Part IV.
Syntax Analysis: Models

Context-Free Grammar (CFG)

Gist: A *grammar* is based on a finite set of grammatical rules, by which it generates strings of its language.

Illustration:

Start nonterminal

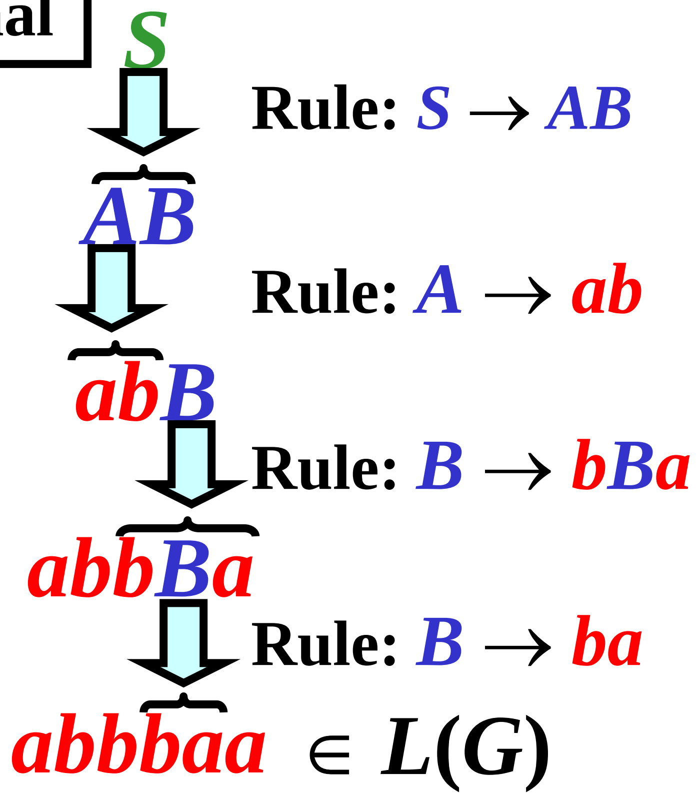
Grammar *G*:

Nonterminals: $A, B, \boxed{S}$

Terminals: a, b, c, d

Rules:

$S \to AB,$
$A \to aAb,$
$A \to ab,$
$B \to bBa,$
$B \to ba$



S

Rule: $S \to AB$

AB

Rule: $A \to ab$

abB

Rule: $B \to bBa$

$abbBa$

Rule: $B \to ba$

$abbbaa \in L(G)$

Context-Free Grammar: Definition

Definition: *A context-free grammar* (CFG) is a quadruple $G = (N, T, P, S)$, where

- N is an alphabet of *nonterminals*
- T is an alphabet of *terminals*, $N \cap T = \emptyset$
- P is a finite set of *rules* of the form $A \rightarrow x$, where $A \in N$, $x \in (N \cup T)^*$
- $S \in N$ is the *start nonterminal*

Mathematical Note on Rules:

- Strictly mathematically, P is a relation from N to $(N \cup T)^*$
- Instead of $(\boldsymbol{A}, \boldsymbol{x}) \in P$, we write $\boldsymbol{A} \rightarrow \boldsymbol{x} \in P$

- $\boldsymbol{A} \rightarrow \boldsymbol{x}$ means that $\boldsymbol{A}$ can be replaced with $\boldsymbol{x}$
- $\boldsymbol{A} \rightarrow \varepsilon$ is called ***ε-rule***

Convention

- $\boldsymbol{A}, \ldots, \boldsymbol{F}, \boldsymbol{S}$: nonterminals
- $\boldsymbol{S}$: the start nonterminal
- $\boldsymbol{a}, \ldots, \boldsymbol{d}$: terminals
- $\boldsymbol{U}, \ldots, \boldsymbol{Z}$: members of $(N \cup T)$
- $\boldsymbol{u}, \ldots, \boldsymbol{z}$: members of $(N \cup T)^*$
- π : sequence of productions

A subset of rules of the form:

$$\boldsymbol{A} \rightarrow \boldsymbol{x}_1, \boldsymbol{A} \rightarrow \boldsymbol{x}_2, \ldots, \boldsymbol{A} \rightarrow \boldsymbol{x}_n$$

can be simply written as:

$$\boldsymbol{A} \rightarrow \boldsymbol{x}_1 \mid \boldsymbol{x}_2 \mid \ldots \mid \boldsymbol{x}_n$$

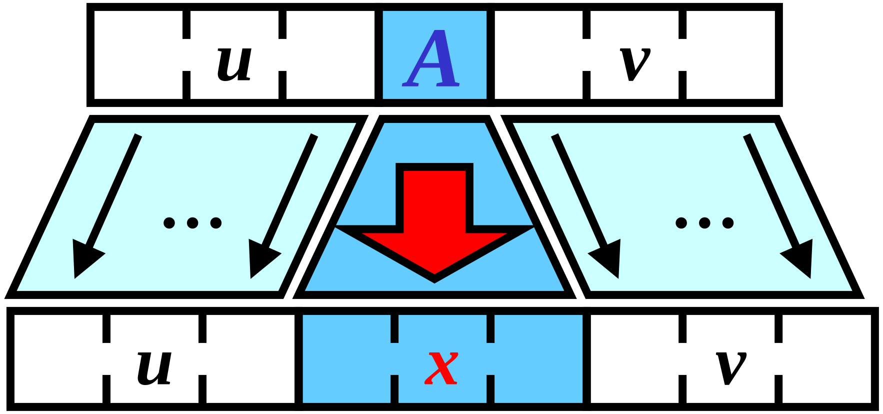
Derivation Step

Gist: A change of a string by a rule.

Definition: Let $G = (N, T, P, S)$ be a CFG. Let $\boldsymbol{u}, \boldsymbol{v} \in (N \cup T)^*$ and $\boldsymbol{p} = \boldsymbol{A} \rightarrow \mathbf{x} \in P$. Then, $\boldsymbol{uAv}$ *directly derives* $\boldsymbol{uxv}$ *according to* $\boldsymbol{p}$ in G, written as $\boldsymbol{uAv} \Rightarrow \boldsymbol{uxv}$ $[\boldsymbol{p}]$ or, simply, $\boldsymbol{uAv} \Rightarrow \boldsymbol{uxv}$.

Note: If $\boldsymbol{uAv} \Rightarrow \boldsymbol{uxv}$ in G, we also say that G makes a *derivation step* from $\boldsymbol{uAv}$ to $\boldsymbol{uxv}$.

Rule: $\boldsymbol{A} \rightarrow \boldsymbol{x}$



Sequence of Derivation Steps 1/2

Gist: Several consecutive derivation steps.

Definition: Let $u \in (N \cup T)^*$. G makes a *zero-step derivation* from u to u; in symbols,

$$u \Rightarrow^0 u\ [\varepsilon] \text{ or, simply, } u \Rightarrow^0 u$$

Definition: Let $u_0,\ldots,u_n \in (N \cup T)^*$, $n \geq 1$, and $u_{i-1} \Rightarrow u_i\ [p_i]$, $p_i \in P$, for all $i = 1,\ldots, n$; that is

$$u_0 \Rightarrow u_1\ [p_1] \Rightarrow u_2\ [p_2] \ldots \Rightarrow u_n\ [p_n]$$

Then, G makes n *derivation steps* from u_0 to u_n,

$$u_0 \Rightarrow^n u_n\ [p_1\ldots p_n] \text{ or, simply, } u_0 \Rightarrow^n u_n$$

Sequence of Derivation Steps 2/2

If $u_0 \Rightarrow^n u_n$ [π] for some $n \geq 1$, then u_0 *properly derives* u_n in G, written as $u_0 \Rightarrow^+ u_n$ [π].

If $u_0 \Rightarrow^n u_n$ [π] for some $n \geq 0$, then u_0 *derives* u_n in G, written as $u_0 \Rightarrow^* u_n$ [π].

Example: Consider

$\boldsymbol{aAb} \Rightarrow \boldsymbol{aaBbb}$ [**1**: $\boldsymbol{A} \rightarrow \boldsymbol{aBb}$], and

$\boldsymbol{aaBbb} \Rightarrow \boldsymbol{aacbb}$ [**2**: $\boldsymbol{B} \rightarrow \boldsymbol{c}$].

Then, $\boldsymbol{aAb} \Rightarrow^2 \boldsymbol{aacbb}$ [**1 2**],

$\boldsymbol{aAb} \Rightarrow^+ \boldsymbol{aacbb}$ [**1 2**],

$\boldsymbol{aAb} \Rightarrow^* \boldsymbol{aacbb}$ [**1 2**]

Generated Language

Gist: ***G generates* a terminal string *w* by a sequence of derivation steps from *S* to *w***

Definition: Let $G = (N, T, P, S)$ be a CFG. The *language generated by G*, $L(G)$, is defined as

$$L(G) = \{w: w \in T^*, S \Rightarrow^* w\}$$

Illustration:

$G = (N, T, P, S)$, let $w = a_1a_2\ldots a_n$; $a_i \in T$ for $i = 1..n$

if $S \Rightarrow \ldots \Rightarrow \ldots \Rightarrow \underbrace{a_1a_2\ldots a_n}_{w}$ **then** $w \in L(G)$;

otherwise, $w \notin L(G)$

Context-Free Language (CFL)

Gist: A language generated by a CFG.

Definition: Let L be a language. L is a *context-free language* (CFL) if there exists a context-free grammar that generates L.

