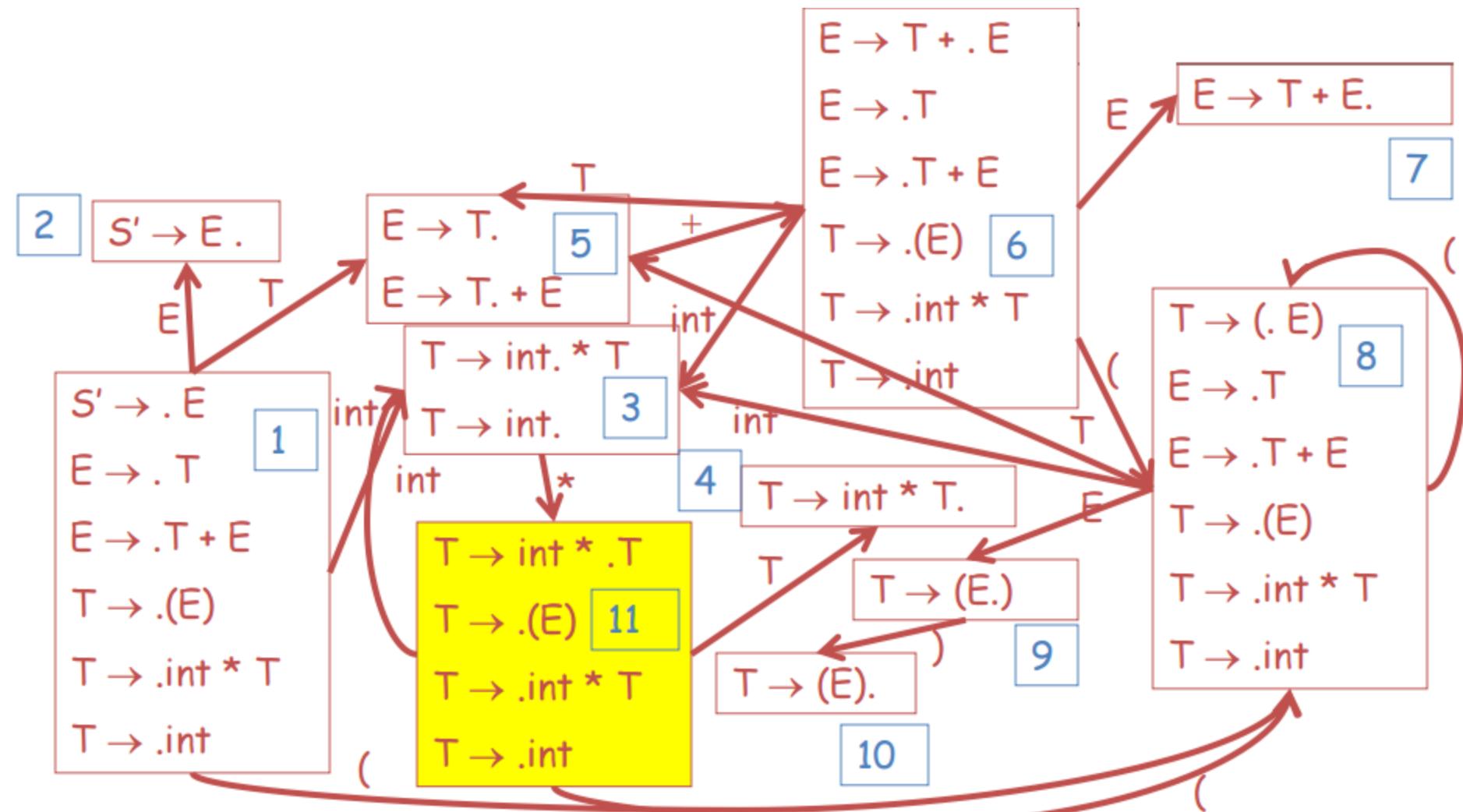
SLR Parsing...



