Lecture 07

Syntax Analyzer (Parser)

Part 4: SLR parsing

Hyosu Kim

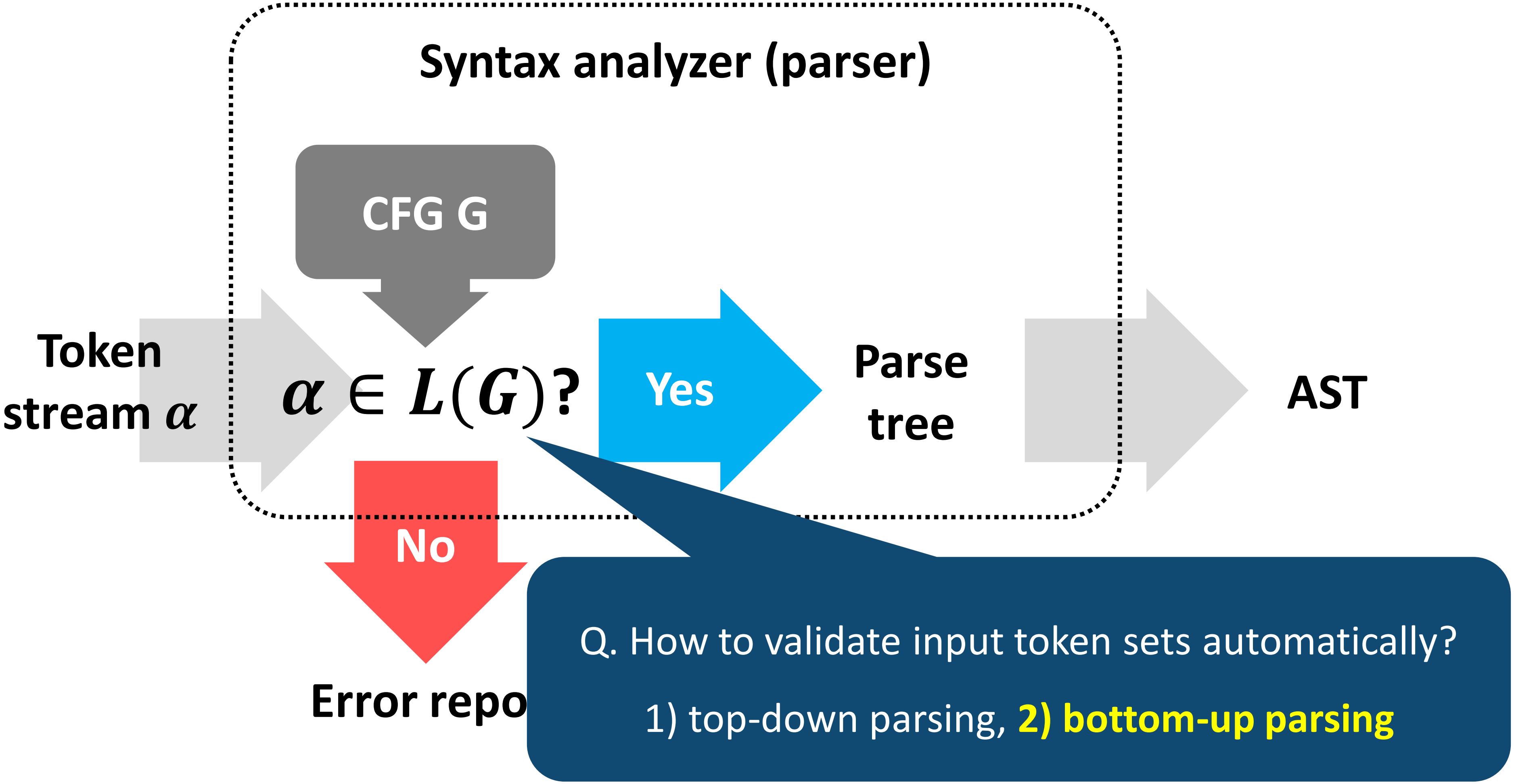
School of Computer Science and Engineering

Chung-Ang University, Seoul, Korea

https://sites.google.com/view/hyosukim

hskimhello@cau.ac.kr, hskim.hello@gmail.com

Syntax analyzer



Bottom-up parsing

Constructs a parse tree for an input string, starting from the leaves (input strings)

and working up towards the root (the start symbol)

It traces **a right derivation** of the input string **in reverse: "reduction"**

$$E \rightarrow T + E|T, \qquad T \rightarrow F * T|F, \qquad F \rightarrow (E)|\, id$$

For $\boldsymbol{id} * \boldsymbol{id}$

- E
- $\Rightarrow_{rm} T$
- $\Rightarrow_{rm} F * T$
- $\Rightarrow_{rm} F * F$
- $\Rightarrow_{rm} F * id$
- $\Rightarrow_{rm} \boldsymbol{id} * \boldsymbol{id}$

id ***** **id**

Bottom-up parsing

Constructs a parse tree for an input string, starting from the leaves (input strings)

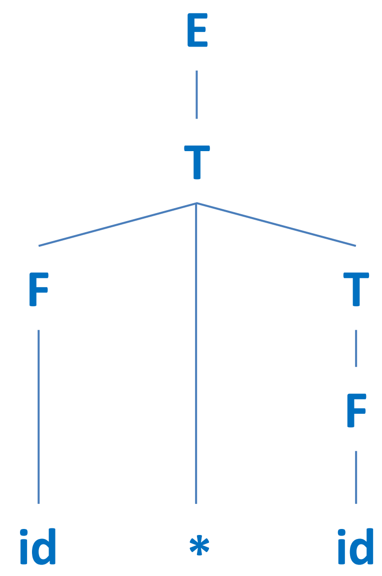
and working up towards the root (the start symbol)

It traces **a right derivation** of the input string **in reverse: "reduction"**

$$E \rightarrow T + E | T, \qquad T \rightarrow F * T | F, \qquad F \rightarrow (E) | id$$

For $\boldsymbol{id * id}$

- $\boldsymbol{E}$
- $\Rightarrow_{rm} T$
- $\Rightarrow_{rm} F * T$
- $\Rightarrow_{rm} F * F$
- $\Rightarrow_{rm} F * id$
- $\Rightarrow_{rm} id * id$



Accept!!

Simple summary of Bottom-up parsing

Shift-reduce parsing

- **Key idea #1: Splitting a string ω into two substrings**

$$\boldsymbol{\omega = \alpha \mid \beta}$$

- **Key idea #2: Shifting or reducing at each step**
  - **Shift:** a splitter | moves to the right

$$XaY|bcd \Rightarrow_{shift} XaYb|cd$$

  - **Reduce:** do a reduction of the suffix (right end substring) of the left substring which matches the RHS (right-hand side) of a production

    If $Z \rightarrow Yb$ is a production of a CFG G,

$$XaYb|cd \Rightarrow_{reduce} XaZ|cb$$

    (In this case, the suffix of the left substring can be XaYb, aYb, Yb, or b)

Simple summary of Bottom-up parsing

Shift-reduce parsing

$$E \rightarrow T + E|T, \qquad T \rightarrow F * T|F, \qquad F \rightarrow (E)|\, id$$

For an input string $id * id + id$

State	Action
$\vert id * id + id$	Shift
$id\vert * id + id$	Reduce by $F \rightarrow id$
$F\vert * id + id$	Shift
$F * \vert id + id$	Shift
$F * id\vert + id$	Reduce by $F \rightarrow id$
$F * F\vert + id$	Reduce by $T \rightarrow F$
$F * T\vert + id$	Reduce by $T \rightarrow F * T$
$T\vert + id$	Shift
$T + \vert id$	Shift
$T + id\vert$	Reduce by $F \rightarrow id, T \rightarrow F, E \rightarrow T, E \rightarrow T + E$
$E\vert$	Accept!!

Simple summary of Bottom-up parsing

But, there are still two types of conflicts

1. Shift-reduce conflict

At each step, we should decide whether to shift or reduce

- e.g., If a production $A \to \alpha$ exists, what should we do with $\boldsymbol{\alpha} \mid \boldsymbol{\beta}$??

State	Action
$id \mid * \, id$	Shift?? Reduce by $F \to id$??

2. Reduce-reduce conflict

In reduction, we should decide which reduction to be used

- e.g., If two productions $A \to \alpha$ and $B \to \alpha$ exist, which reduction should be used for $\boldsymbol{\alpha} \mid \boldsymbol{\beta}$??

State	Action
$F * T \mid$	Reduce by $T \to F * T$?? Reduce by $E \to T$??

Simple summary of Bottom-up parsing

But, there are still two types of conflicts

1. Shift-reduce conflict

At each step, we should d… shift or reduce

- e.g.

Q. How to address these problems????

First, we should study two terms: **"handle" & "viable prefix"**

2. Reduce-reduce conflict

In reduction, we should decide which reduction to be used

- e.g., If two productions $A \rightarrow \alpha$ and $B \rightarrow \alpha$ exist, which reduction should be used for $\boldsymbol{\alpha} \mid \boldsymbol{\beta}$??

State	Action
$F * T \mid$	Reduce by $T \rightarrow F * T$?? Reduce by $E \rightarrow T$??

Definition: Handle

A handle is a substring that matches the complete RHS of a production and is used for a reduction

When $S \Rightarrow_{rm}^{*} \alpha X \omega \Rightarrow_{rm} \alpha\beta\omega$ (S is a start symbol),

- For the reduction of $\alpha\beta|\omega$, a production $X \rightarrow \beta$ is used: $\alpha\beta|\omega \Rightarrow_{reduce} \alpha X|\omega$
- **β is a handle of $\alpha\beta\omega$**

CFG	State	Handle	Action
	$\|id * id$		Shift
	$id\| * id$	$\boldsymbol{id}$	Reduce by $F \rightarrow id$
	$F\| * id$		Shift
$E \rightarrow T + E\|T$	$F * \|id$		Shift
$T \rightarrow F * T\|F,$	$F * id\|$	$\boldsymbol{id}$	Reduce by $F \rightarrow id$
$F \rightarrow (E)\|\ id$	$F * F\|$	$\boldsymbol{F}$	Reduce by $T \rightarrow F$
	$F * T\|$	$\boldsymbol{F * T}$	Reduce by $T \rightarrow F * T$
	$T\|$	$\boldsymbol{T}$	Reduce by $E \rightarrow T$
	$E\|$		Accept

Definition: Handle

A handle is a substring that matches the complete RHS of a production and is used for a reduction

When $S \Rightarrow_{rm}^{*} \alpha X \omega \Rightarrow_{rm} \alpha\beta\omega$ (S is a start symbol),

- For the reduction of $\alpha\beta|\omega$, a production $X \rightarrow \beta$ is used: $\alpha\beta|\omega \Rightarrow_{reduce} \alpha X|\omega$
- $\boldsymbol{\beta}$ **is a handle of** $\boldsymbol{\alpha\beta\omega}$

Handles are always the suffix of a left substring during shift-reduce parsing

- e.g., $S \Rightarrow_{rm}^{*} X\omega \Rightarrow_{rm} \alpha Y \omega \Rightarrow_{rm} \alpha\beta\omega$ (S is a start symbol),
  - β is a handle of $\alpha\beta\omega$, when $\alpha\beta|\omega$
  - The next handle of β is αY (a handle of $\alpha Y\omega$, when $\alpha Y|\omega$)

Definition: Handle

A

a

W

How to address the conflict problems????

- If there is a handle, do reduction
- Otherwise, do shift

Q. How to know whether there is a handle or not??

Handles are always the suffix of a left substring during shift-reduce parsing

- e.g., $S \Rightarrow^{*}_{rm} X\omega \Rightarrow_{rm} \alpha Y \omega \Rightarrow_{rm} \alpha\beta\omega$ (S is a start symbol),
  - β is a handle of $\alpha\beta\omega$, when $\alpha\beta|\omega$
  - The next handle of β is αY (a handle of $\alpha Y \omega$, when $\alpha Y|\omega$)

Definition: Viable prefix

A left substring that can appear during shift-reduce parsing of an acceptable input string

If the left substring is a viable prefix, an input string still has a possibility to be accepted

CFG	State	Viable prefix	Handle	Action
	$\|id * id$			Shift
	$id\| * id$	$\boldsymbol{id}$	$\boldsymbol{id}$	Reduce by $F \rightarrow id$
	$F\| * id$	$\boldsymbol{F}$		Shift
$E \rightarrow T + E\|T$	$F * \|id$	$\boldsymbol{F} *$		Shift
$T \rightarrow F * T\|F,$	$F * id\|$	$\boldsymbol{F * id}$	$\boldsymbol{id}$	Reduce by $F \rightarrow id$
$F \rightarrow (E)\| id$	$F * F\|$	$\boldsymbol{F * F}$	$\boldsymbol{F}$	Reduce by $T \rightarrow F$
	$F * T\|$	$\boldsymbol{F * T}$	$\boldsymbol{F * T}$	Reduce by $T \rightarrow F * T$
	$T\|$	$\boldsymbol{T}$	$\boldsymbol{T}$	Reduce by $E \rightarrow T$
	$E\|$	$\boldsymbol{E}$		Accept

Definition: Viable prefix

A viable prefix is a concatenation of prefixes of RHS of productions

For $\alpha|\beta$, $\alpha = prefix_1 prefix_2 \ldots prefix_n$, where $prefix_i$ is a prefix of a production's RHS.