Example:

$G = (N, T, P, S)$, where $N = \{S\}$, $T = \{a, b\}$,

$P = \{1\colon S \rightarrow aSb,\ 2\colon S \rightarrow \varepsilon\}$

$S \Rightarrow \varepsilon \quad [2]$

$S \Rightarrow aSb\ [1] \Rightarrow ab \quad [2]$

$S \Rightarrow aSb\ [1] \Rightarrow aaSbb\ [1] \Rightarrow aabb\ [2]$

⋮

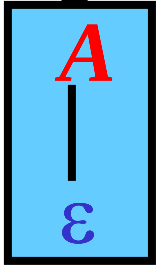
$L(G) = \{a^n b^n\colon n \geq 0\}$

$L = \{a^n b^n\colon n \geq 0\}$ is a CFL.

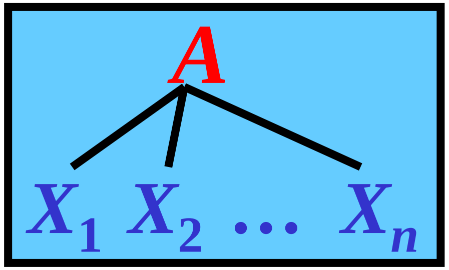
Rule Tree

- **Rule tree graphically represents a rule**

1) $A \to \varepsilon$:

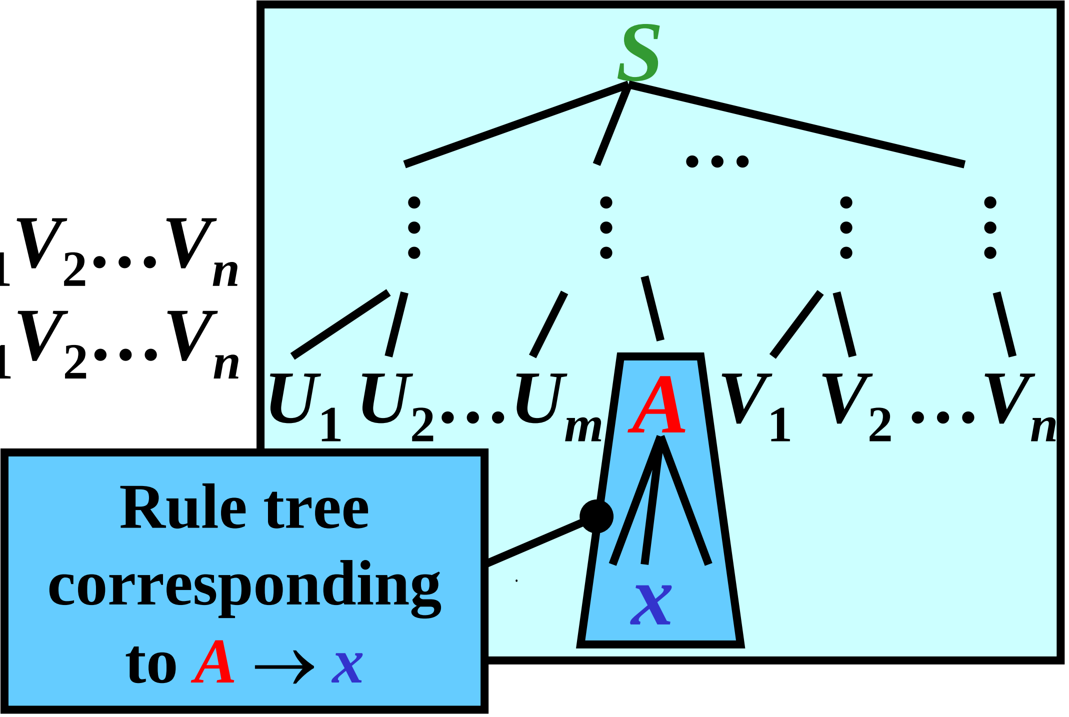


2) $A \to X_1X_2\ldots X_n$:



- **Derivation tree corresponding to a derivation**

$$\begin{aligned} S &\Rightarrow \ldots \\ &\vdots \\ &\Rightarrow U_1U_2\ldots U_m A V_1V_2\ldots V_n \\ &\Rightarrow U_1U_2\ldots U_m x V_1V_2\ldots V_n \end{aligned}$$



Rule tree corresponding to $A \to x$

Derivation Tree: Example

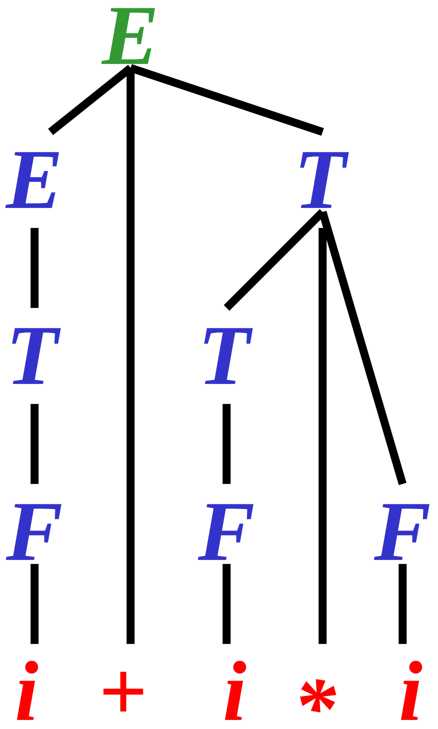
$G = (N, T, P, E)$, where $N = \{E, F, T\}$, $T = \{i, +, *, (,)\}$,

$P = \{$ 1: $E \rightarrow E+T$, 2: $E \rightarrow T$, 3: $T \rightarrow T*F$,
4: $T \rightarrow F$, 5: $F \rightarrow (E)$, 6: $F \rightarrow i$ $\}$

Derivation:

$$\underline{E} \Rightarrow E + \underline{T} \quad [1]$$
$$\Rightarrow E + \underline{T} * F \quad [3]$$
$$\Rightarrow E + \underline{F} * F \quad [4]$$
$$\Rightarrow \underline{E} + i * F \quad [6]$$
$$\Rightarrow T + i * \underline{F} \quad [2]$$
$$\Rightarrow \underline{T} + i * i \quad [6]$$
$$\Rightarrow \underline{F} + i * i \quad [4]$$
$$\Rightarrow i + i * i \quad [6]$$

Derivation tree:



Leftmost Derivation

Gist: During a *leftmost derivation step*, the leftmost nonterminal is rewritten.

Definition: Let $G = (N, T, P, S)$ be a CFG, let $\boldsymbol{u} \in T^*$, $\boldsymbol{v} \in (N \cup T)^*$. Let $\boldsymbol{p} = \boldsymbol{A} \to \boldsymbol{x} \in P$ be a rule. Then, $\boldsymbol{uAv}$ directly derives $\boldsymbol{uxv}$ in the *leftmost way* according to $\boldsymbol{p}$ in G, written as

$$\boldsymbol{uAv} \Rightarrow_{lm} \boldsymbol{uxv}\ [\boldsymbol{p}]$$

Note: We define $\Rightarrow_{lm}^+$ and $\Rightarrow_{lm}^*$ by analogy with $\Rightarrow^+$ and $\Rightarrow^*$, respectively.

Leftmost Derivation: Example

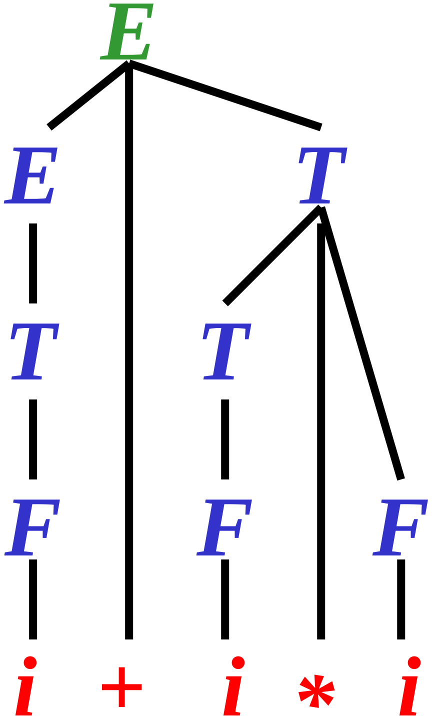
$G = (N, T, P, E)$, where $N = \{E, F, T\}$, $T = \{i, +, *, (,)\}$,

$P = \{$ 1: $E \rightarrow E+T$, 2: $E \rightarrow T$, 3: $T \rightarrow T*F$,
4: $T \rightarrow F$, 5: $F \rightarrow (E)$, 6: $F \rightarrow i$ $\}$

Leftmost derivation:

$$
\begin{aligned}
\underline{E} &\Rightarrow_{lm} \underline{E} + T && [1] \\
&\Rightarrow_{lm} \underline{T} + T && [2] \\
&\Rightarrow_{lm} \underline{F} + T && [4] \\
&\Rightarrow_{lm} i + \underline{T} && [6] \\
&\Rightarrow_{lm} i + \underline{T} * F && [3] \\
&\Rightarrow_{lm} i + \underline{F} * F && [4] \\
&\Rightarrow_{lm} i + i * \underline{F} && [6] \\
&\Rightarrow_{lm} i + i * i && [6]
\end{aligned}
$$

Derivation tree:



Rightmost Derivation

Gist: During a *rightmost derivation step*, the rightmost nonterminal is rewritten.

Definition: Let $G = (N, T, P, S)$ be a CFG, let $\boldsymbol{u} \in (N \cup T)^*$, $\boldsymbol{v} \in T^*$. Let $\boldsymbol{p} = \boldsymbol{A} \rightarrow \boldsymbol{x} \in P$ be a rule. Then, ***uAv*** directly derives ***uxv*** in the *rightmost way* according to ***p*** in *G*, written as

$$\boldsymbol{uAv} \Rightarrow_{rm} \boldsymbol{uxv}\ [\boldsymbol{p}]$$

Note: We define $\Rightarrow_{rm}{}^+$ and $\Rightarrow_{rm}{}^*$ by analogy with $\Rightarrow^+$ and $\Rightarrow^*$, respectively.