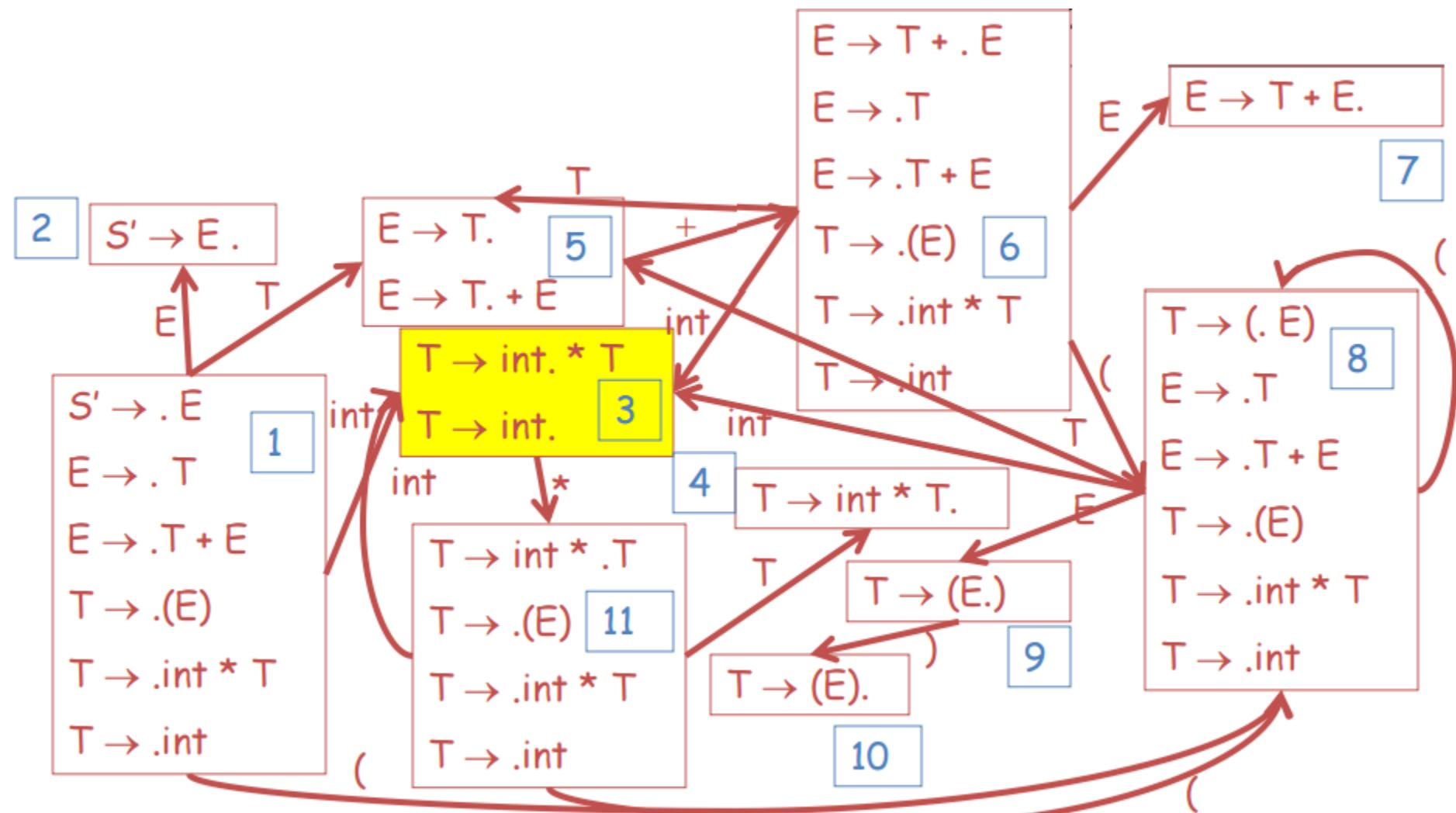
SLR Parsing...



SLR Parsing...



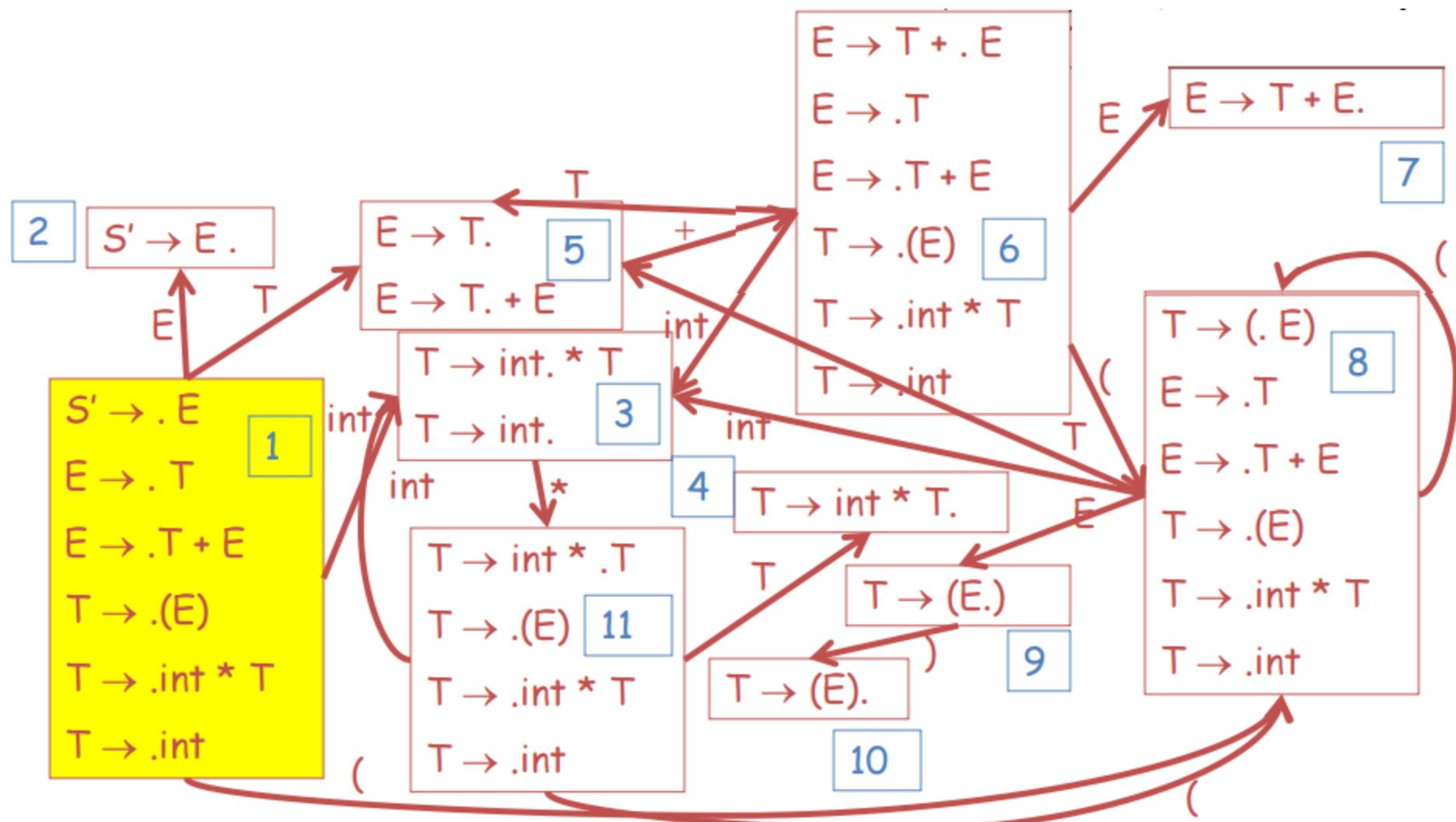
SLR Parsing...