- $prefix_k$, where $1 \leq k < n$, is not a handle (matches an incomplete RHS of a production)
- If $prefix_n$ is a handle, then it is reduced
- Otherwise, it is concatenated with the reduced output of $prefix_{n+1} prefix_{n+2}$ …

  and eventually becomes a handle

CFG	State	Viable prefix
	$\|id * id$	
	$id\| * id$	$\boldsymbol{id} = a\ prefix\ of\ F \rightarrow \boldsymbol{id}$
	$F\| * id$	$\boldsymbol{F} = a\ prefix\ of\ T \rightarrow \boldsymbol{F} * T$
$E \rightarrow T + E\|T$ $T \rightarrow F * T\|F,$ $F \rightarrow (E)\|\ id$	$F * \|id$	$\boldsymbol{F} * = a\ prefix\ of\ T \rightarrow \boldsymbol{F} * T$
	$F * id\|$	$\boldsymbol{F} * \boldsymbol{id} = a\ prefix\ of\ T \rightarrow \boldsymbol{F} * T\ and\ a\ prefix\ of\ F \rightarrow \boldsymbol{id}$
	$F * F\|$	$\boldsymbol{F} * \boldsymbol{F} = a\ prefix\ of\ T \rightarrow \boldsymbol{F} * T\ and\ a\ prefix\ of\ T \rightarrow \boldsymbol{F}$
	$F * T\|$	$\boldsymbol{F} * \boldsymbol{T} = a\ prefix\ of\ T \rightarrow \boldsymbol{F} * \boldsymbol{T}$
	…	…

Definition: Viable prefix

A

Fo

How to know whether there is a handle or not

For $\alpha\beta|b\omega$ where the left substring $\alpha\beta$ is a viable prefix,

- If there is a production $X \rightarrow \beta$ and αX is a viable prefix,

  then β is a handle of $\alpha\beta b\omega$ and do reduction
- If any suffix of $\alpha\beta$ is not a handle and $\alpha\beta b$ is a viable prefix, then do shift
- Otherwise, reject

Q. How to know whether a given left substring is a viable prefix or not?

$E \rightarrow T + E|T$

$T \rightarrow F * T|F,$

$F \rightarrow (E)|\ id$

$F| * id$	$\boldsymbol{F} = a$ … $\rightarrow \boldsymbol{F} * T$
$F * |id$	$\boldsymbol{F} * = a\ pre$…$ix\ of\ T \rightarrow \boldsymbol{F} * T$
$F * id|$	$\boldsymbol{F} * \boldsymbol{id} = a\ prefix\ of\ T \rightarrow \boldsymbol{F} * T\ and\ a\ prefix\ of\ F \rightarrow \boldsymbol{id}$
$F * F|$	$\boldsymbol{F} * \boldsymbol{F} = a\ prefix\ of\ T \rightarrow \boldsymbol{F} * T\ and\ a\ prefix\ of\ T \rightarrow \boldsymbol{F}$
$F * T|$	$\boldsymbol{F} * \boldsymbol{T} = a\ prefix\ of\ T \rightarrow \boldsymbol{F} * \boldsymbol{T}$
…	…

Definition: Viable prefix

A set of viable prefixes is a regular language

⇒ A viable prefix can be recognized by using a finite automata

For $\alpha|\beta$, $\alpha = prefix_1 prefix_2 \ldots prefix_n$, where $prefix_i$ is a prefix of a production's RHS.

- $prefix_k$, where $1 \le k < n$, is not a handle (matches an incomplete RHS of a production)
- $prefix_n$ is a handle or this is concatenated with the reduced output of $prefix_{n+1} prefix_{n+2} \ldots$ and eventually becomes a handle

e.g., $E \rightarrow T + E|T, \quad T \rightarrow F * T|F, \quad F \rightarrow (E)|\, id$

- if $prefix_1 = T +$
- $Possible\ prefix_2 = T+,\ F *, (:$

  Prefixes of RHS of productions which can appear after $prefix_1$ and not a handle

  …
- $Possible\ prefix_n = \ldots$

  Prefixes of RHS of productions which can appear after $prefix_{n-1}$ and can be a handle

Definition: Viable prefix

A set of viable prefixes is a regular language

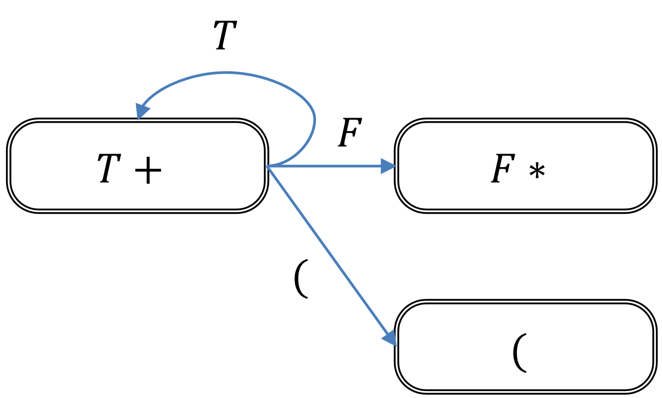
⇒ A viable prefix can be recognized by using a finite automata

e.g., $E \rightarrow T + E|T, \quad T \rightarrow F * T|F, \quad F \rightarrow (E)|\ id$

- For $\alpha|\beta$, $\alpha = prefix_1 prefix_2 \ldots prefix_n$, if $prefix_1 = T +$
- $Possible\ prefix_2 = T+,\ F *, ($:

  Prefixes of RHS of productions which can appear after $prefix_1$ and not a handle

  …



Definition: Viable prefix

A set of viable prefixes is a regular language

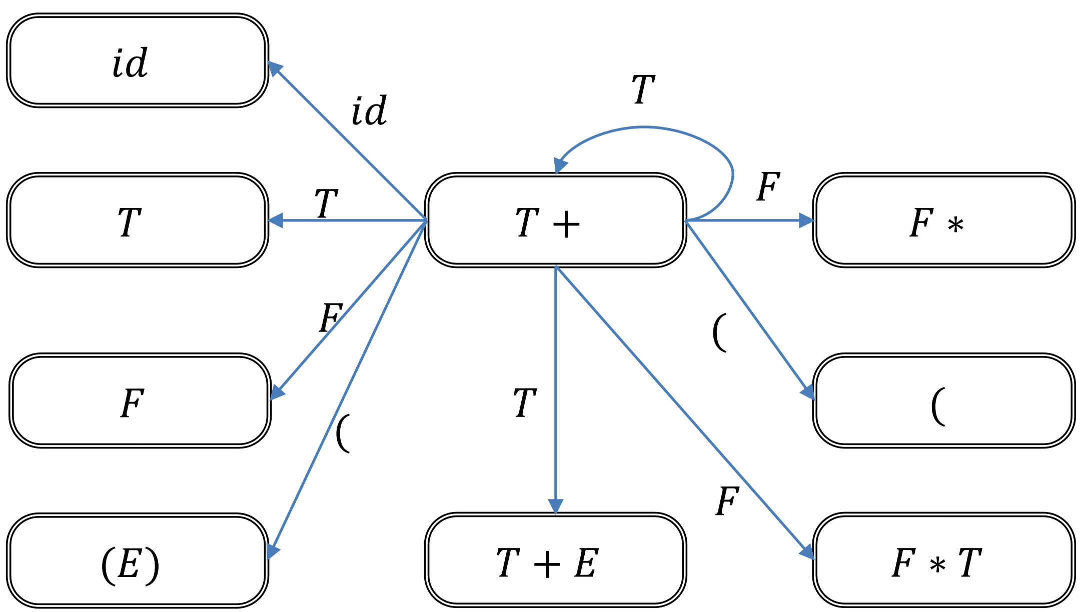
⇒ A viable prefix can be recognized by using a finite automata

e.g., $E \rightarrow T + E|T, \quad T \rightarrow F * T|F, \quad F \rightarrow (E)|\, id$

- For $\alpha|\beta$, $\alpha = prefix_1 prefix_2 \dots prefix_n$, if $\boldsymbol{prefix_{n-1}} = T +$
- $\boldsymbol{Possible\ prefix_n} = T+,\ F *, (, \boldsymbol{T, F, (E), id, T + E, F * T}$:

  Prefixes of RHS of productions which can appear after $prefix_{n-1}$ and can be a handle

  ...



Definition: Viable prefix

A set of viable prefixes is a regular language

⇒ A viable prefix can be recognized by using a finite automata

e.g., $E \to$

- if *pr*
- ***Poss***

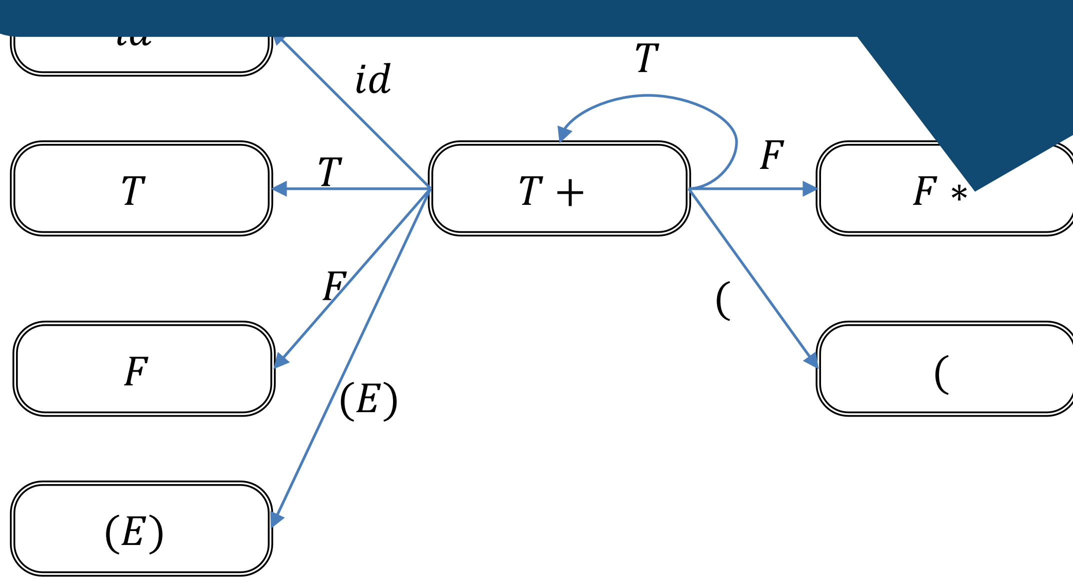
  Prefi

  …

The number of states is finite!

The number of transitions is finite!

So, we can recognize a viable prefix by using a finite automata



NFA recognizing viable prefixes

Step 1. Make items for each production

Definition: an item is a production with a "." somewhere on the RHS of the production

- A production $E \rightarrow T + E$ has four items as follows:

$$E \rightarrow .T + E$$

$$E \rightarrow T.+E$$

$$E \rightarrow T+.E$$

$$E \rightarrow T + E.$$

- A production $A \rightarrow \epsilon$ has one item as follows:

$$A \rightarrow .$$

- Item $T \rightarrow (E.)$ says that so far we have seen "$(E$" of the handle (E) and hope to see ")"

NFA recognizing viable prefixes

Step 2. Add a dummy production $S' \rightarrow S$ to a CFG $\boldsymbol{G}$, where S is the start symbol of G

- Items for $S' \rightarrow S$ is also created

Step 3. Construct a NFA

- Alphabet = a set of non-terminals and terminals of G
- States = the items of G
- A start state = $S' \rightarrow .S$, accepting states = all states

Step 4. For item $\boldsymbol{E \rightarrow \alpha . X\beta}$, add transition $\boldsymbol{\delta(E \rightarrow \alpha . X\beta, X) = E \rightarrow \alpha X . \beta}$

Step 5. For item $\boldsymbol{E \rightarrow \alpha . X\beta}$ and production $\boldsymbol{X \rightarrow \omega}$,

add transition $\boldsymbol{\delta(E \rightarrow \alpha . X\beta, \epsilon) = X \rightarrow . \omega}$