Rightmost Derivation: Example

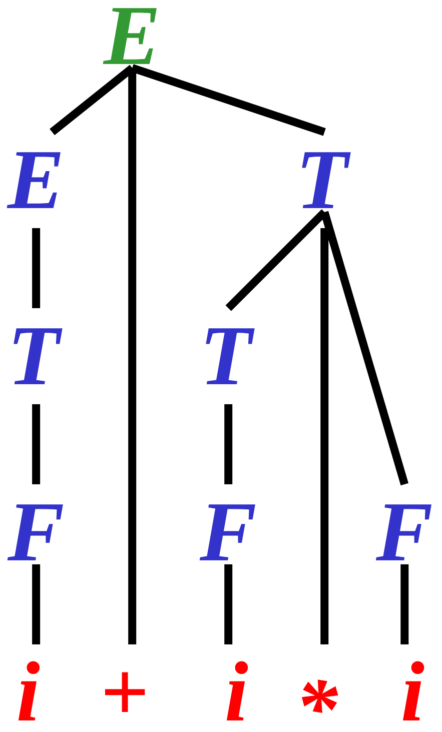
$G = (N, T, P, E)$, where $N = \{E, F, T\}$, $T = \{i, +, *, (,)\}$,

$P = \{$ 1: $E \rightarrow E+T$, 2: $E \rightarrow T$, 3: $T \rightarrow T*F$,
4: $T \rightarrow F$, 5: $F \rightarrow (E)$, 6: $F \rightarrow i$ $\}$

Rightmost derivation:

$$
\begin{aligned}
\underline{E} &\Rightarrow_{rm} E + \underline{T} && [1] \\
&\Rightarrow_{rm} E + T * \underline{F} && [3] \\
&\Rightarrow_{rm} E + \underline{T} * i && [6] \\
&\Rightarrow_{rm} E + \underline{F} * i && [4] \\
&\Rightarrow_{rm} \underline{E} + i * i && [6] \\
&\Rightarrow_{rm} \underline{T} + i * i && [2] \\
&\Rightarrow_{rm} \underline{F} + i * i && [4] \\
&\Rightarrow_{rm} i + i * i && [6]
\end{aligned}
$$

Derivation tree:



Derivations: Summary

- Let $\boldsymbol{A} \rightarrow \mathbf{x} \in P$ be a rule.

1) Derivation:

Let $\boldsymbol{u}, \mathbf{v} \in (N \cup T)^*$: $\boldsymbol{u}\boldsymbol{A}\mathbf{v} \Rightarrow \boldsymbol{u}\mathbf{x}\mathbf{v}$

Note: Any nonterminal is rewritten

2) Leftmost derivation:

Let $\boldsymbol{u} \in T^*, \mathbf{v} \in (N \cup T)^*$: $\boldsymbol{u}\boldsymbol{A}\mathbf{v} \Rightarrow_{\mathbf{lm}} \boldsymbol{u}\mathbf{x}\mathbf{v}$

Note: Leftmost nonterminal is rewritten

3) Rightmost derivation:

Let $\boldsymbol{u} \in (N \cup T)^*, \mathbf{v} \in T^*$: $\boldsymbol{u}\boldsymbol{A}\mathbf{v} \Rightarrow_{\mathbf{rm}} \boldsymbol{u}\mathbf{x}\mathbf{v}$

Note: Rightmost nonterminal is rewritten

Reduction of the Number of Derivations

Gist: Without any loss of generality, we can consider only leftmost or rightmost derivations.

Theorem: Let $G = (N, T, P, S)$ be a CFG. The next three languages coincide

(1) $\{w: w \in T^*, S \Rightarrow_{lm}^* w\}$

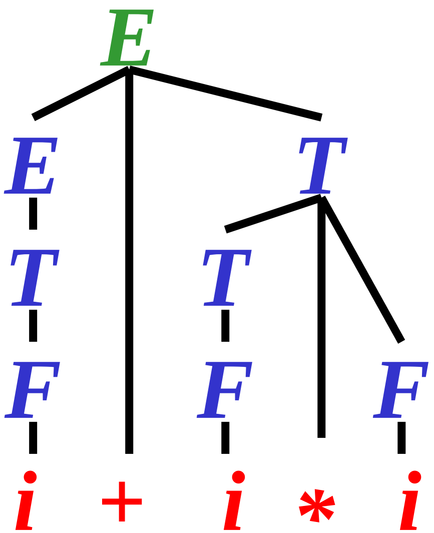
(2) $\{w: w \in T^*, S \Rightarrow_{rm}^* w\}$

(3) $\{w: w \in T^*, S \Rightarrow^* w\} = L(G)$

Introduction to Ambiguity

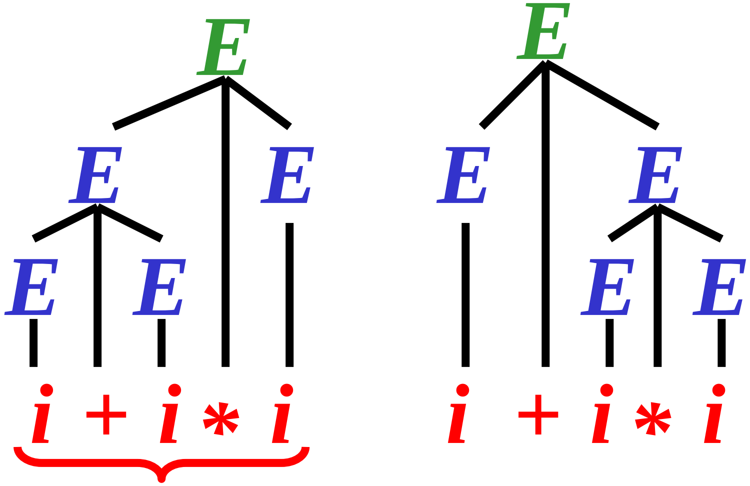
$G_{expr1} = (N, T, P, E)$, where
$N = \{E, F, T\}$, $T = \{i, +, *, (,)\}$,
$P = \{$ 1: $E \rightarrow E+T$, 2: $E \rightarrow T$,
3: $T \rightarrow T*F$, 4: $T \rightarrow F$,
5: $F \rightarrow (E)$, 6: $F \rightarrow i$ $\}$

Theory: ☹ × Practice: ☺



$G_{expr2} = (N, T, P, E)$, where
$N = \{E\}$, $T = \{i, +, *, (,)\}$,
$P = \{$ 1: $E \rightarrow E+E$, 2: $E \rightarrow E*E$,
3: $E \rightarrow (E)$, 4: $E \rightarrow i$ $\}$

Theory: ☺ × Practice: ☹



Improper during compilation

Note: $L(G_{expr1}) = L(G_{expr2})$

Grammatical Ambiguity

Definition: Let $G = (N, T, P, S)$ be a CFG. If there exists $x \in L(G)$ with more than one derivation tree, then G is *ambiguous*; otherwise, G is *unambiguous*.

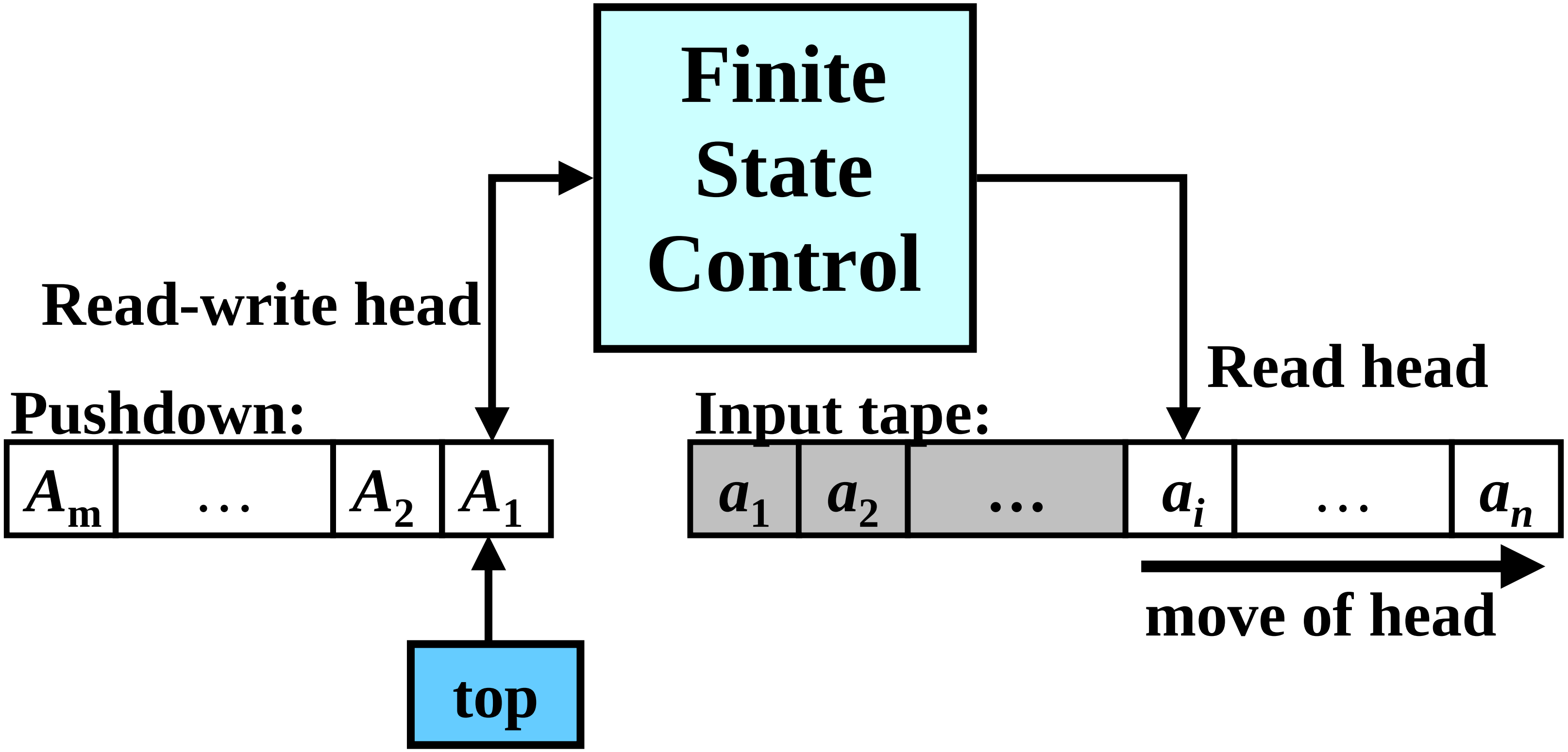
Definition: A CFL, L, is *inherently ambiguous* if L is generated by no unambiguous grammar.