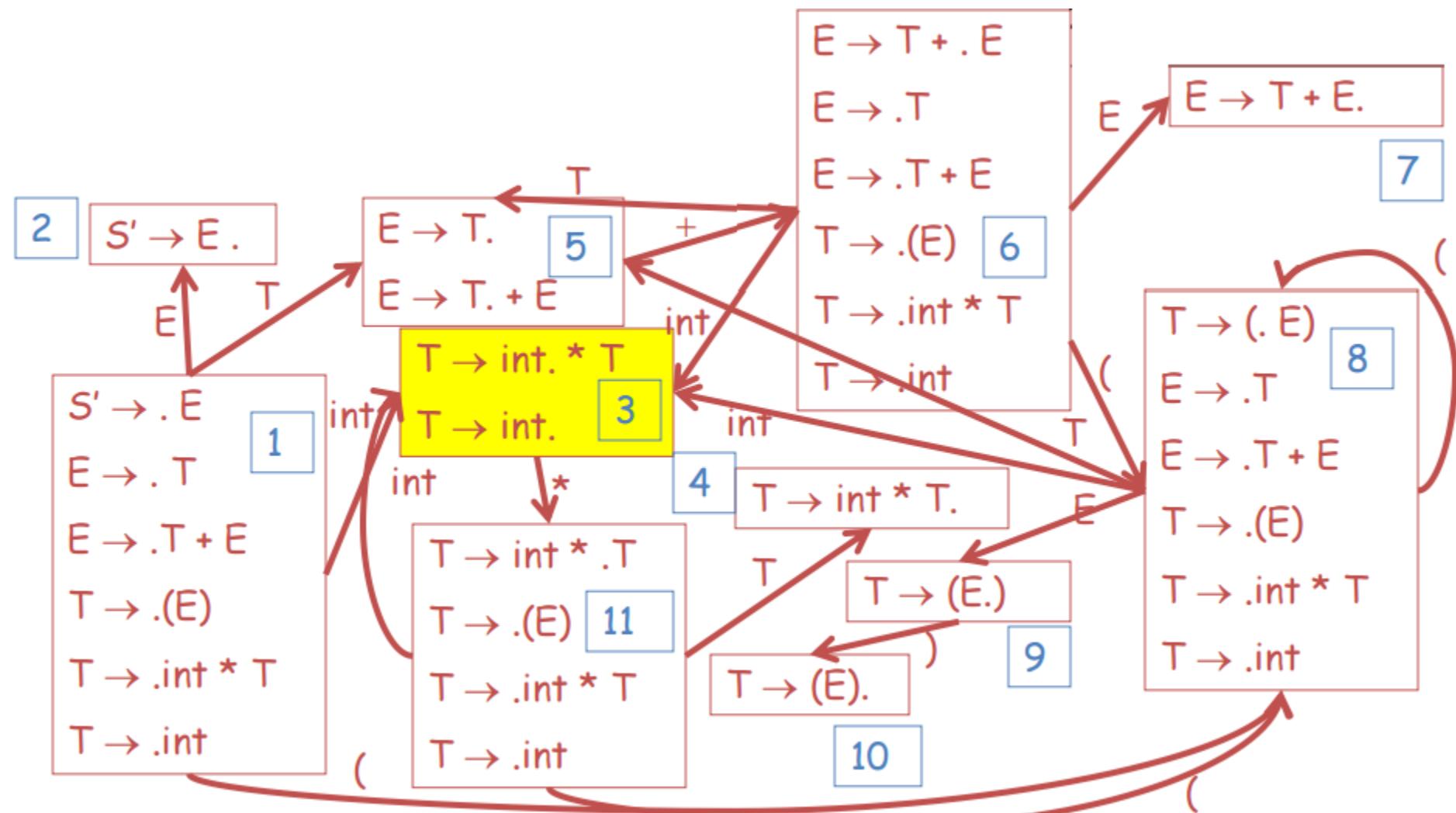
SLR Parsing...

Configuración	Estado	Acción
int * int\$	1	shift
int * int\$	3 * not in Follow(T)	shift
int * int\$	11	shift
int * int \$	3 \$ ∈ Follow(T)	red. T→int
int * T \$		