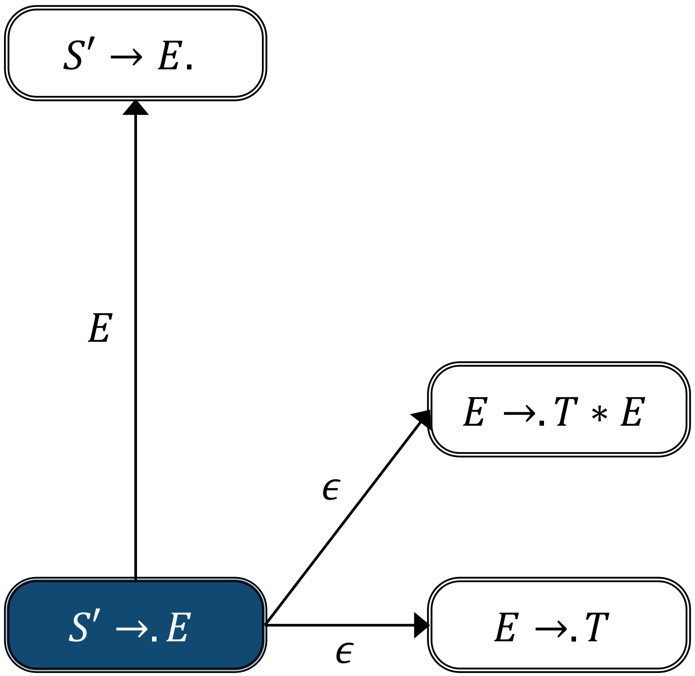
NFA recognizing viable prefixes

$$S' \to E, \qquad E \to T * E | T, \qquad T \to (E) | id$$

$\boxed{S' \to .E}$

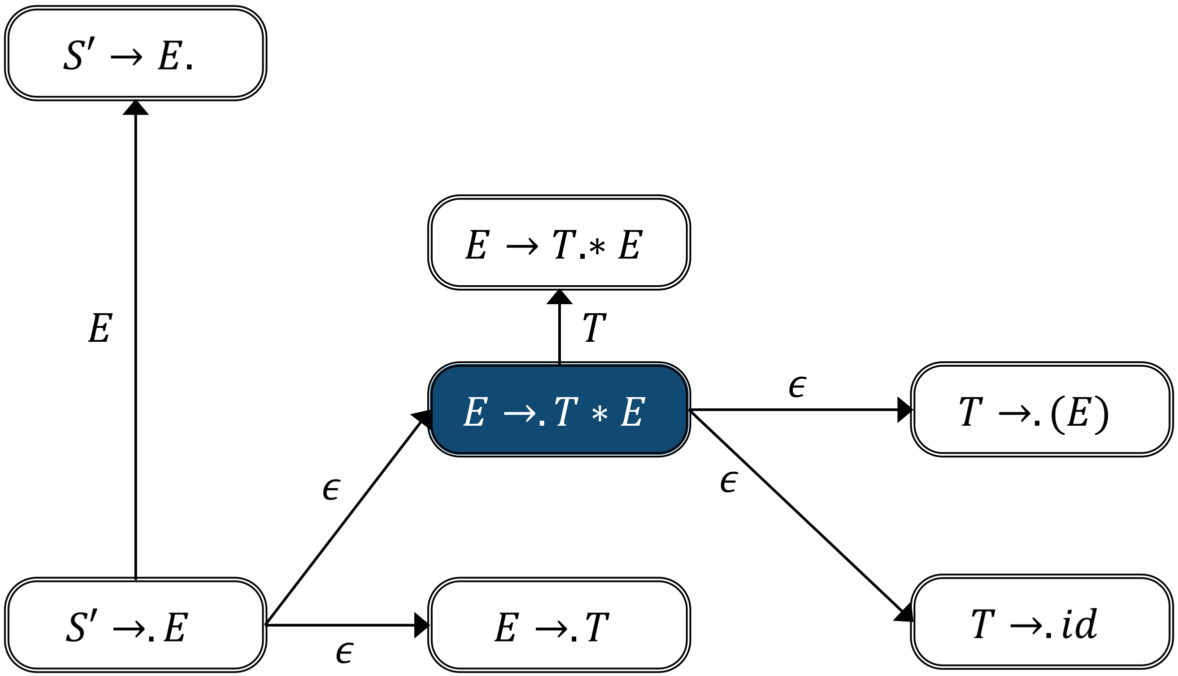
NFA recognizing viable prefixes

$$S' \rightarrow E, \qquad E \rightarrow T * E | T, \qquad T \rightarrow (E) | id$$



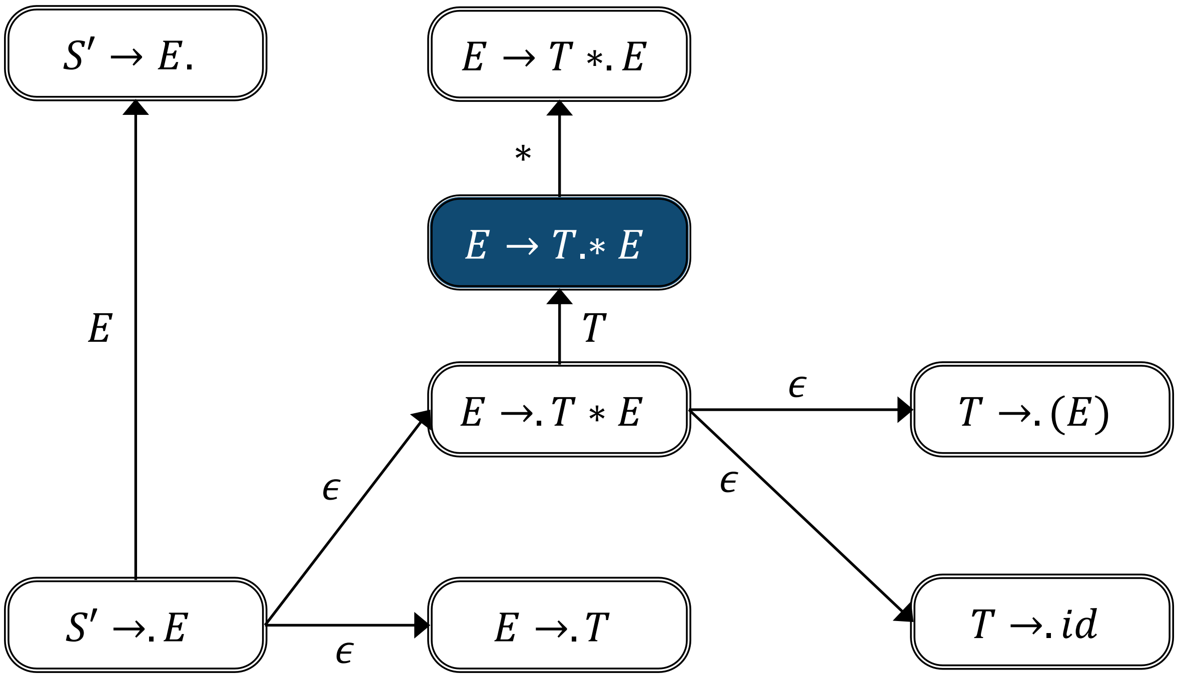
NFA recognizing viable prefixes

$$S' \to E, \qquad E \to T * E | T, \qquad T \to (E) | id$$



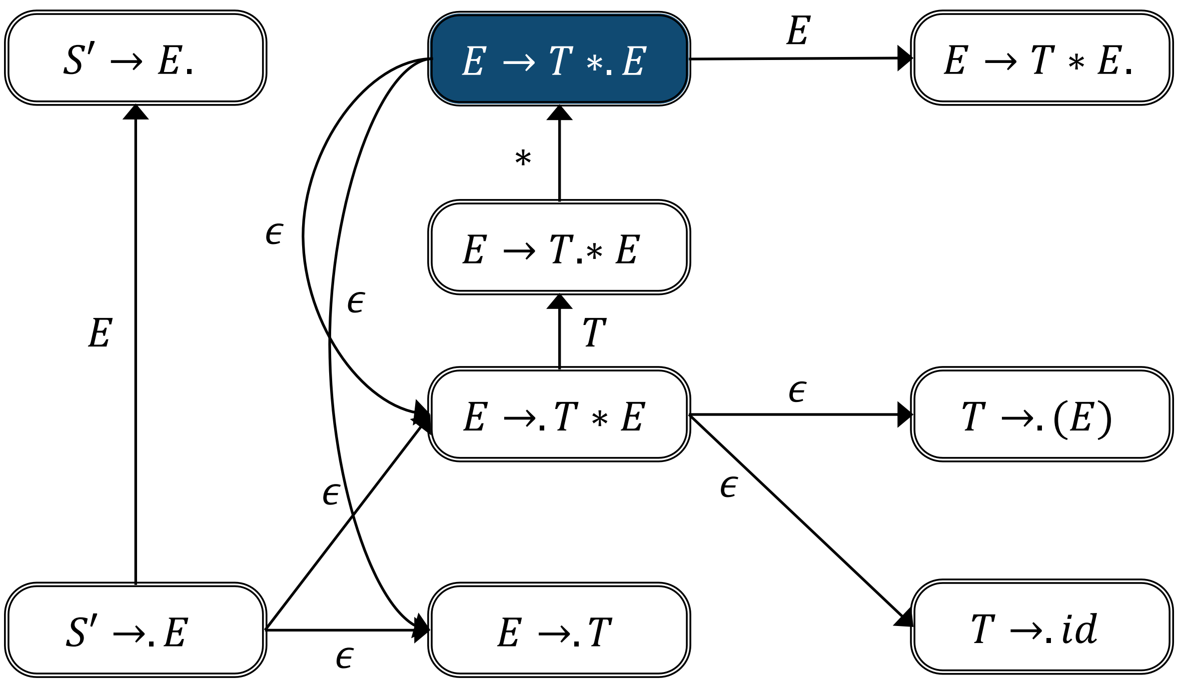
NFA recognizing viable prefixes

$$S' \to E, \qquad E \to T * E | T, \qquad T \to (E) | id$$



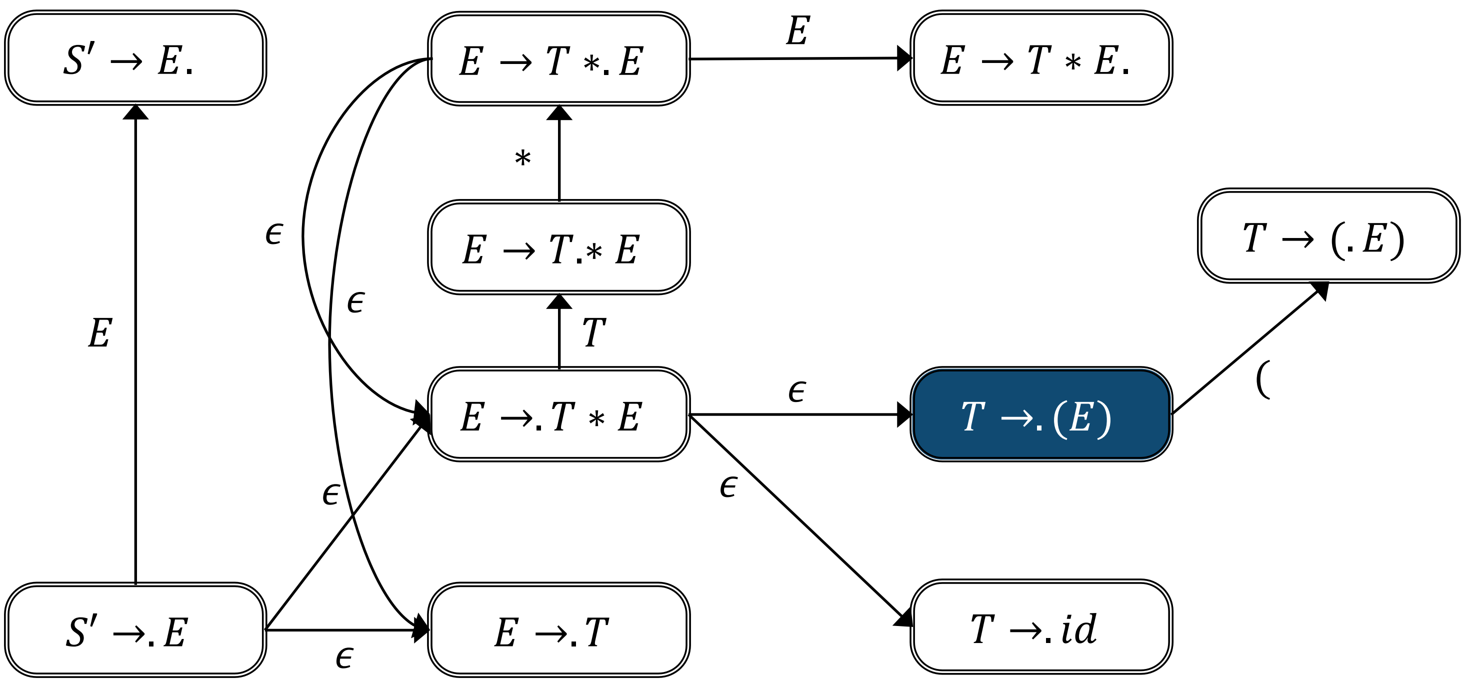
NFA recognizing viable prefixes

$$S' \to E, \qquad E \to T * E | T, \qquad T \to (E) | id$$



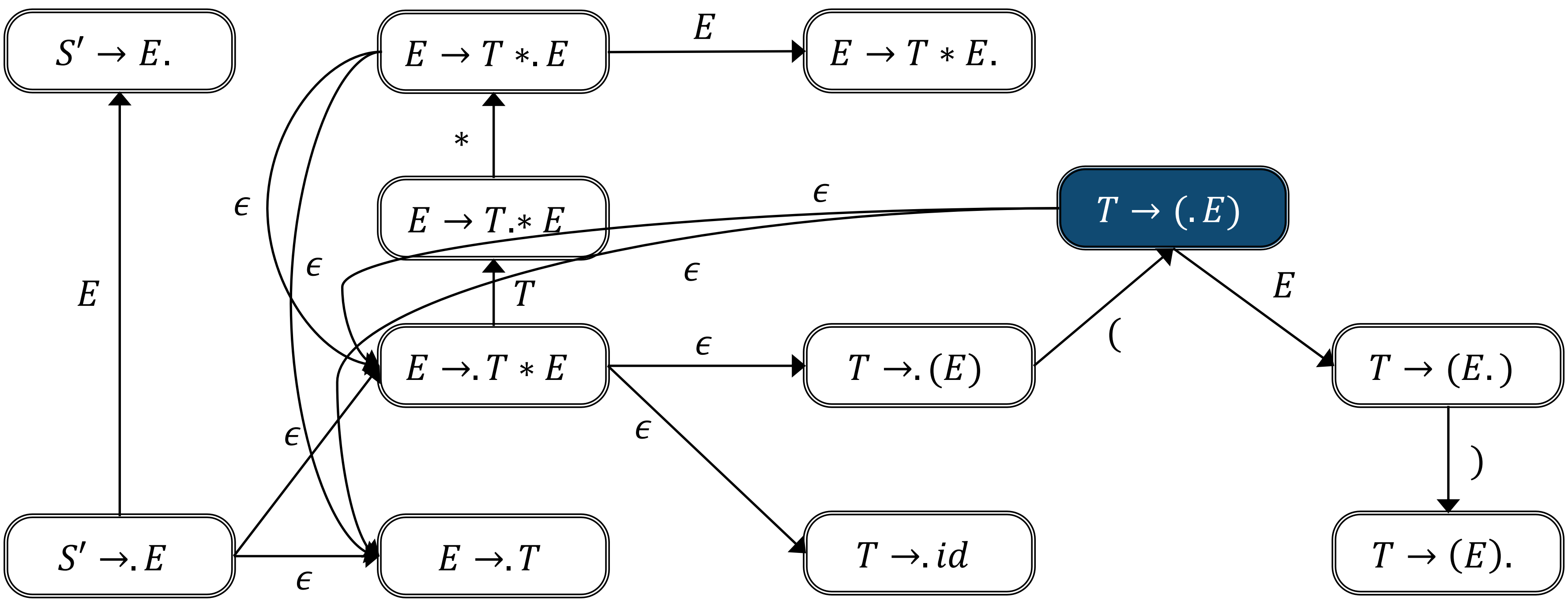
NFA recognizing viable prefixes

$$S' \rightarrow E, \qquad E \rightarrow T * E | T, \qquad T \rightarrow (E) | id$$