Example:

- G_{expr1} is **unambiguous**, because for every $x \in L(G_{expr1})$ there exists **only one derivation tree**
- G_{expr2} is **ambiguous**, because for $i+i*i \in L(G_{expr2})$ there exist **two derivation trees**
- $L_{expr} = L(G_{expr1}) = L(G_{expr2})$ is **not inherently ambiguous** because G_{expr1} **is unambiguous**

Pushdown Automata (PDA)

Gist: An FA extended by a pushdown store.



Pushdown Automata: Definition

Definition: *A pushdown automaton* (PDA) is a 7-tuple $M = (Q, \Sigma, \Gamma, R, s, S, F)$, where

- Q is a *finite set of states*
- Σ is an *input alphabet*
- Γ is a *pushdown alphabet*
- R is a *finite set of rules* of the form: $Apa \rightarrow wq$
  where $A \in \Gamma$, p, $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $w \in \Gamma^*$
- $s \in Q$ is the *start state*
- $S \in \Gamma$ is the *start pushdown symbol*
- $F \subseteq Q$ is a set of *final states*

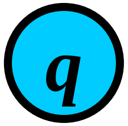
Notes on PDA Rules

Mathematical note on rules:

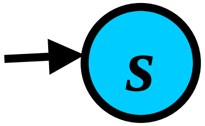
- Strictly mathematically, *R* is a relation from $\Gamma \times Q \times (\Sigma \cup \{\varepsilon\})$ to $\Gamma^* \times Q$
- Instead of $(\boldsymbol{Apa}, \boldsymbol{wq}) \in R$, however, we write $\boldsymbol{Apa} \rightarrow \boldsymbol{wq} \in R$

- **Interpretation of** $\boldsymbol{Apa} \rightarrow \boldsymbol{wq}$: if the current state is ***p***, current input symbol is ***a***, and the topmost symbol on the pushdown is ***A***, then *M* can read ***a***, replace ***A*** with ***w*** and change state ***p*** to ***q***.
- **Note**: if ***a*** = ε, no symbol is read

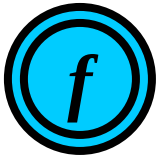
Graphical Representation



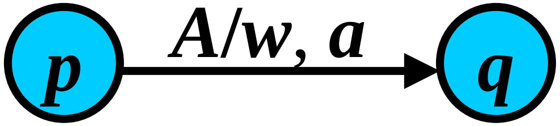
represents $q \in Q$



represents the initial state $s \in Q$



represents a final state $f \in F$



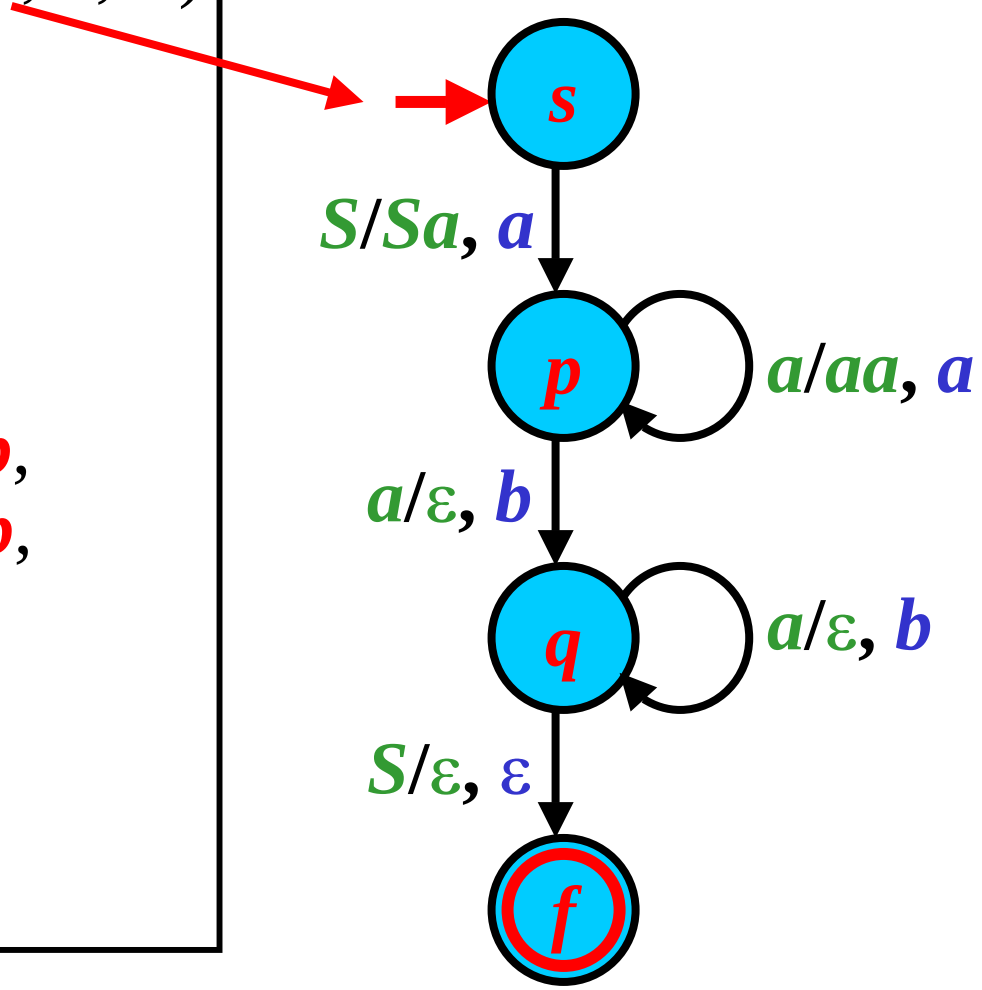
denotes $Apa \to wq \in R$

Graphical Representation: Example

$M = (Q, \Sigma, \Gamma, R, s, S, F)$

where:

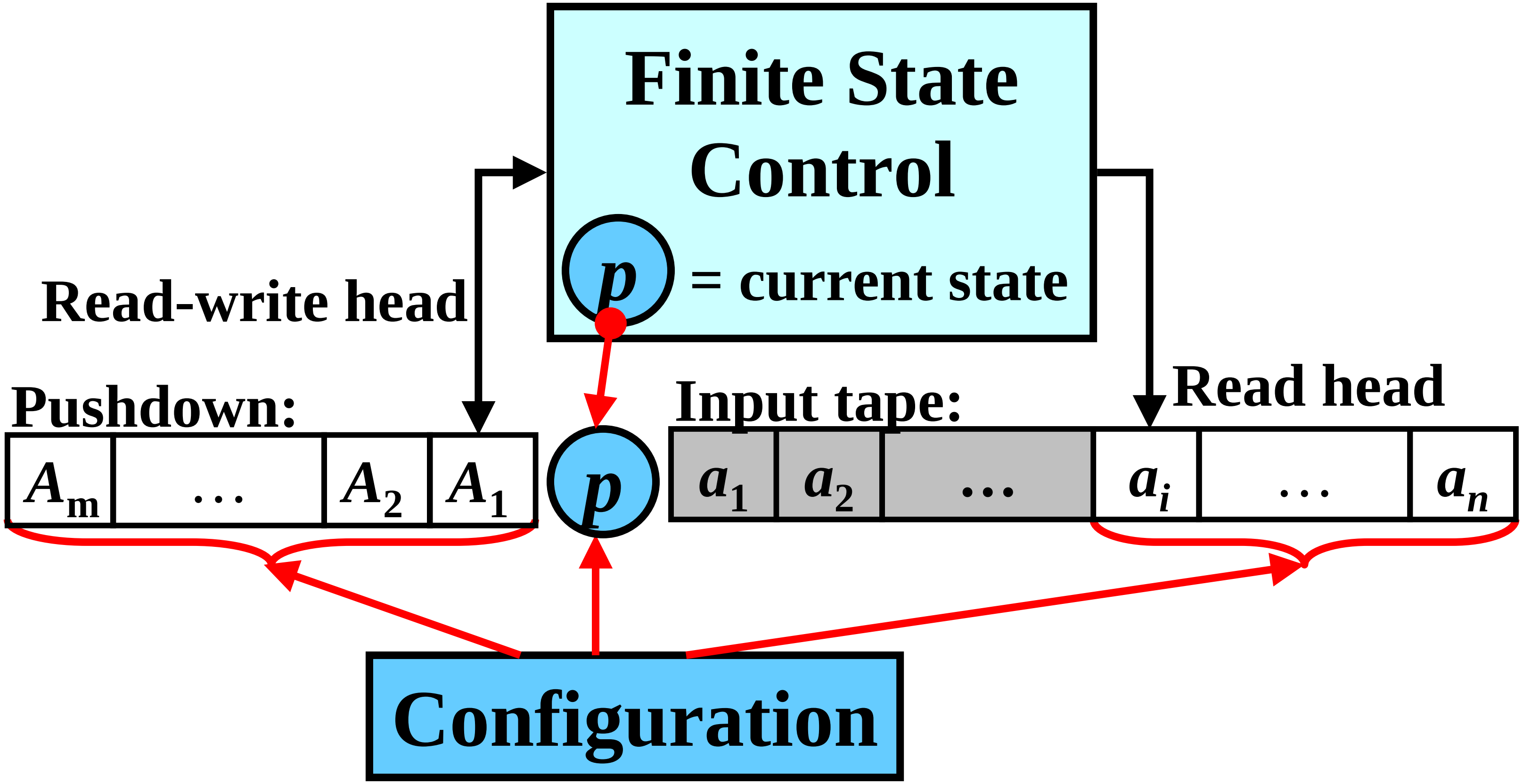
- $Q = \{s, p, q, f\}$;
- $\Sigma = \{a, b\}$;
- $\Gamma = \{a, S\}$;
- $R = \{Ssa \rightarrow Sap,$
  $apa \rightarrow aap,$
  $apb \rightarrow q,$
  $aqb \rightarrow q,$
  $Sq \rightarrow f\}$
- $F = \{f\}$