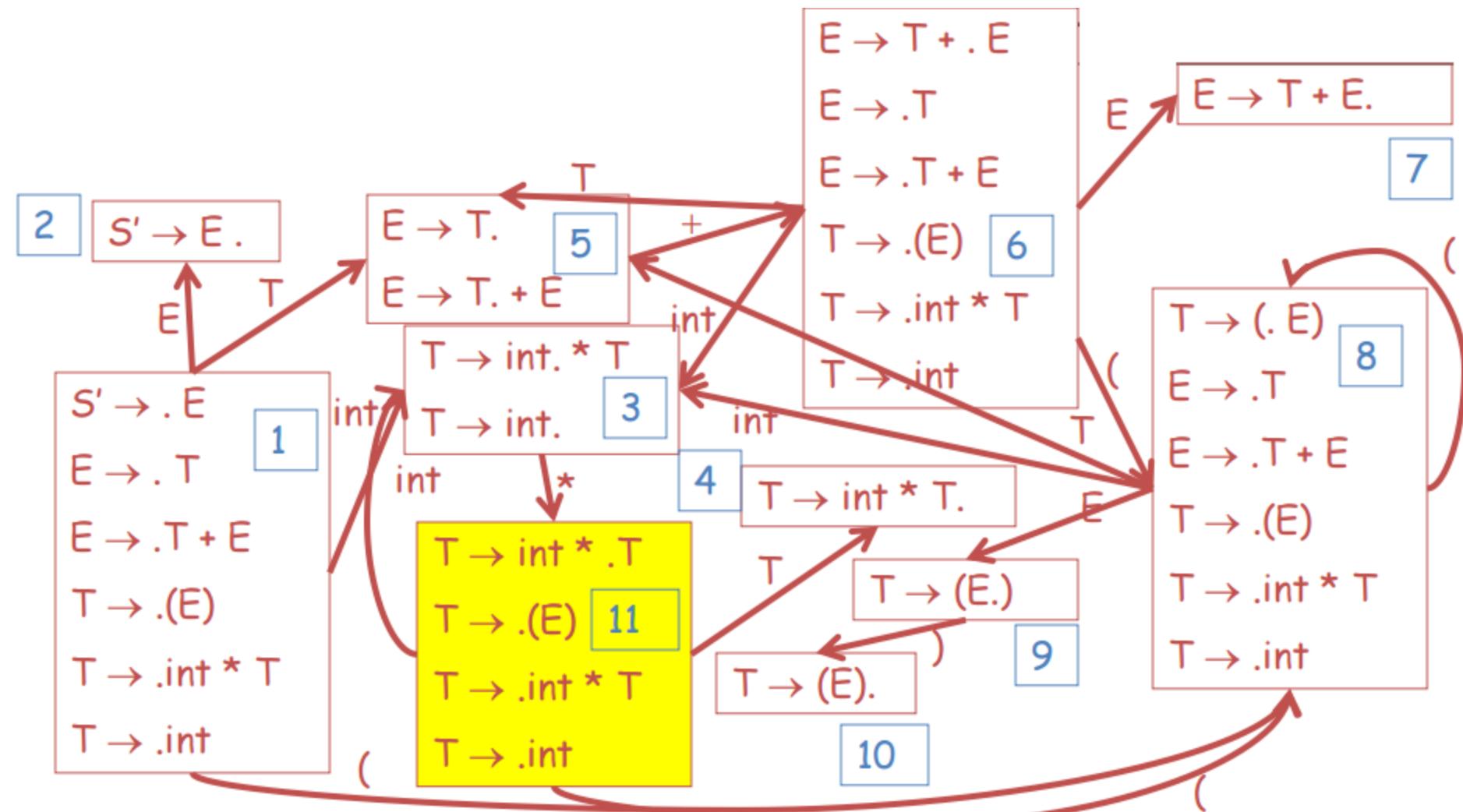
SLR Parsing...



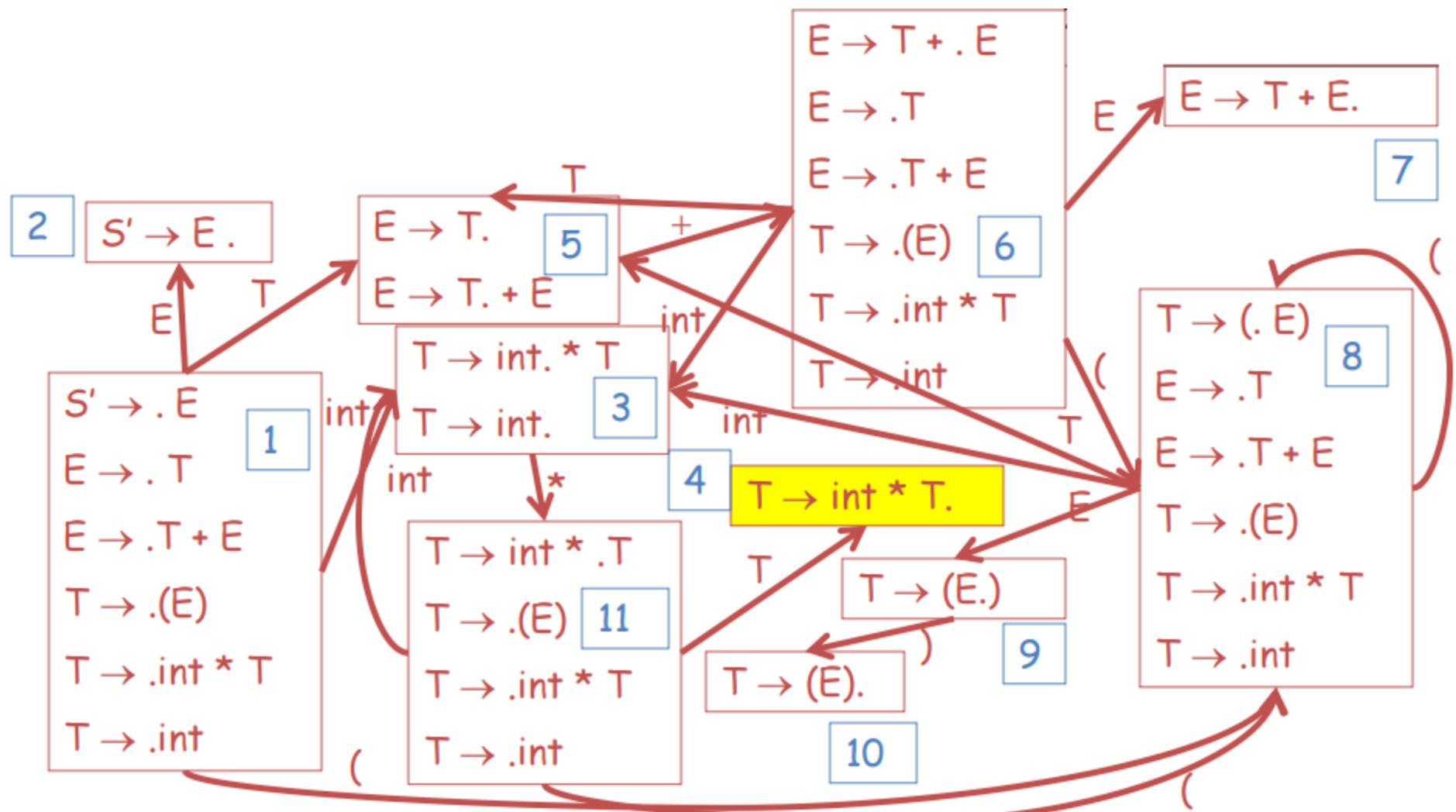
SLR Parsing...