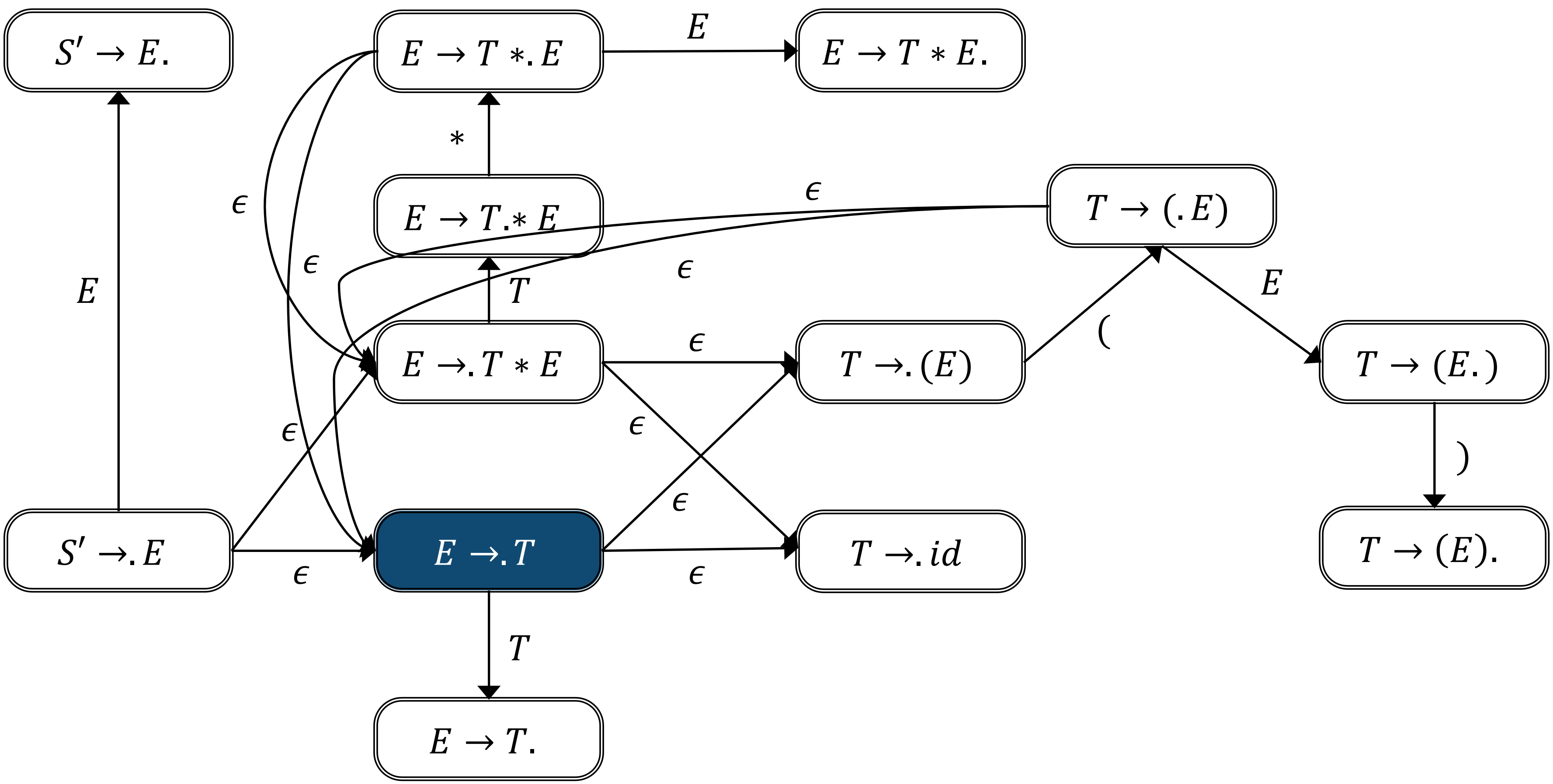
NFA recognizing viable prefixes

$$S' \to E, \qquad E \to T * E | T, \qquad T \to (E) | id$$



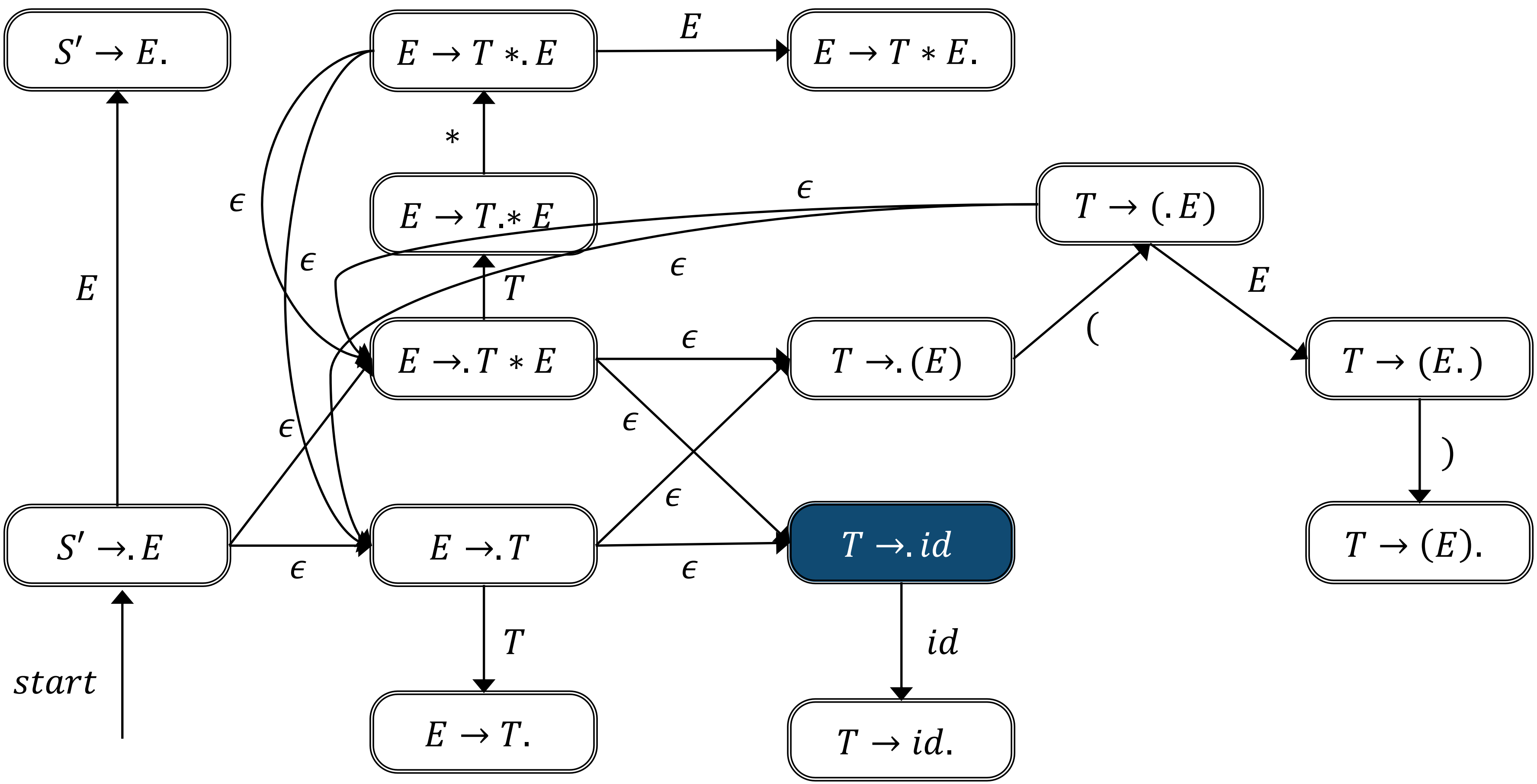
NFA recognizing viable prefixes

$$S' \to E, \qquad E \to T * E | T, \qquad T \to (E) | id$$



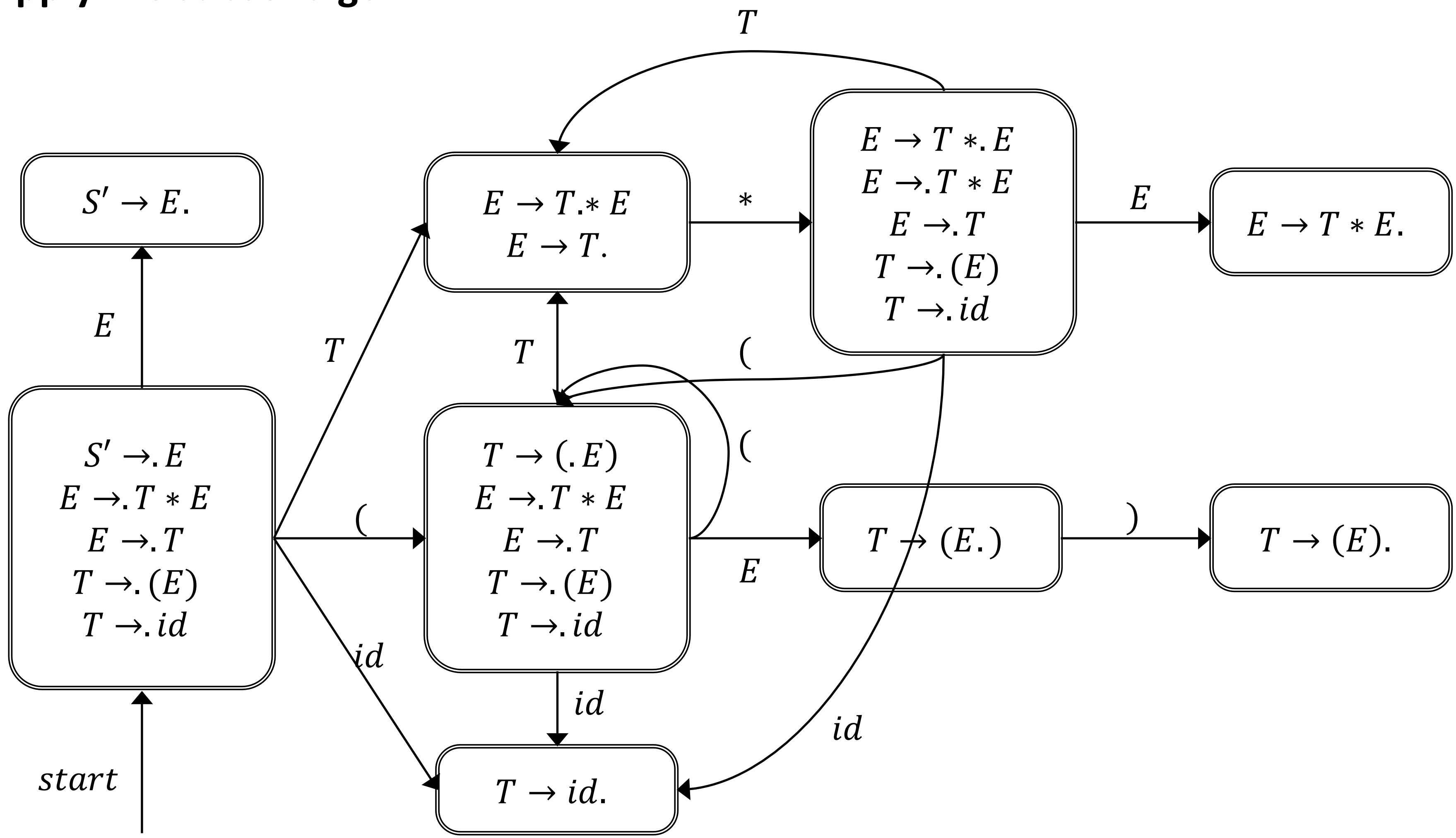
NFA recognizing viable prefixes

$$S' \to E, \qquad E \to T * E | T, \qquad T \to (E) | id$$



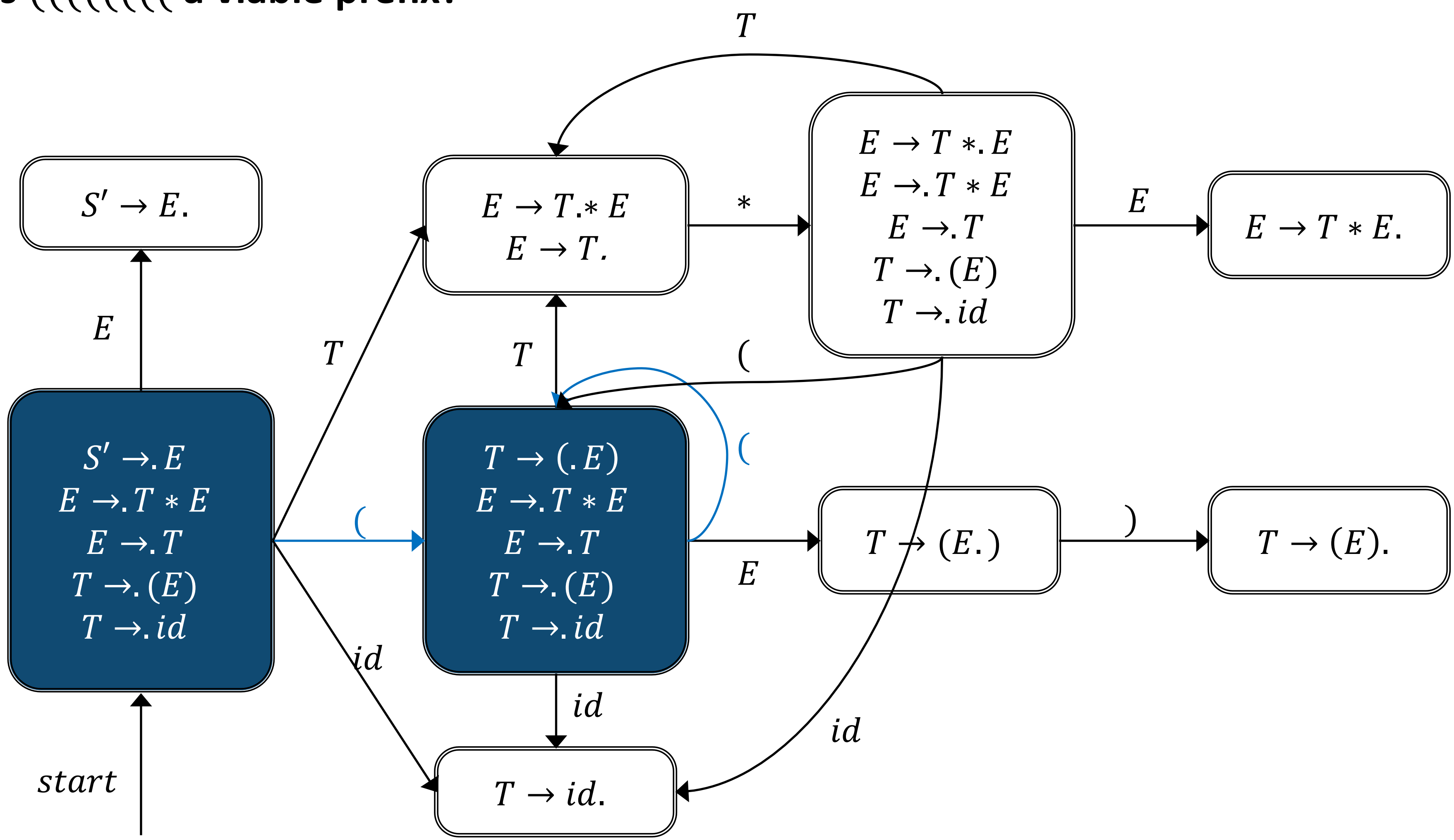
DFA recognizing viable prefixes

Apply the subset algorithm!!



Examples of recognizing viable prefixes

Is ((((((((a viable prefix?



LR(0) parsing with DFA

When $\boldsymbol{\alpha|b\omega}$ and DFA terminates in state $\boldsymbol{q_i}$ with $\boldsymbol{\alpha}$,