PDA Configuration

Gist: Instantaneous description of PDA

Definition: Let $M = (Q, \Sigma, \Gamma, R, s, S, F)$ be a PDA. *A configuration* of M is a string $\chi \in \Gamma^*Q\Sigma^*$



Move

Gist: A computational step made by a PDA

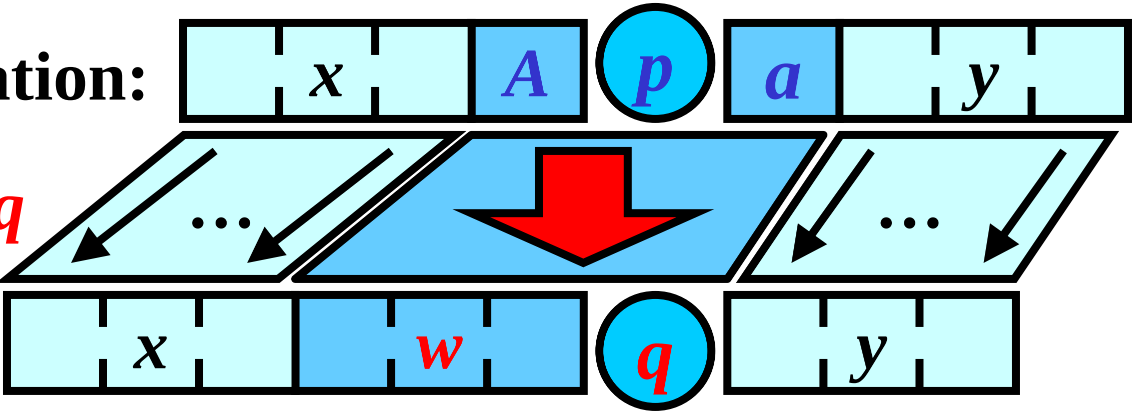
Definition: Let $xApay$ and $xwqy$ be two configurations of a PDA, M, where
$x, w \in \Gamma^*$, $A \in \Gamma$, $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, and $y \in \Sigma^*$.
Let $r = Apa \rightarrow wq \in R$ be a rule. Then, M makes a *move* from $xApay$ to $xwqy$ according to r, written as
$xApay \vdash xwqy\ [r]$ or, simply, $xApay \vdash xwqy$.

Note: if $a = \varepsilon$, no input symbol is read

Configuration:

Rule: $Apa \rightarrow wq$

New configuration:



Sequence of Moves 1/2

Gist: Several consecutive computational steps

Definition: Let χ be a configuration. M makes *zero moves* from χ to χ; in symbols,

$\chi \vdash^0 \chi\ [\varepsilon]$ or, simply, $\chi \vdash^0 \chi$

Definition: Let $\chi_0, \chi_1, \dots, \chi_\mathrm{n}$ be a sequence of configurations, $n \geq 1$, and $\chi_{i\text{-}1} \vdash \chi_i\ [r_i]$, $r_i \in R$, for all $i = 1, \dots, n$; that is,

$$\chi_0 \vdash \chi_1\ [r_1] \vdash \chi_2\ [r_2] \ \dots \ \vdash \chi_n\ [r_n]$$

Then M makes *n moves* from χ_0 to χ_n,

$\chi_0 \vdash^n \chi_n\ [r_1 \dots r_n]$ or, simply, $\chi_0 \vdash^n \chi_n$

Sequence of Moves 2/2

If $\chi_0 \vdash^n \chi_n$ [ρ] for some $n \geq 1$, then

$\chi_0 \vdash^+ \chi_n$ [ρ] or, simply, $\chi_0 \vdash^+ \chi_n$

If $\chi_0 \vdash^n \chi_n$ [ρ] for some $n \geq 0$, then

$\chi_0 \vdash^* \chi_n$ [ρ] or, simply, $\chi_0 \vdash^* \chi_n$

Example: Consider

AApabc $\vdash$ ***ABqbc*** [**1**: ***Apa*** → ***Bq***], and

ABqbc $\vdash$ ***ABCrc*** [**2**: ***Bqb*** → ***BCr***].

Then,

AApabc $\vdash^2$ ***ABCrc*** [**1 2**],

AApabc $\vdash^+$ ***ABCrc*** [**1 2**],

AApabc $\vdash^*$ ***ABCrc*** [**1 2**]

Accepted Language: Three Types

Definition: Let $M = (Q, \Sigma, \Gamma, R, s, S, F)$ be a PDA.

1) The *language that M accepts* ***by final state***, denoted by $\boldsymbol{L(M)_f}$, is defined as
$L(M)_f = \{w: w \in \Sigma^*, Ssw \vdash^* zf, \boldsymbol{z} \in \Gamma^*, \boldsymbol{f} \in \boldsymbol{F}\}$

2) The *language that M accepts* ***by empty pushdown***, denoted by $\boldsymbol{L(M)_\varepsilon}$, is defined as
$L(M)_\varepsilon = \{w: w \in \Sigma^*, Ssw \vdash^* zf, \boldsymbol{z} = \varepsilon, \boldsymbol{f} \in \boldsymbol{Q}\}$

3) The *language that M accepts* ***by final state and empty pushdown***, denoted by $\boldsymbol{L(M)_{f\varepsilon}}$, is defined as
$L(M)_{f\varepsilon} = \{w: w \in \Sigma^*, Ssw \vdash^* zf, \boldsymbol{z} = \varepsilon, \boldsymbol{f} \in \boldsymbol{F}\}$

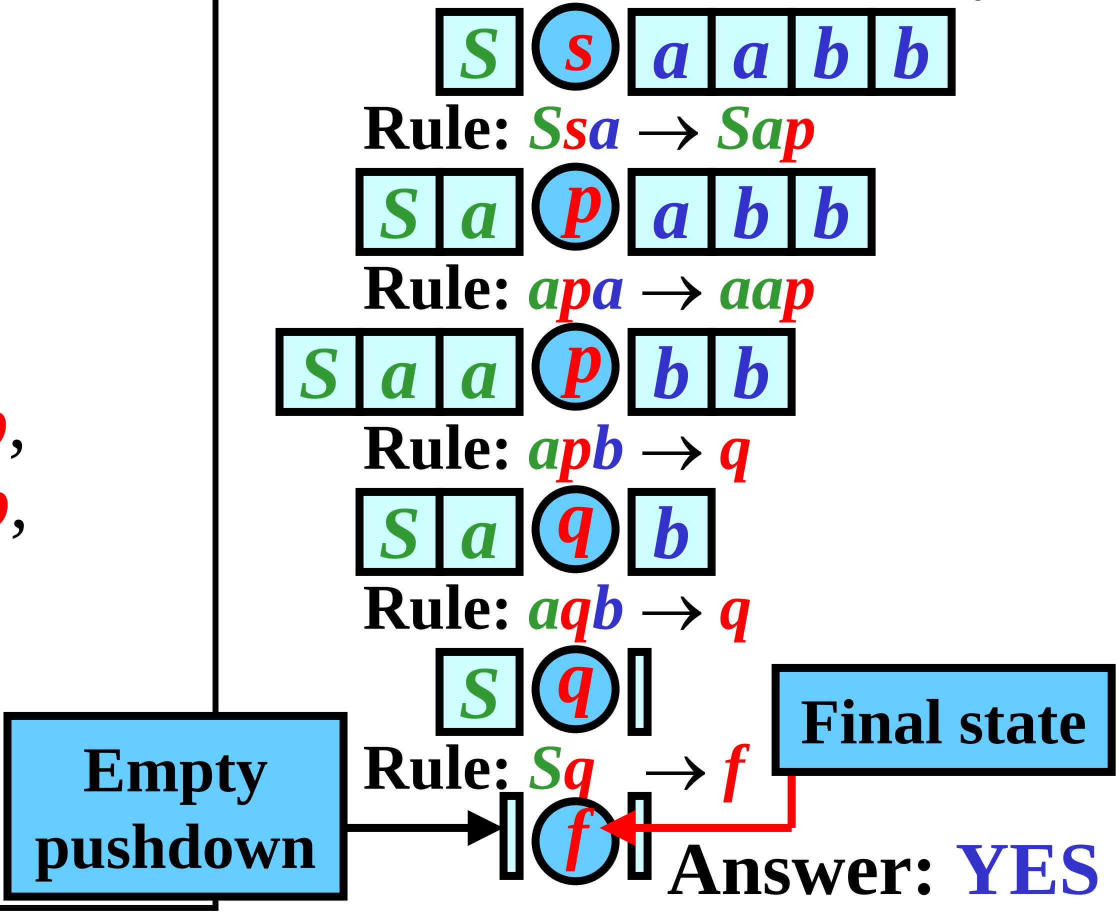
PDA: Example

$M = (Q, \Sigma, \Gamma, R, s, S, F)$

where:

- $Q = \{s, p, q, f\}$;
- $\Sigma = \{a, b\}$;
- $\Gamma = \{a, S\}$;
- $R = \{Ssa \to Sap,$
  $apa \to aap,$
  $apb \to q,$
  $aqb \to q,$
  $Sq \to f\}$
- $F = \{f\}$

Question: $aabb \in L(M)_{f\varepsilon}$?