SLR Parsing...



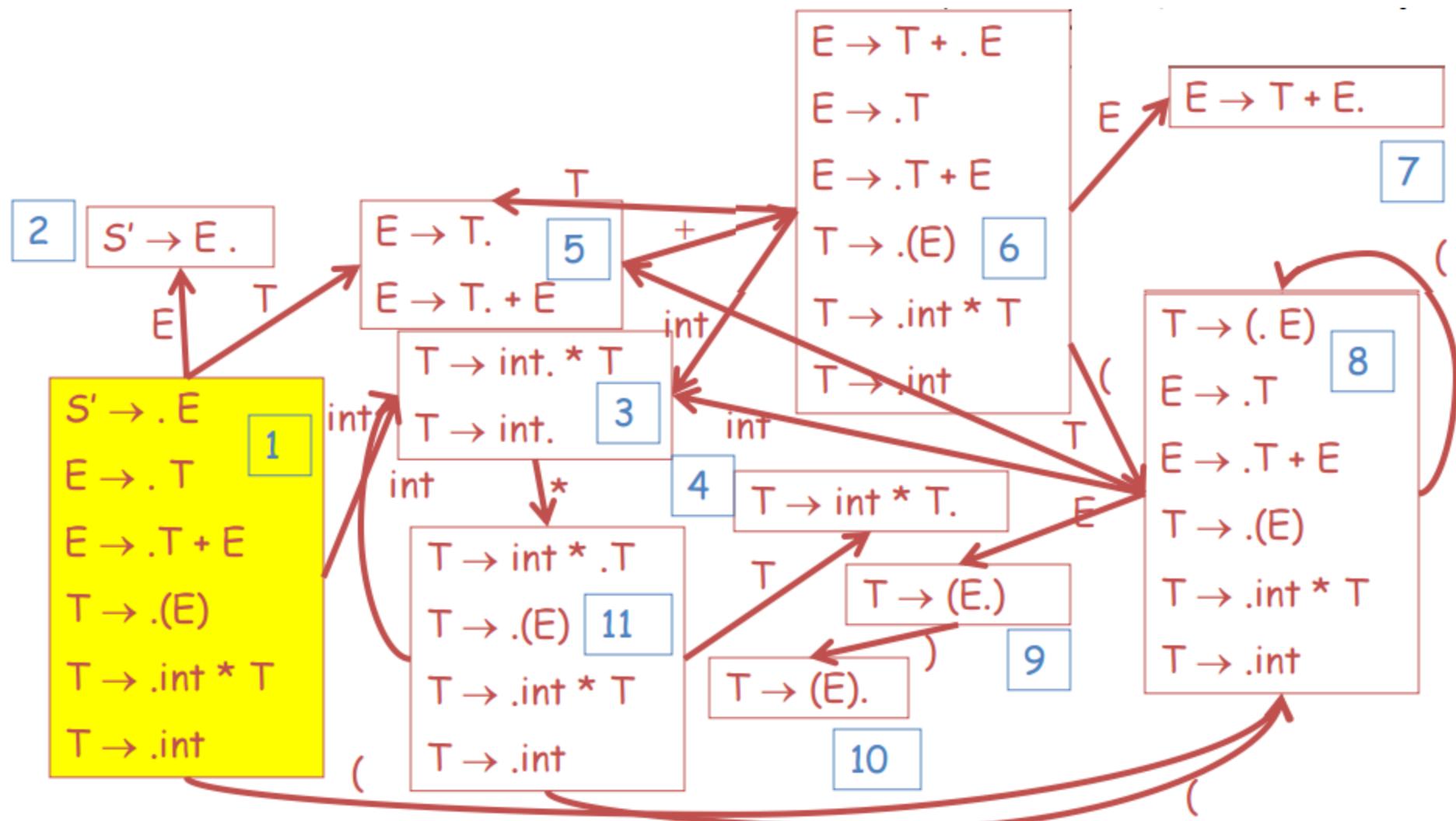
SLR Parsing...