- **Reduce by $\boldsymbol{X \rightarrow \beta}$** if q_i contains item $X \rightarrow \beta.$, where β is a suffix of α
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

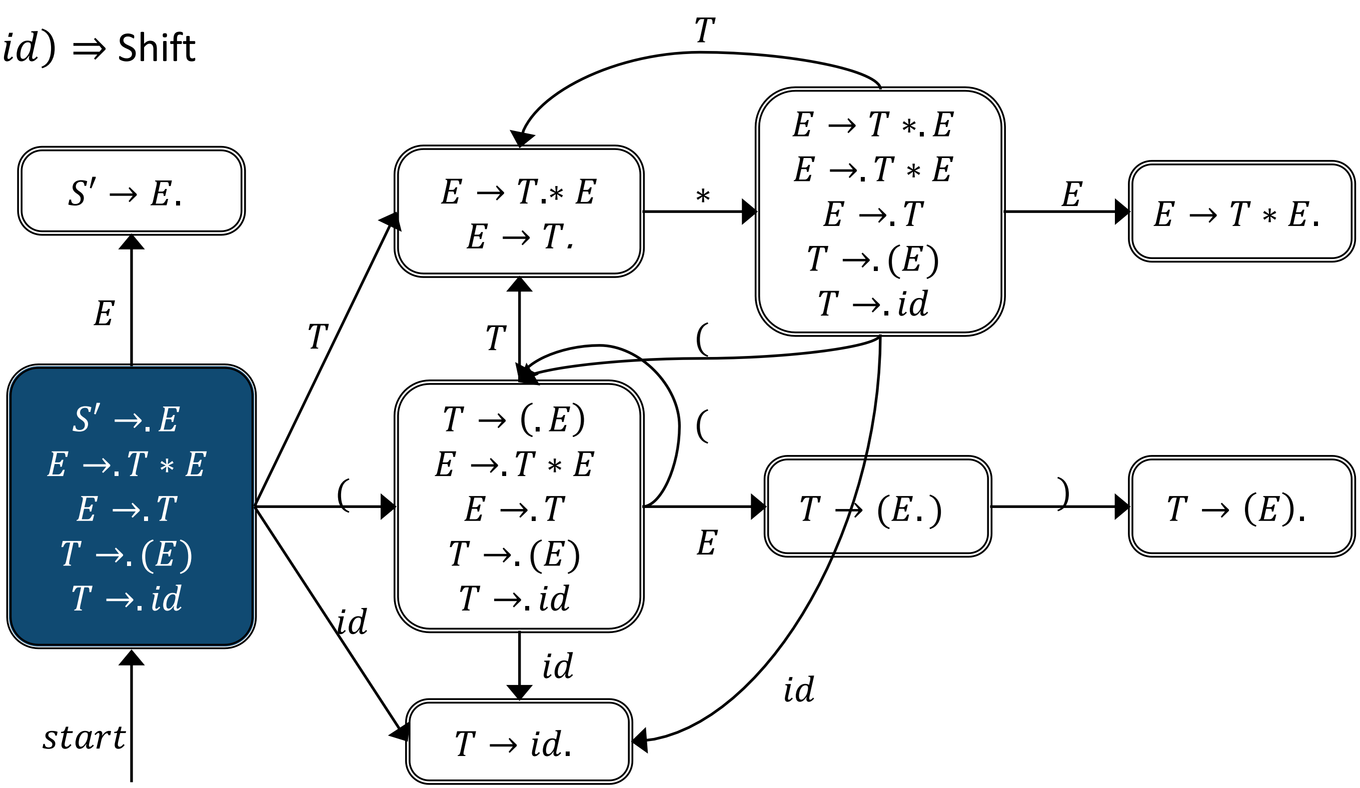
LR(0) parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce** if q_i contains item $X \rightarrow \alpha.$
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$|(id * id) \Rightarrow$ Shift



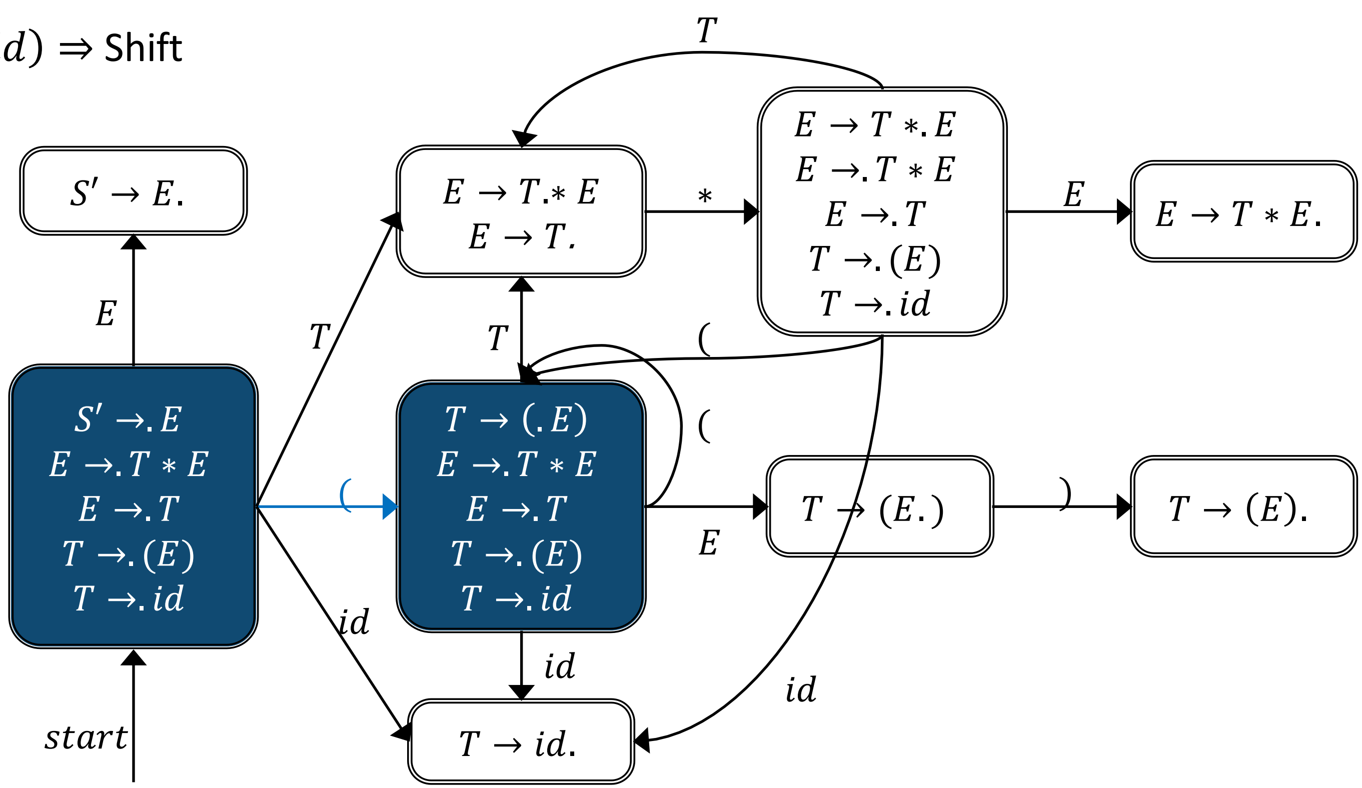
LR(0) parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce** if q_i contains item $X \rightarrow \alpha.$
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$(|id * id) \Rightarrow$ Shift