Rule: $Ssa \to Sap$

Rule: $apa \to aap$

Rule: $apb \to q$

Rule: $aqb \to q$

Rule: $Sq \to f$

Empty pushdown

Final state

Answer: YES

$Ssaabb \vdash Sapabb \vdash Saapbb \vdash Saqb \vdash Sq \vdash f$

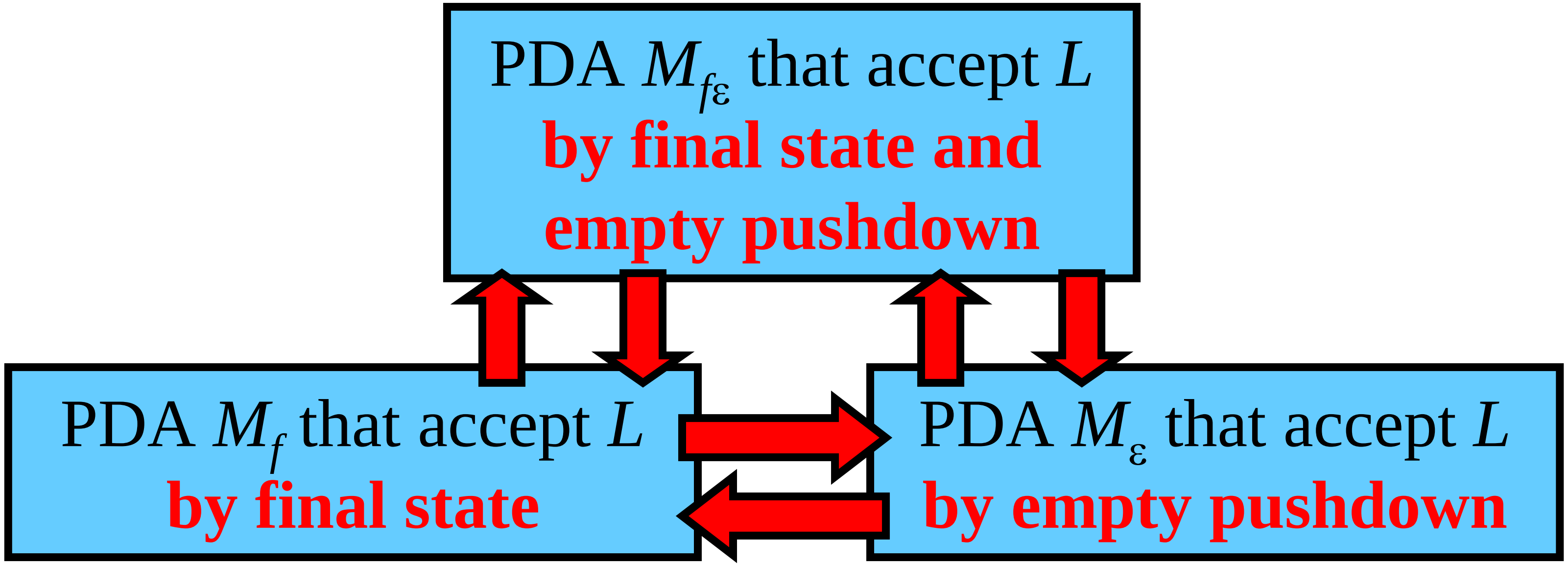
Note: $L(M)_f = L(M)_\varepsilon = L(M)_{f\varepsilon} = \{a^n b^n : n \geq 1\}$

Three Types of Acceptance: Equivalence

Theorem:

- $L = L(M_f)_f$ for a PDA $M_f \Leftrightarrow L = L(M_{f\varepsilon})_{f\varepsilon}$ for a PDA $M_{f\varepsilon}$
- $L = L(M_\varepsilon)_\varepsilon$ for a PDA $M_\varepsilon \Leftrightarrow L = L(M_{f\varepsilon})_{f\varepsilon}$ for a PDA $M_{f\varepsilon}$
- $L = L(M_f)_f$ for a PDA $M_f \Leftrightarrow L = L(M_\varepsilon)_\varepsilon$ for a PDA M_ε

Note: There exist these conversions:



PDA $M_{f\varepsilon}$ that accept L **by final state and empty pushdown**

PDA M_f that accept L **by final state**

PDA M_ε that accept L **by empty pushdown**

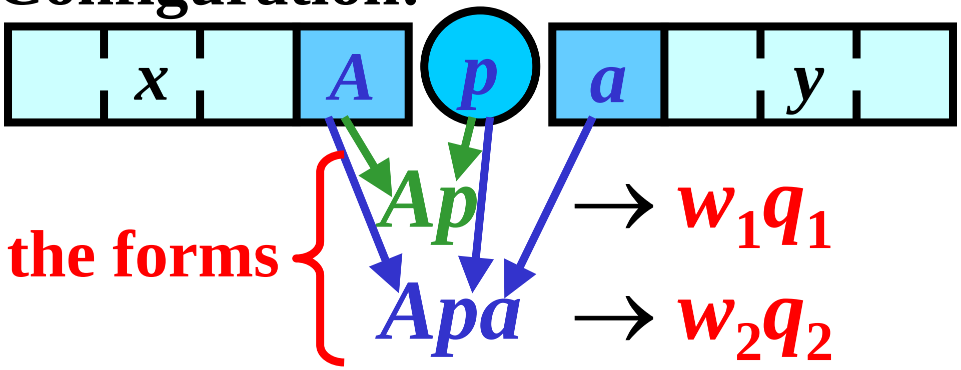
Deterministic PDA (DPDA)

Gist: Deterministic PDA makes no more than one move from any configuration.

Definition: Let $M = (Q, \Sigma, \Gamma, R, s, S, F)$ be a PDA. M is a *deterministic PDA* if for each rule $\boldsymbol{Apa} \rightarrow \boldsymbol{wq} \in R$, it holds that $R - \{\boldsymbol{Apa} \rightarrow \boldsymbol{wq}\}$ contains no rule with the left-hand side equal to $\boldsymbol{Apa}$ or $\boldsymbol{Ap}$.

Illustration:

Configuration:



No more that one rule of the forms

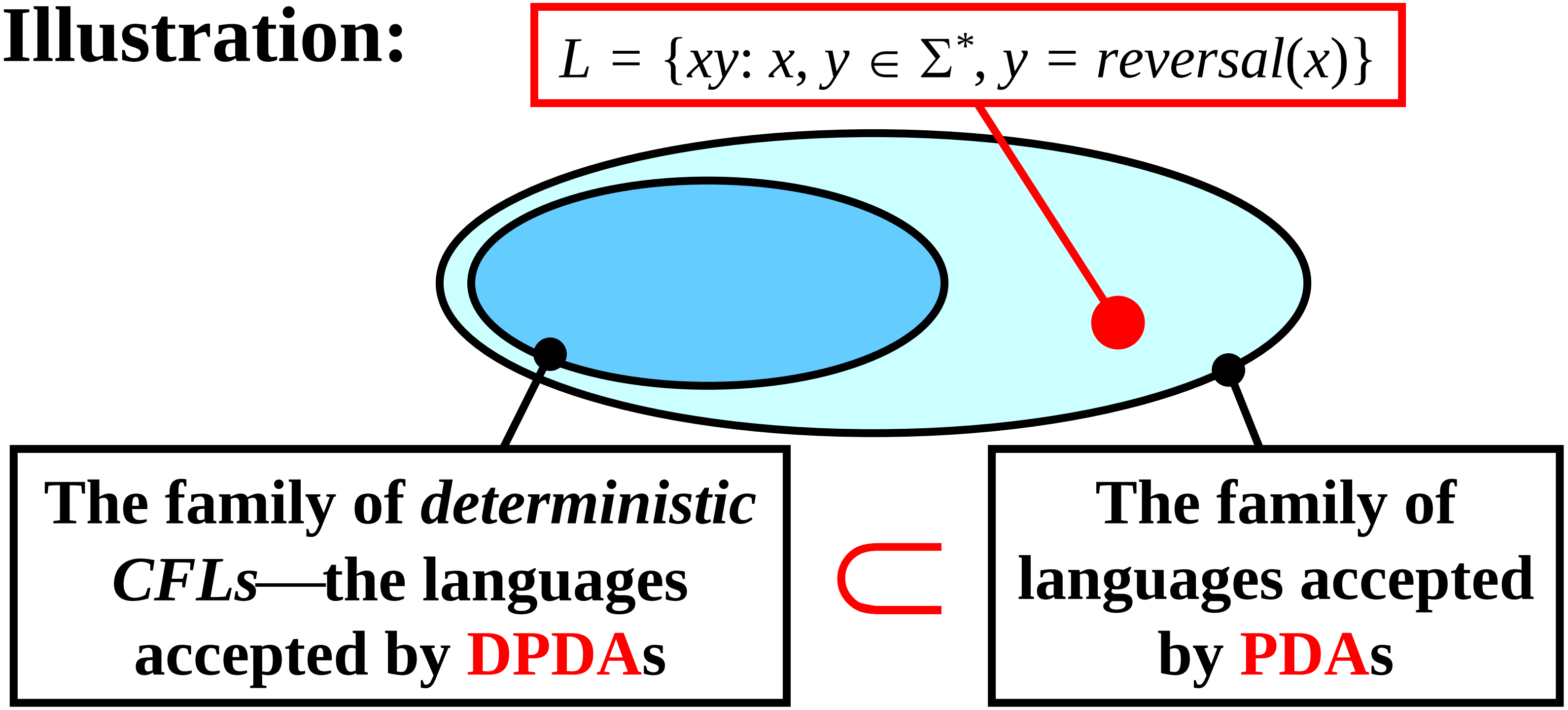
PDAs are Stronger than DPDAs

Theorem: There exists no DPDA $M_{f\varepsilon}$ that accepts

$$L = \{xy: x, y \in \Sigma^*, y = reversal(x)\}$$

Proof: See page 431 in [Meduna: Automata and Languages]

Illustration:



$L = \{xy: x, y \in \Sigma^*, y = reversal(x)\}$

The family of *deterministic CFLs*—the languages accepted by DPDAs

$\subset$

The family of languages accepted by PDAs

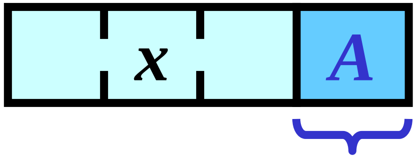
Extended PDA (EPDA)

Gist: The pushdown top of an EPDA represents a string rather than a single symbol.