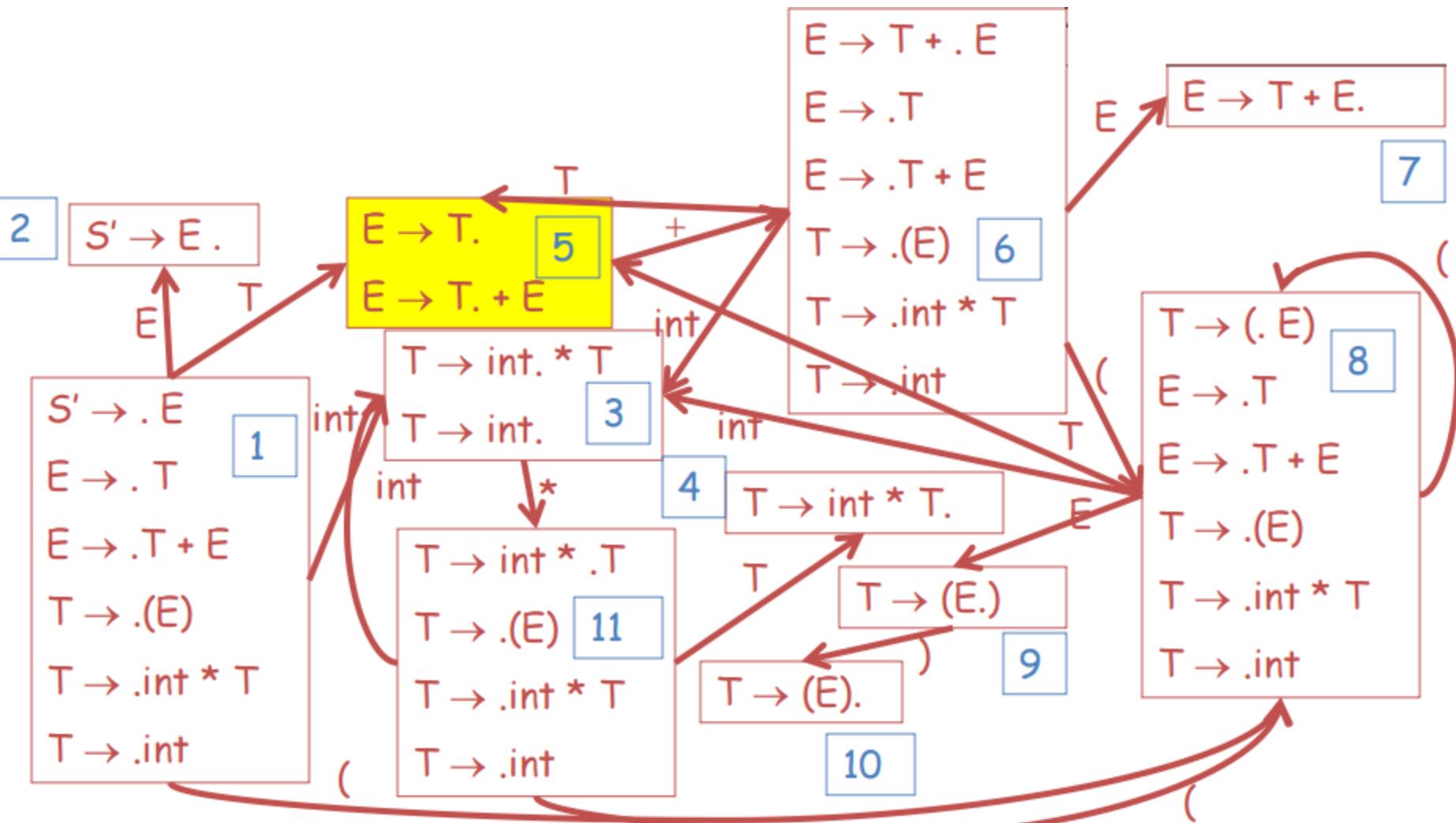
SLR Parsing...

Configuración	Estado	Acción
int * int\$	1	shift
int * int\$	3 * not in Follow(T)	shift
int * int\$	11	shift
int * int \$	3 \$ ∈ Follow(T)	red. T→int
int * T \$	4 \$ ∈ Follow(T)	red. T→int*T
T \$		

SLR Parsing...