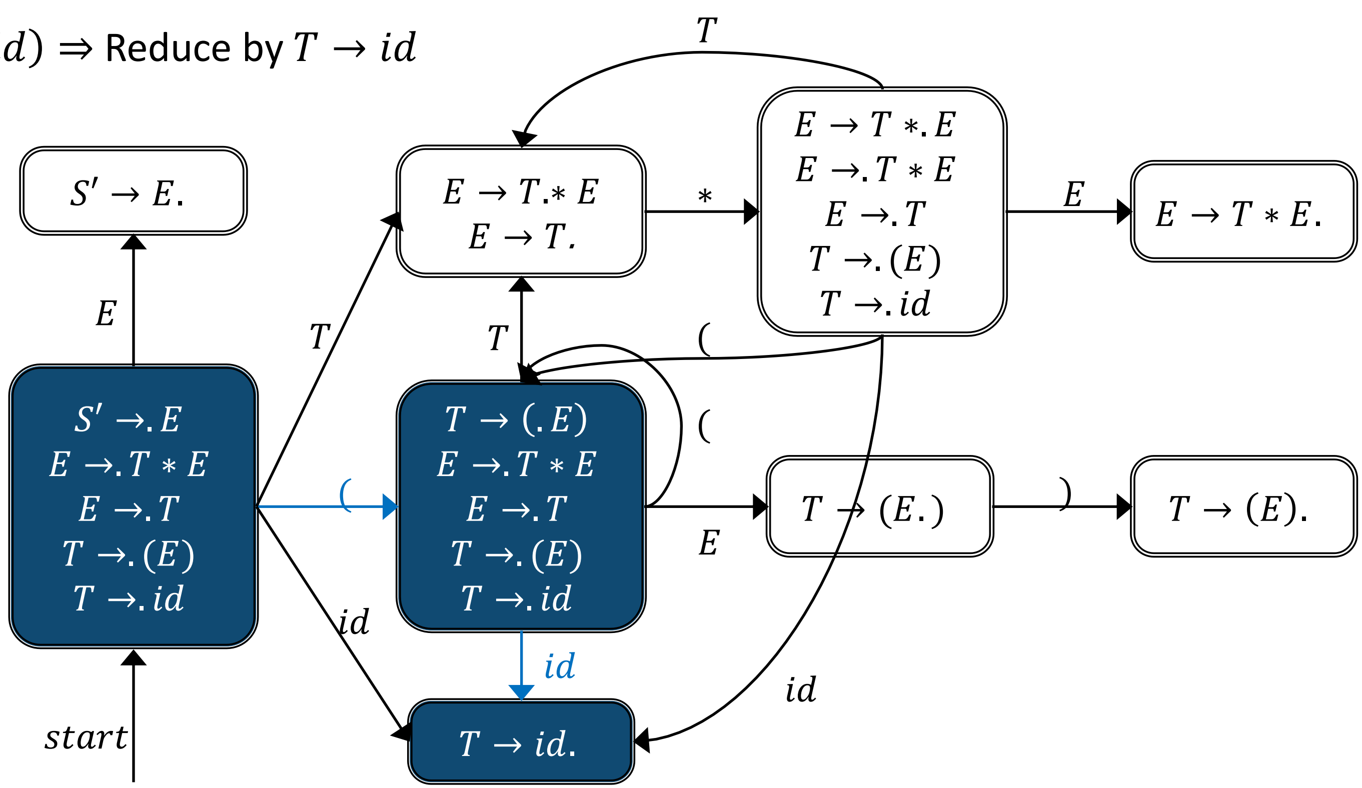
LR(0) parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce** if q_i contains item $X \to \alpha.$
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$(id| * id) \Rightarrow$ Reduce by $T \to id$



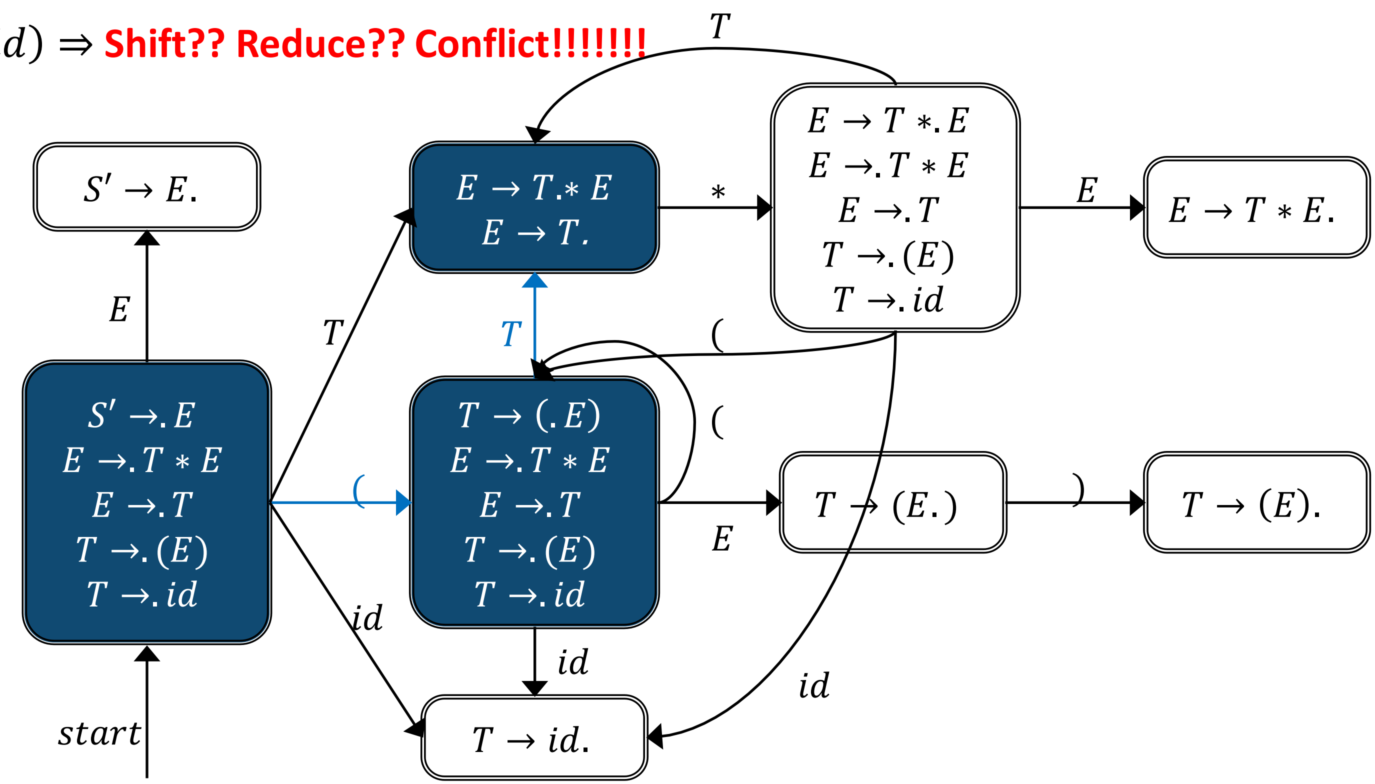
LR(0) parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce** if q_i contains item $X \to \alpha.$
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$(T| * id) \Rightarrow$ **Shift?? Reduce?? Conflict!!!!!!!**



SLR parsing with DFA

When $\boldsymbol{\alpha}|\boldsymbol{b\omega}$ and DFA terminates in state $\boldsymbol{q_i}$ with $\boldsymbol{\alpha}$,

- **Reduce by $\boldsymbol{X \rightarrow \beta}$** if q_i contains item $X \rightarrow \beta$. and $\boldsymbol{b \in Follow(X)}$, where β is a suffix of α
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

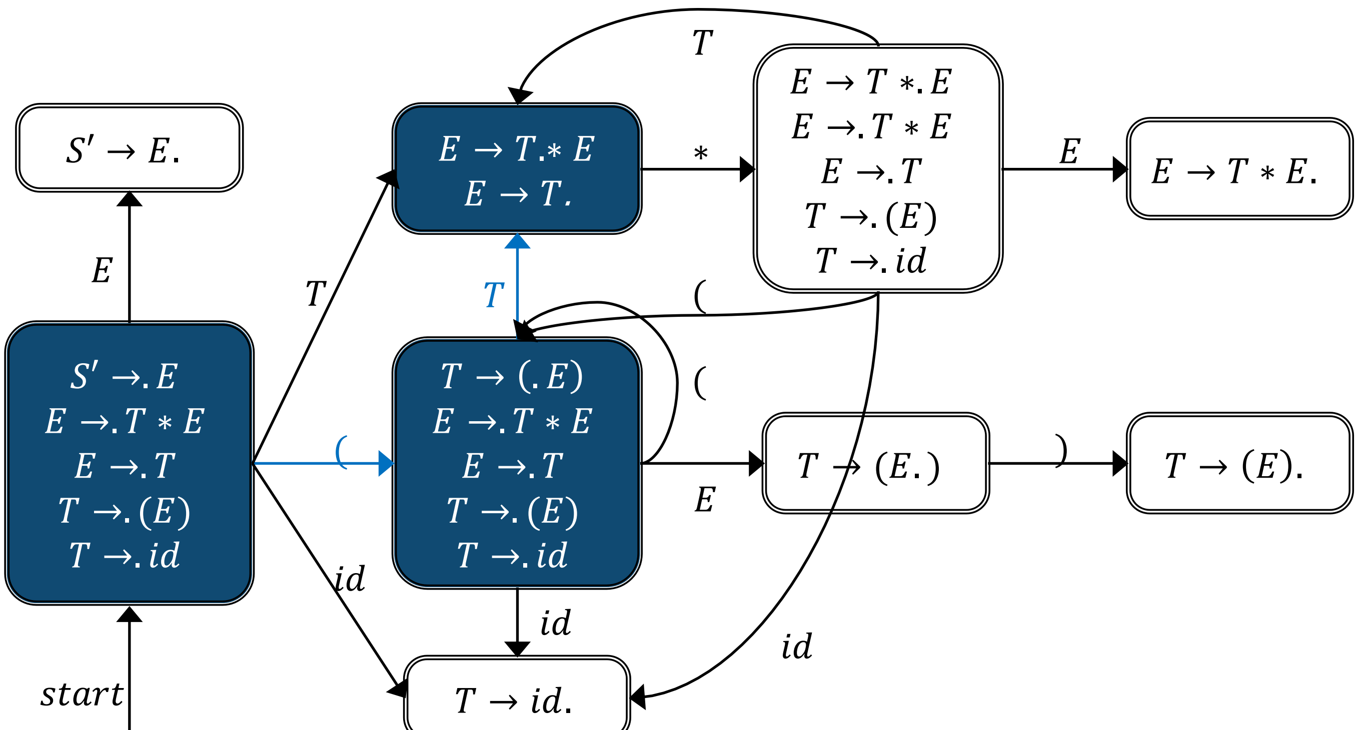
SLR parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce by $X \rightarrow \beta$** if q_i contains item $X \rightarrow \beta.$ and $b \in Follow(X)$, where β is a suffix of α
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$(T| * id)\$ \Rightarrow$ **Shift because** $* \notin Follow(E) = \{), \$\}$



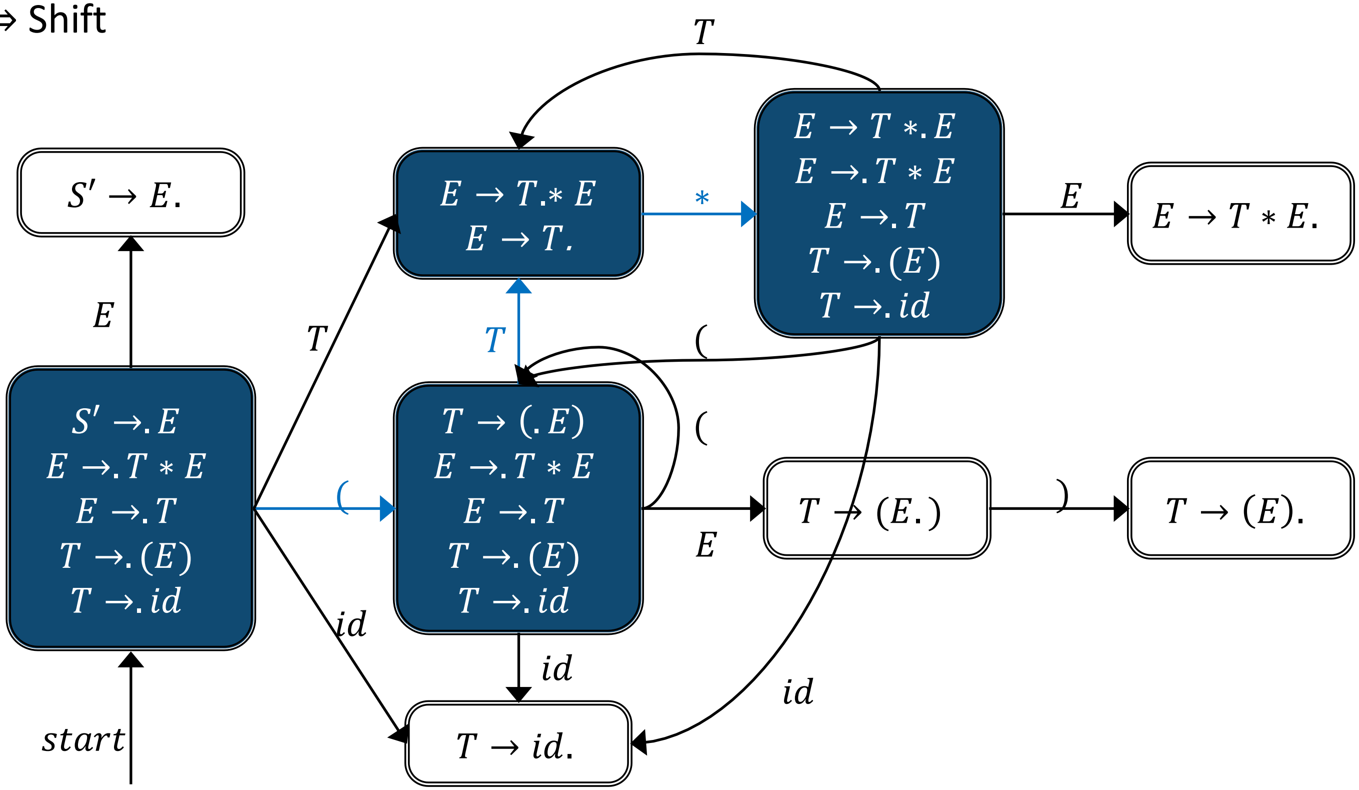
SLR parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce by $X \to \beta$** if q_i contains item $X \to \beta.$ and $b \in Follow(X)$, where β is a suffix of α
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$(T * |id)\$ \Rightarrow$ Shift