Definition: *An Extended Pushdown automaton* (EPDA) is a 7-tuple $M = (Q, \Sigma, \Gamma, R, s, S, F)$, where $Q, \Sigma, \Gamma, s, S, F$ are defined as in an PDA and R is a *finite set of rules* of the form: $vpa \rightarrow wq$, where $v, w \in \Gamma^*$, $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$

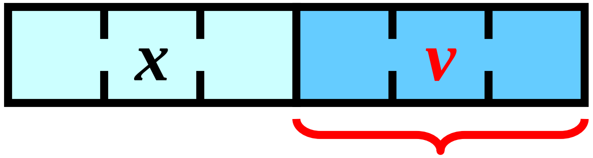
Illustration:

Pushdown of PDA:



PDA has a single symbols as the pushdown top

Pushdown of EPDA:



EPDA has a string as the pushdown top

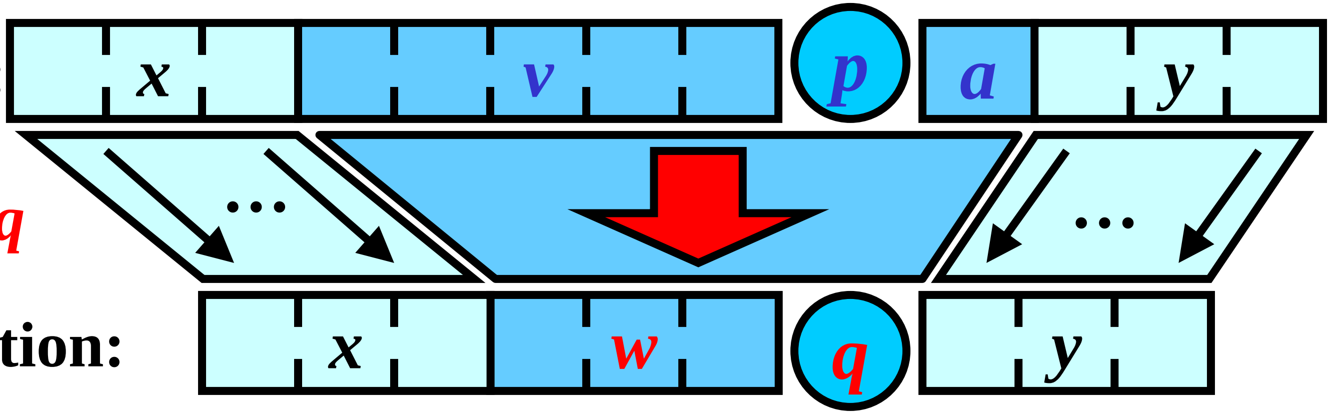
Move in EPDA

Definition: Let $\boldsymbol{xvpay}$ and $\boldsymbol{xwqy}$ be two configurations of an EPDA, M, where $\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{w} \in \Gamma^*$, $\boldsymbol{p}, \boldsymbol{q} \in Q$, $\boldsymbol{a} \in \Sigma \cup \{\varepsilon\}$, and $\boldsymbol{y} \in \Sigma^*$. Let $\boldsymbol{r} = \boldsymbol{vpa} \rightarrow \boldsymbol{wq} \in R$ be a rule. Then, M makes a *move* from $\boldsymbol{xvpay}$ to $\boldsymbol{xwqy}$ according to $\boldsymbol{r}$, written as $\boldsymbol{xvpay} \mid\!\!- \boldsymbol{xwqy}$ $[\boldsymbol{r}]$ or $\boldsymbol{xvpay} \mid\!\!- \boldsymbol{xwqy}$.

Configuration:

Rule: $\boldsymbol{vpa} \rightarrow \boldsymbol{wq}$

New configuration:



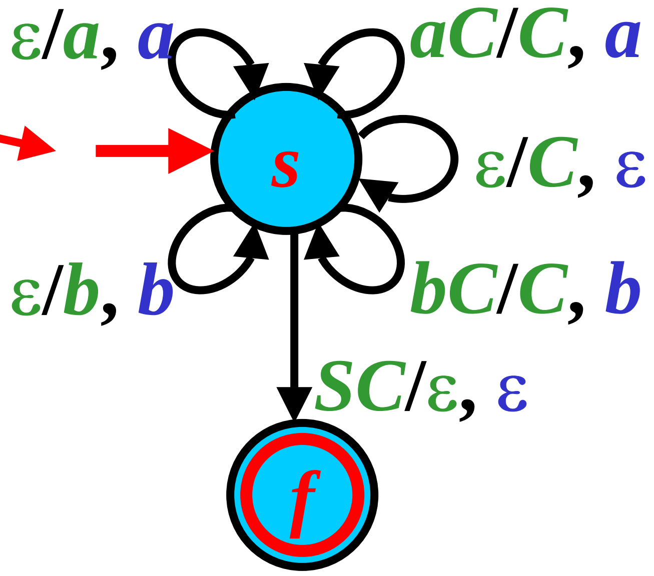
Note: $\mid\!\!-^n$, $\mid\!\!-^+$, $\mid\!\!-^*$, $L(M)_f$, $L(M)_\varepsilon$, and $L(M)_{f\varepsilon}$ are defined analogically to the corresponding definitions for PDA.

EPDA: Example

$M = (Q, \Sigma, \Gamma, R, s, S, F)$

where:

- $Q = \{s, f\}$;
- $\Sigma = \{a, b\}$;
- $\Gamma = \{a, b, S, C\}$;
- $R = \{\ sa \rightarrow as,\ sb \rightarrow bs,\ s \rightarrow Cs,\ aCsa \rightarrow Cs,\ bCsb \rightarrow Cs,\ SCs \rightarrow f\ \}$
- $F = \{f\}$



Question: $abba \in L_{f\varepsilon}(M)$?

$$S\underline{sa}bba \vdash Sa\underline{sb}ba \vdash Sab\underline{s}ba \vdash Sa\underline{bCsb}a \vdash S\underline{aCsa} \vdash \underline{SCs} \vdash f$$

Answer: YES

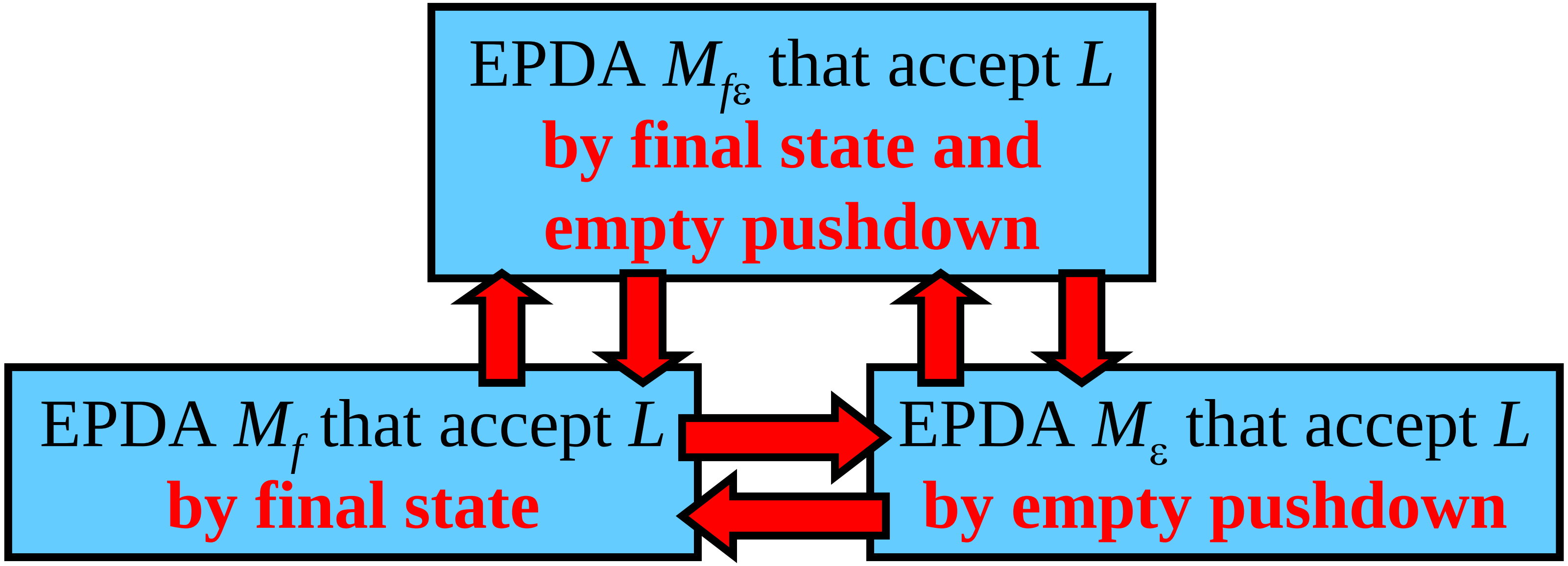
Note: $L(M)_f = L(M)_\varepsilon = L(M)_{f\varepsilon} = \{xy: x, y \in \Sigma^*, y = \text{reversal}(x)\}$