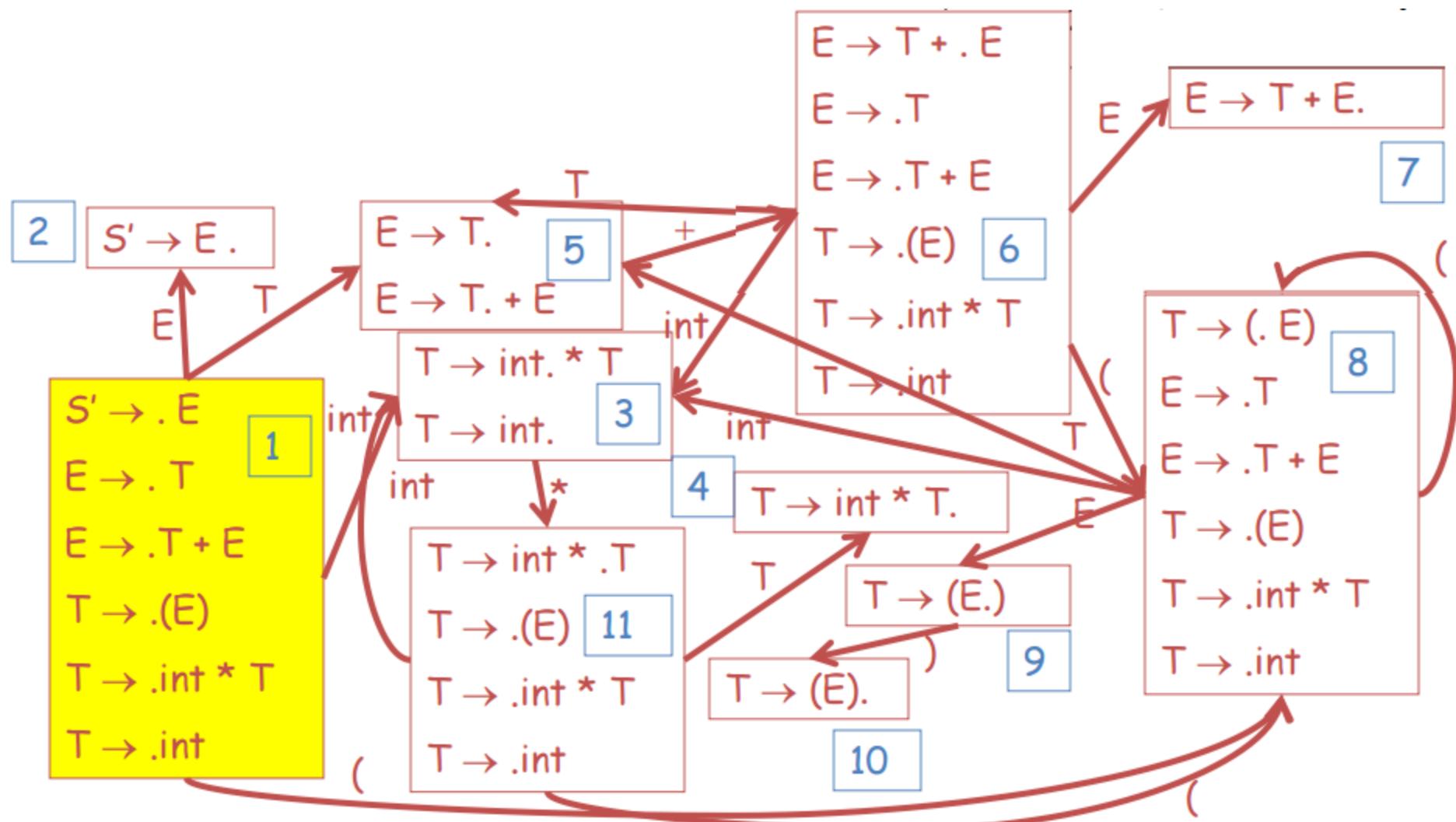
SLR Parsing...



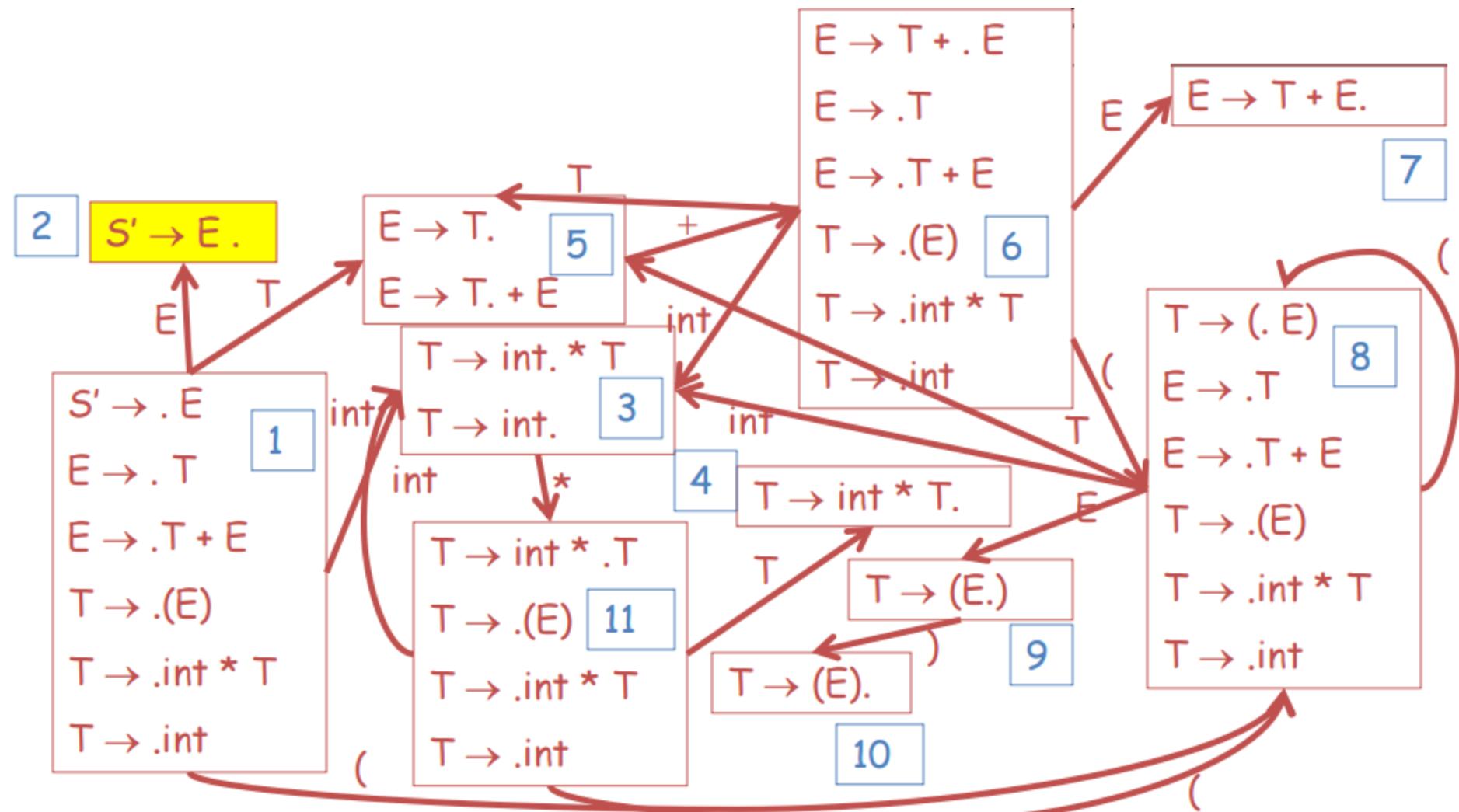
SLR Parsing...