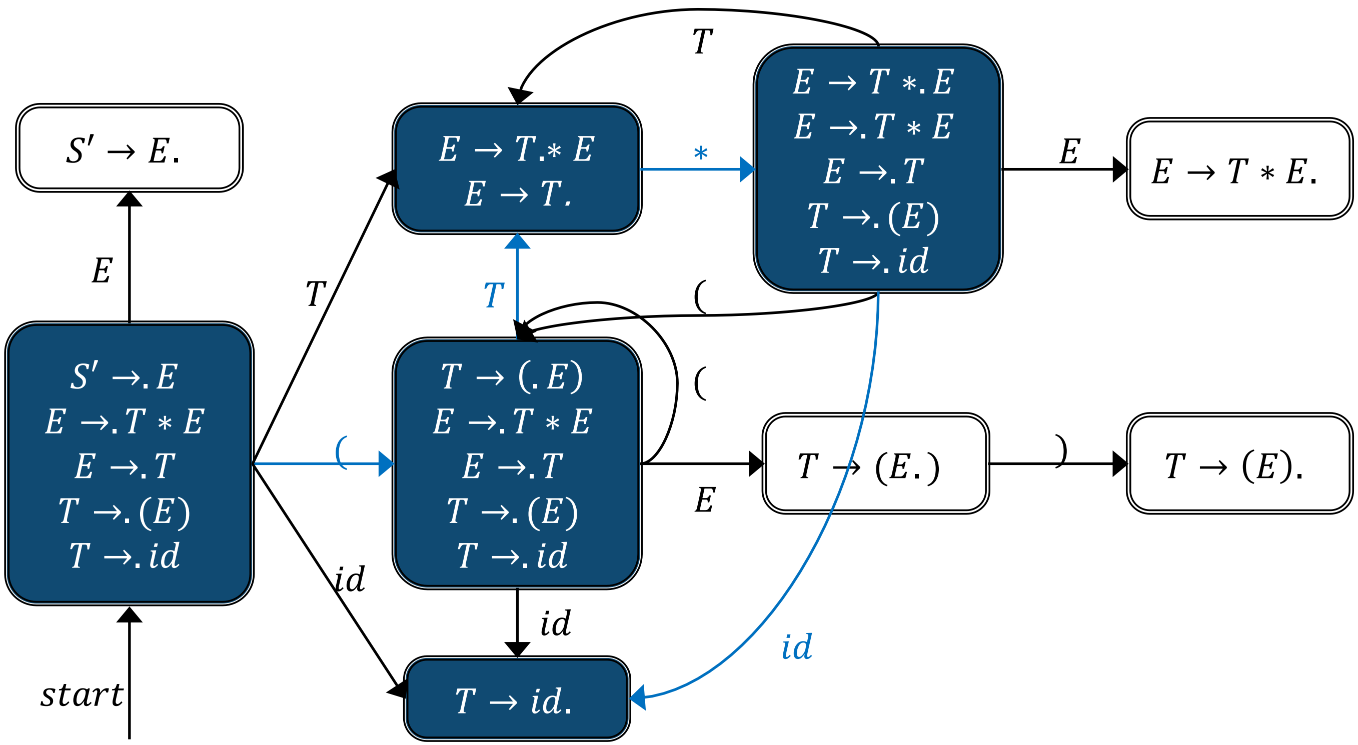
SLR parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce by $X \to \beta$** if q_i contains item $X \to \beta.$ and $b \in Follow(X)$, where β is a suffix of α
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$(T * id|)\$ \Rightarrow$ Reduce by $T \to id$ **because** $) \in Follow(T) = \{*,), \$\}$



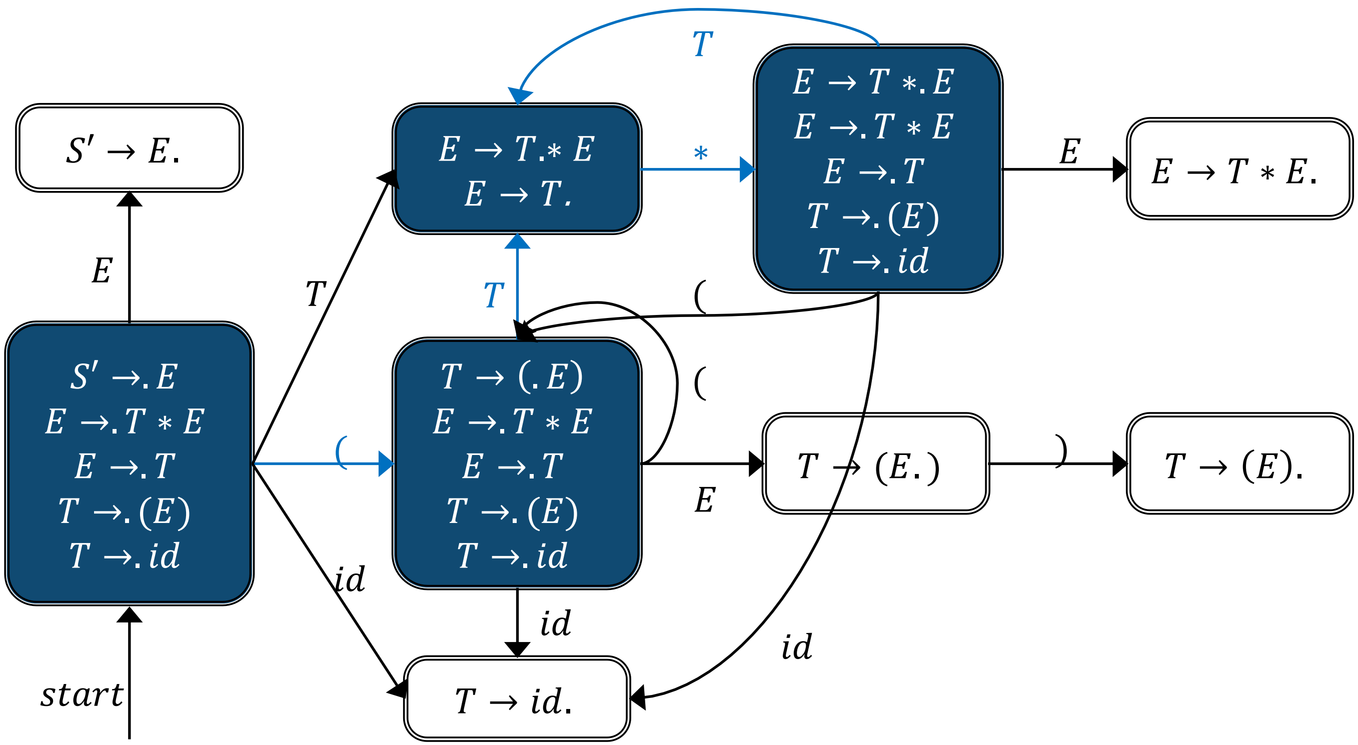
SLR parsing with DFA

When $\boldsymbol{\alpha|b\omega}$ and DFA terminates in state $\boldsymbol{q_i}$ with $\boldsymbol{\alpha}$,

- **Reduce by $\boldsymbol{X \rightarrow \beta}$** if q_i contains item $X \rightarrow \beta.$ and $\boldsymbol{b \in Follow(X)}$, where β is a suffix of α
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$(T * T|)\$ \Rightarrow$ Reduce by $E \rightarrow T$ **because** $) \in \boldsymbol{Follow(E)} = \{), \$\}$



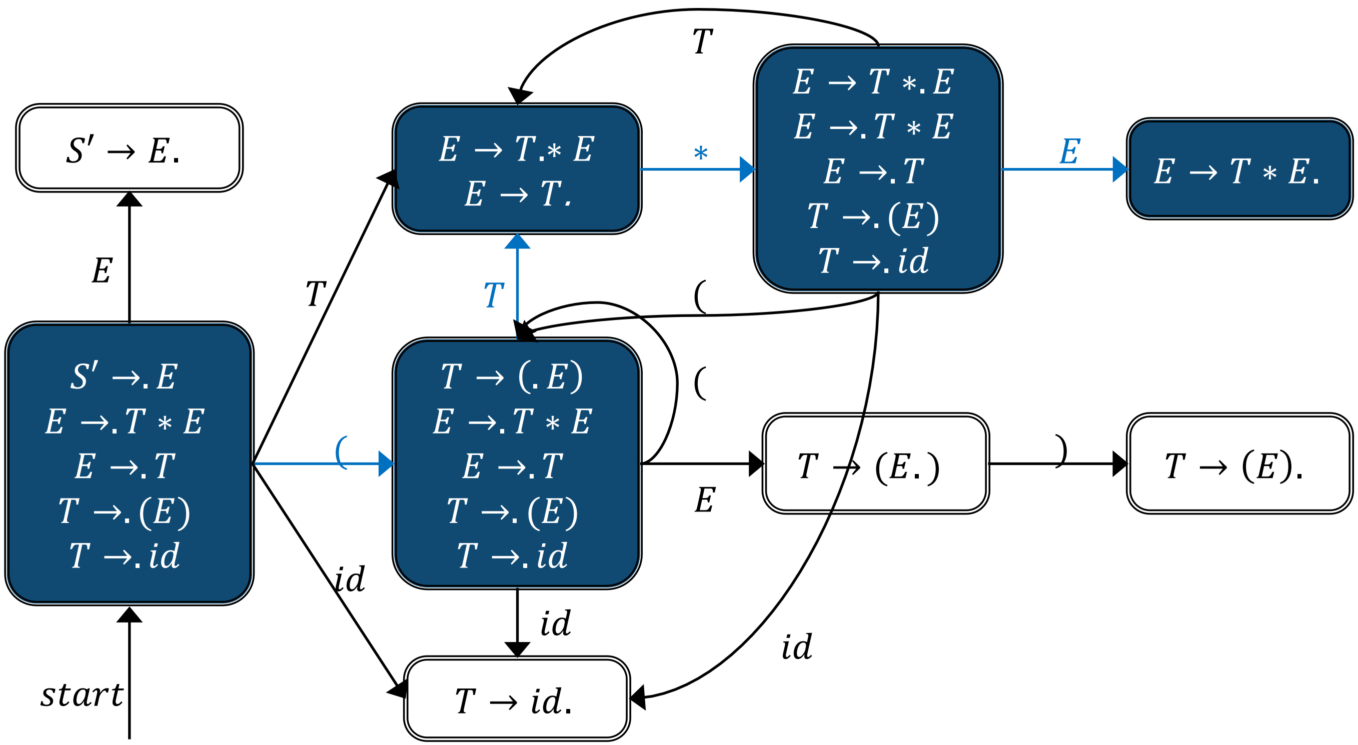
SLR parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce by $X \to \beta$** if q_i contains item $X \to \beta.$ and $b \in Follow(X)$, where β is a suffix of α
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$(T * E|)\$ \Rightarrow$ Reduce by $E \to T * E$ **because** $) \in Follow(E) = \{), \$\}$



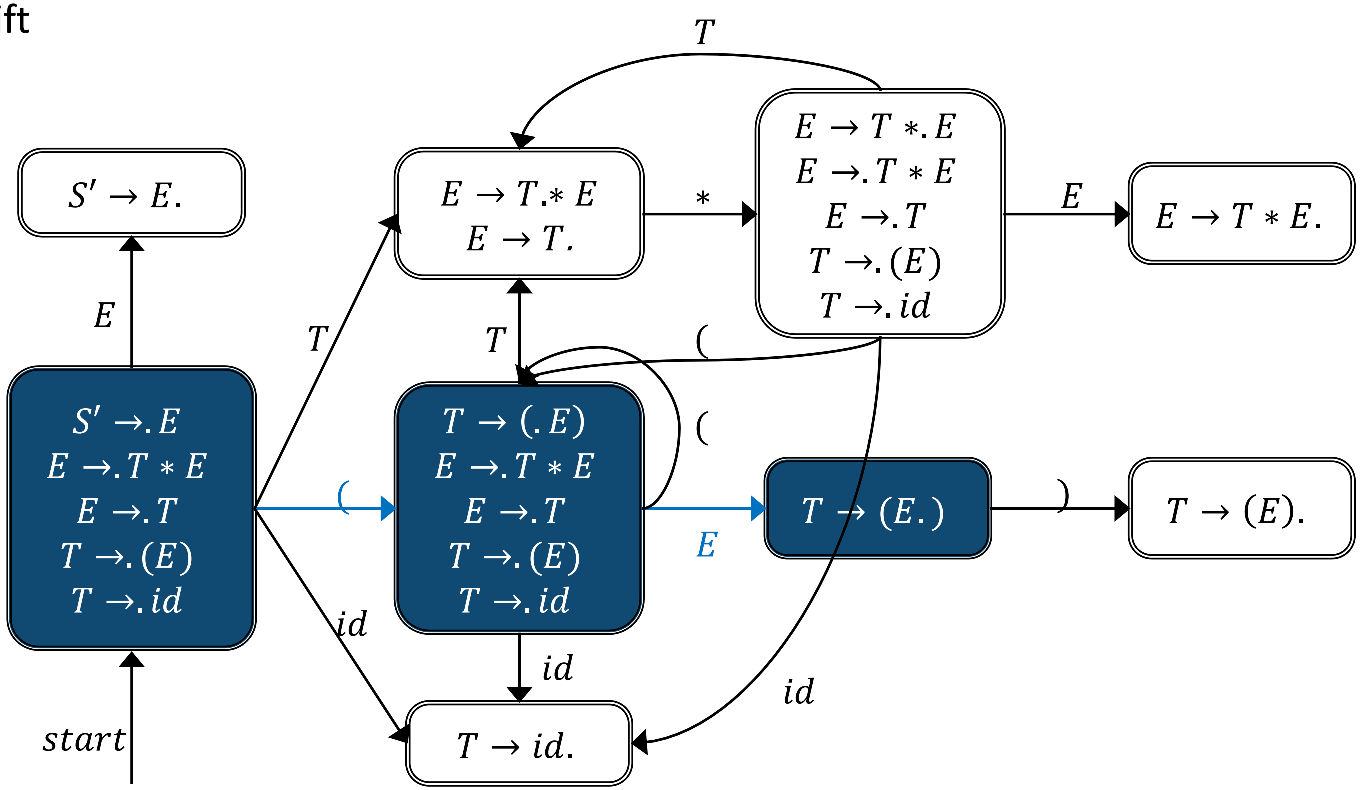
SLR parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce by $X \to \beta$** if q_i contains item $X \to \beta.$ and $b \in Follow(X)$, where β is a suffix of α
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$(E|)\$ \Rightarrow$ Shift



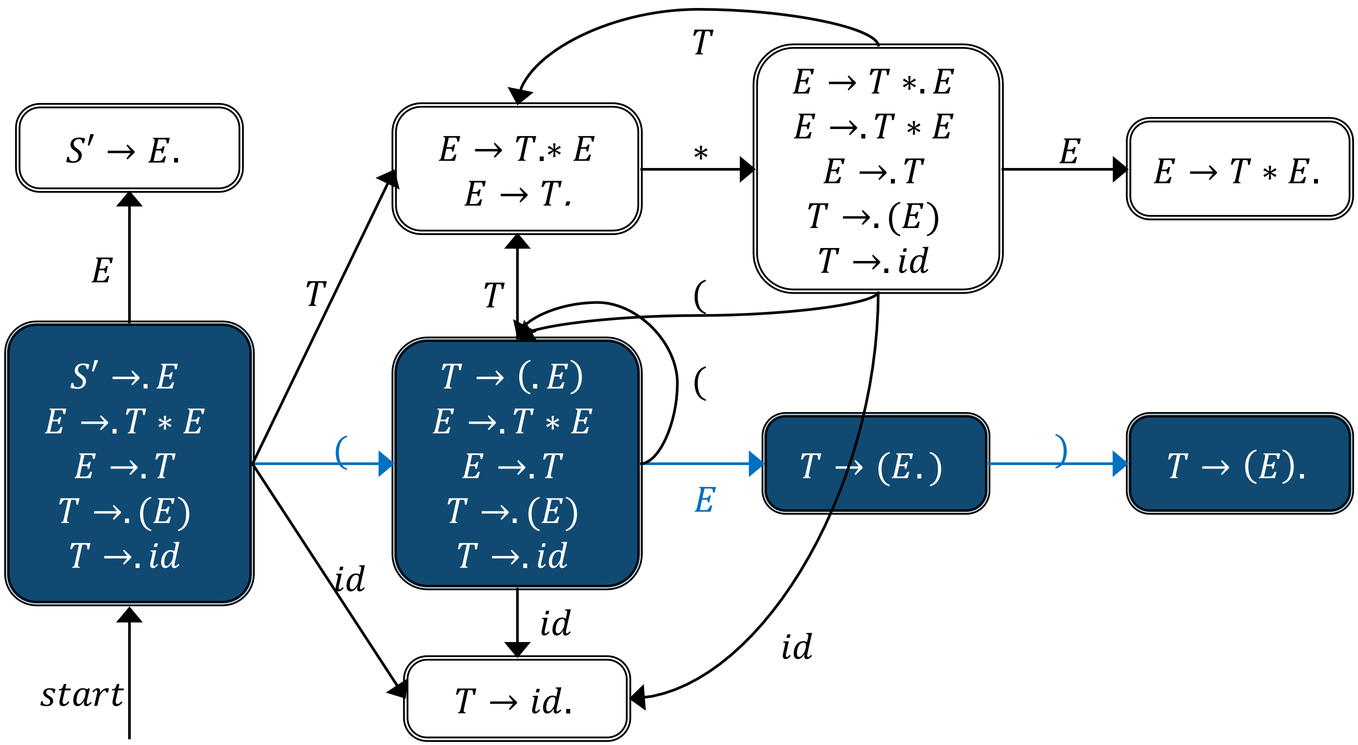
SLR parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce by $X \rightarrow \beta$** if q_i contains item $X \rightarrow \beta.$ and $b \in Follow(X)$
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$(E)|\$ \Rightarrow$ Reduce by $T \rightarrow (E)$ **because** $\$ \in Follow(T) = \{*,), \$\}$



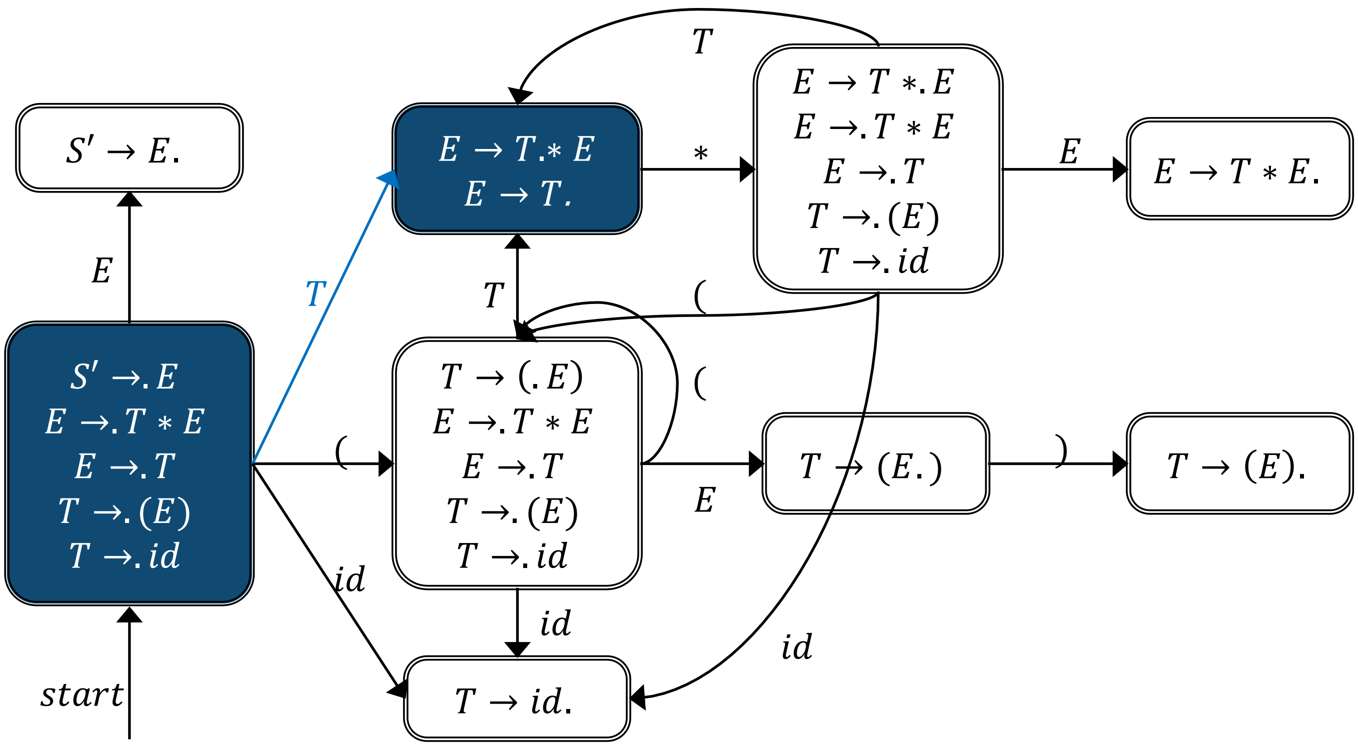
SLR parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce by $X \rightarrow \beta$** if q_i contains item $X \rightarrow \beta.$ and **$b \in Follow(X)$**
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$T|\$ \Rightarrow$ Reduce by $E \rightarrow T$ **because $\$ \in Follow(E) = \{), \$\}$**



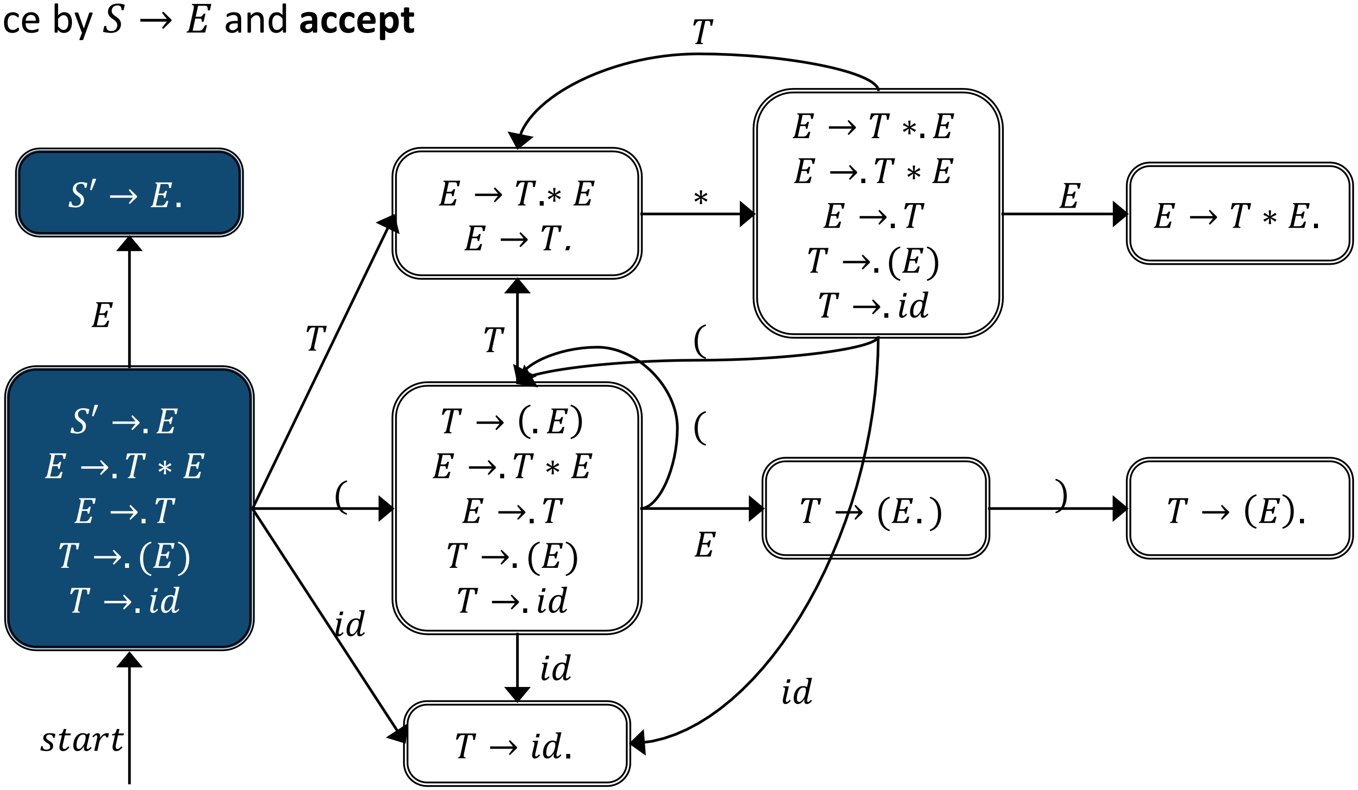
SLR parsing with DFA

When $\alpha|b\omega$ and DFA terminates in state q_i with α,

- **Reduce by $X \rightarrow \beta$** if q_i contains item $X \rightarrow \beta.$ and $b \in Follow(X)$
- **Shift** if q_i has a transition on an input symbol b, **reject** otherwise

For an input $(id * id)$,

$E|\$ \Rightarrow$ Reduce by $S \rightarrow E$ and **accept**



SLR parsing with DFA

If there are still conflicts, the grammar is not a SLR context free grammar

(e.g., ambiguous grammars)

For an ambiguous grammar: $E \rightarrow E + E|E * E|id$ and an input $id + id\ *\ id$

- When $E +\ E|*\ id$, reduce will be selected because $* \in Follow(E)$
- But, this is not what we wanted (we want to do shift)

- Solution: precedence declaration
  - e.g., Declare **"* has a higher precedence (priority) than +"**
    - If there is a conflict between * and +, do the operation related with *

Summary of bottom-up parsing

Constructs a parse tree for an input string, starting from the leaves (input strings)

and working up towards the root (the start symbol)

It traces **a right derivation** of the input string **in reverse: "reduction"**

But, there are still two types of conflicts

1. **Shift-reduce conflict**
2. **Reduce-reduce conflict**

Q. How to address these problems????

Handle, viable prefix, NFA, DFA, LR(0), SLR parsing…