Configuración	Estado	Acción
int * int\$	1	shift
int * int\$	3 * not in Follow(T)	shift
int * int\$	11	shift
int * int \$	3 $\$ \in \text{Follow}(T)$	red. $T \rightarrow \text{int}$
int * T \$	4 $\$ \in \text{Follow}(T)$	red. $T \rightarrow \text{int}^*T$
T \$	5 $\$ \in \text{Follow}(E)$	red. $E \rightarrow T$
E \$		

SLR Parsing...



SLR Parsing...



SLR Parsing...

Configuración	Estado	Acción
int * int\$	1	shift
int * int\$	3 * not in Follow(T)	shift
int * int\$	11	shift
int * int \$	3 \$ ∈ Follow(T)	red. T→int
int * T \$	4 \$ ∈ Follow(T)	red. T→int*T
T \$	5 \$ ∈ Follow(T)	red. E→T
E \$		accept

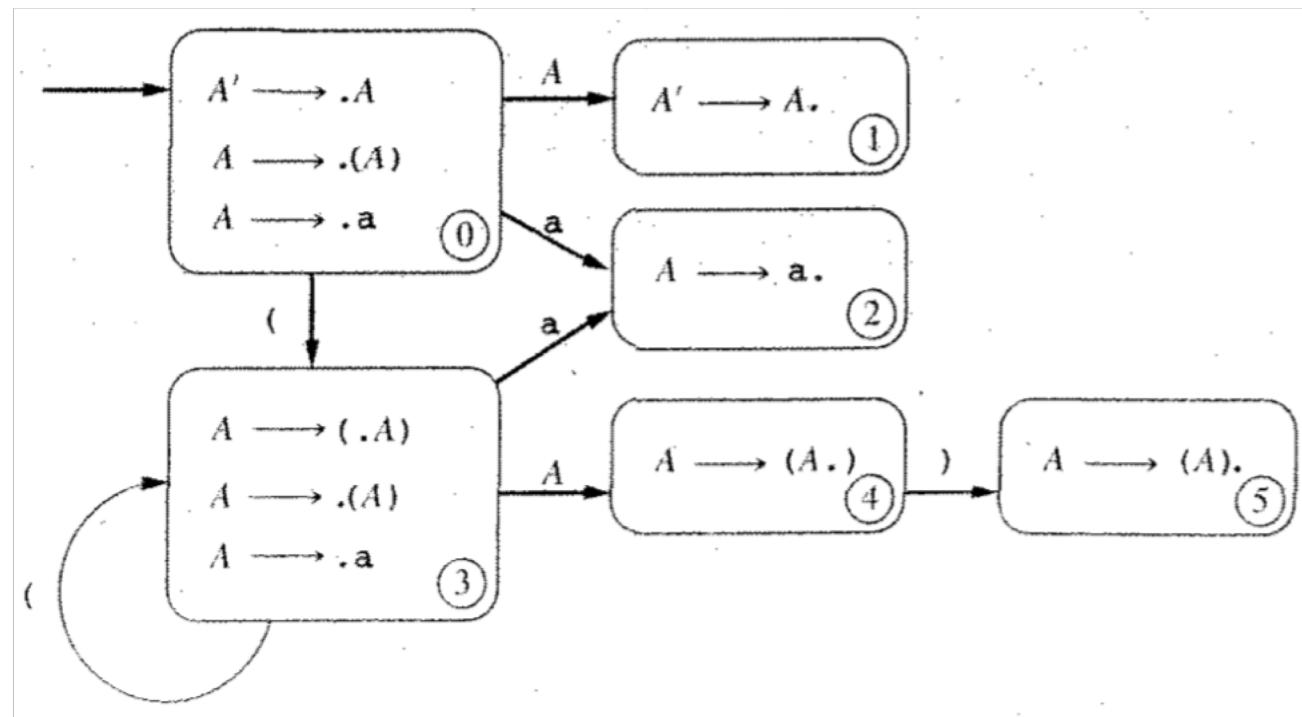
Bottom Up Parsing...

- Dada la siguiente gramática

$$A \rightarrow (A) \mid a$$

- Encuentre el autómata de estados finitos determinístico utilizando el algoritmo Bottom Up Parsing
- Ejecute la derivación de la hilera de entrada “((a))”

Bottom Up Parsing...



Bottom Up Parsing...

Pila	Entrada	Acción
\$	((a)) \$	Shift
(\$	(a)) \$	Shift
((\$	a)) \$	Shift
((a \$)) \$	Reduce A -> a
((A \$)) \$	Shift
((A) \$) \$	Reduce A -> (A)
(A \$) \$	Shift
(A) \$	\$	Reduce A -> (A)
\$	\$	Aceptar

Reconocimiento Ascendente Bottom Up Parsing