Three Types of Acceptance: Equivalence

Theorem:

- $L = L(M_f)_f$ for an EPDA $M_f \Leftrightarrow L = L(M_{f\varepsilon})_{f\varepsilon}$ for an EPDA $M_{f\varepsilon}$
- $L = L(M_\varepsilon)_\varepsilon$ for an EPDA $M_\varepsilon \Leftrightarrow L = L(M_{f\varepsilon})_{f\varepsilon}$ for an EPDA $M_{f\varepsilon}$
- $L = L(M_f)_f$ for an EPDA $M_f \Leftrightarrow L = L(M_\varepsilon)_\varepsilon$ for an EPDA M_ε

Note: There exist these conversion:



EPDA $M_{f\varepsilon}$ that accept L **by final state and empty pushdown**

EPDA M_f that accept L **by final state**

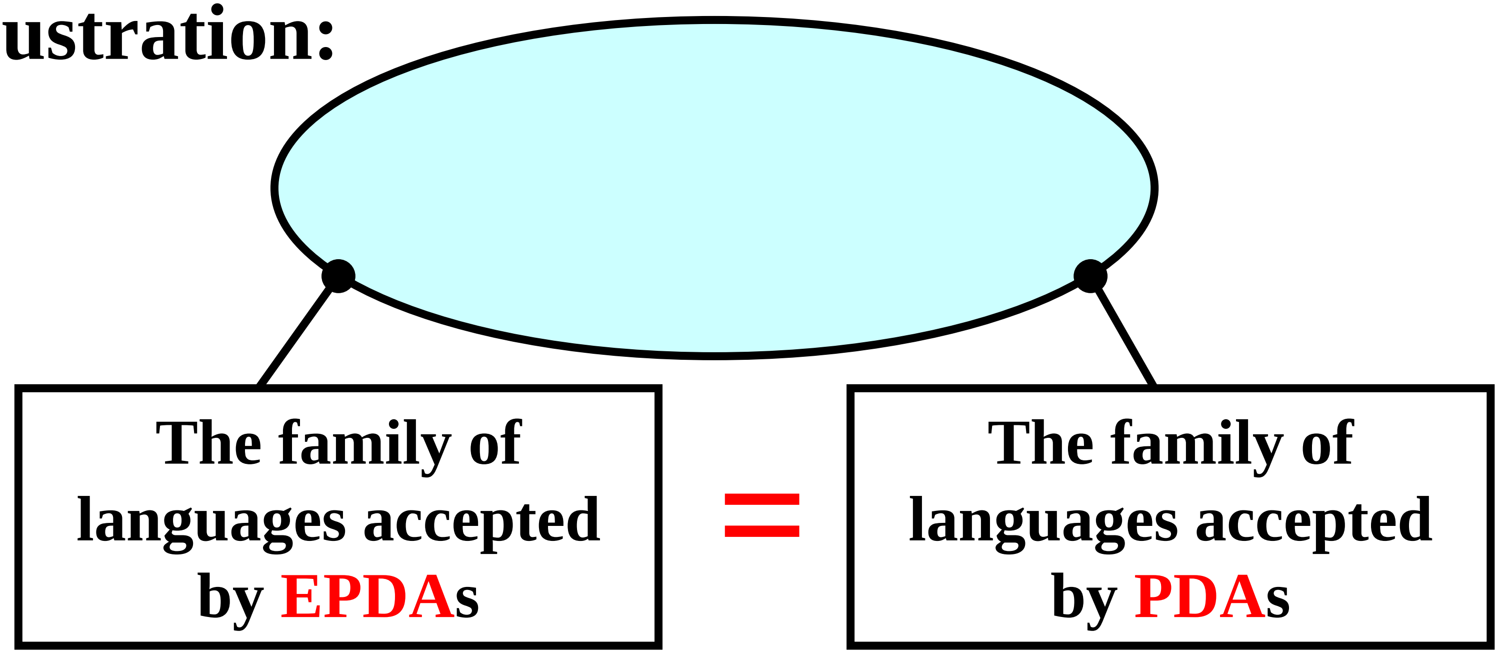
EPDA M_ε that accept L **by empty pushdown**

EPDAs and PDAs are Equivalent

Theorem: For every EPDA *M*, there is a PDA *M'*, and $L(M)_f = L(M')_f$.

Proof: See page 419 in [Meduna: Automata and Languages]

Illustration:



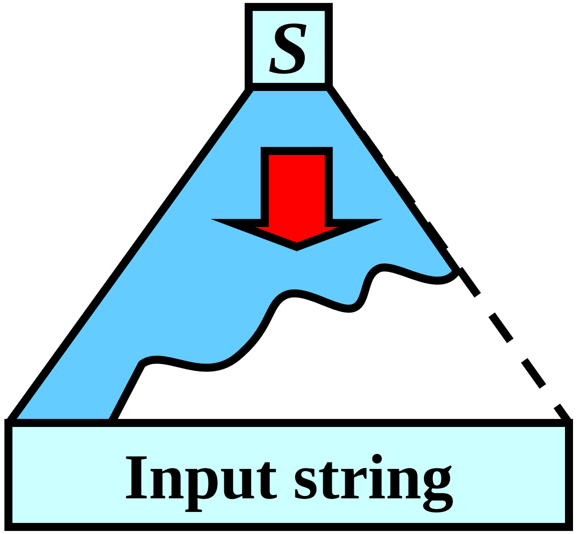
The family of languages accepted by EPDAs = **The family of languages accepted by PDAs**

EPDAs and PDAs as Parsing Models for CFGs

Gist: An EPDA or a PDA can simulate the construction of a derivation tree for a CFG

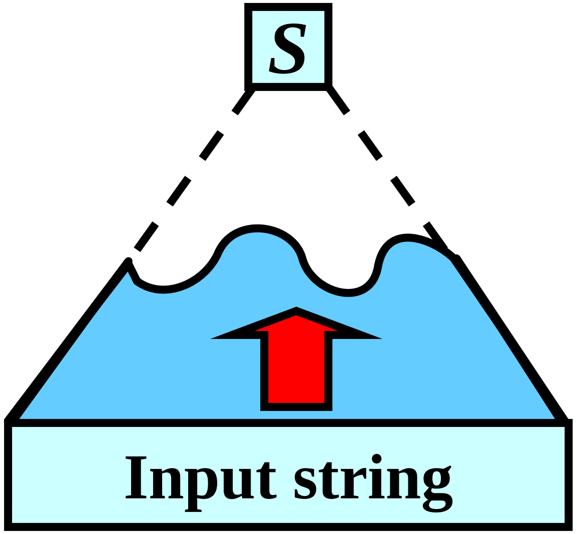
- **Two basic approaches:**

1) Top-Down Parsing



From *S* towards the input string

2) Bottom-Up Parsing

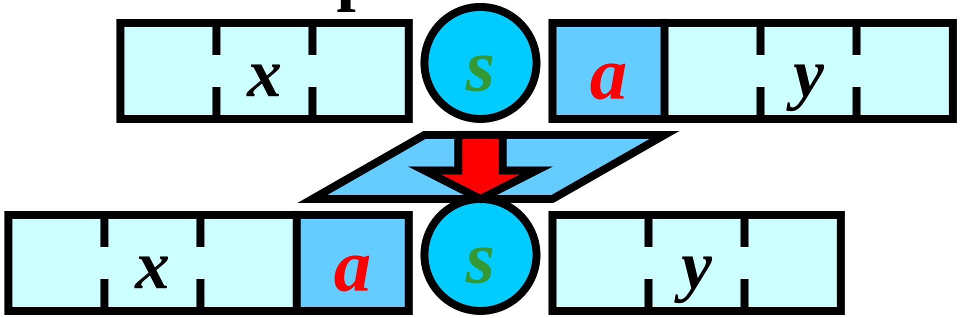


From the input string towards *S*

EPDAs as Models of Bottom-Up Parsers 1/2

Gist: An EPDA *M* underlies a bottom-up parser

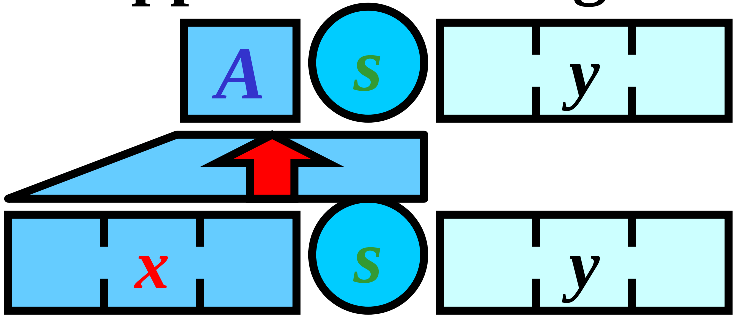
1) *M* contains *shift* rules that copy the input symbols onto the pushdown:



for every ***a*** ∈ Σ:

add ***sa*** → ***as*** to *R*;

2) *M* contains *reduction* rules that simulate the application of a grammatical rule in reverse:



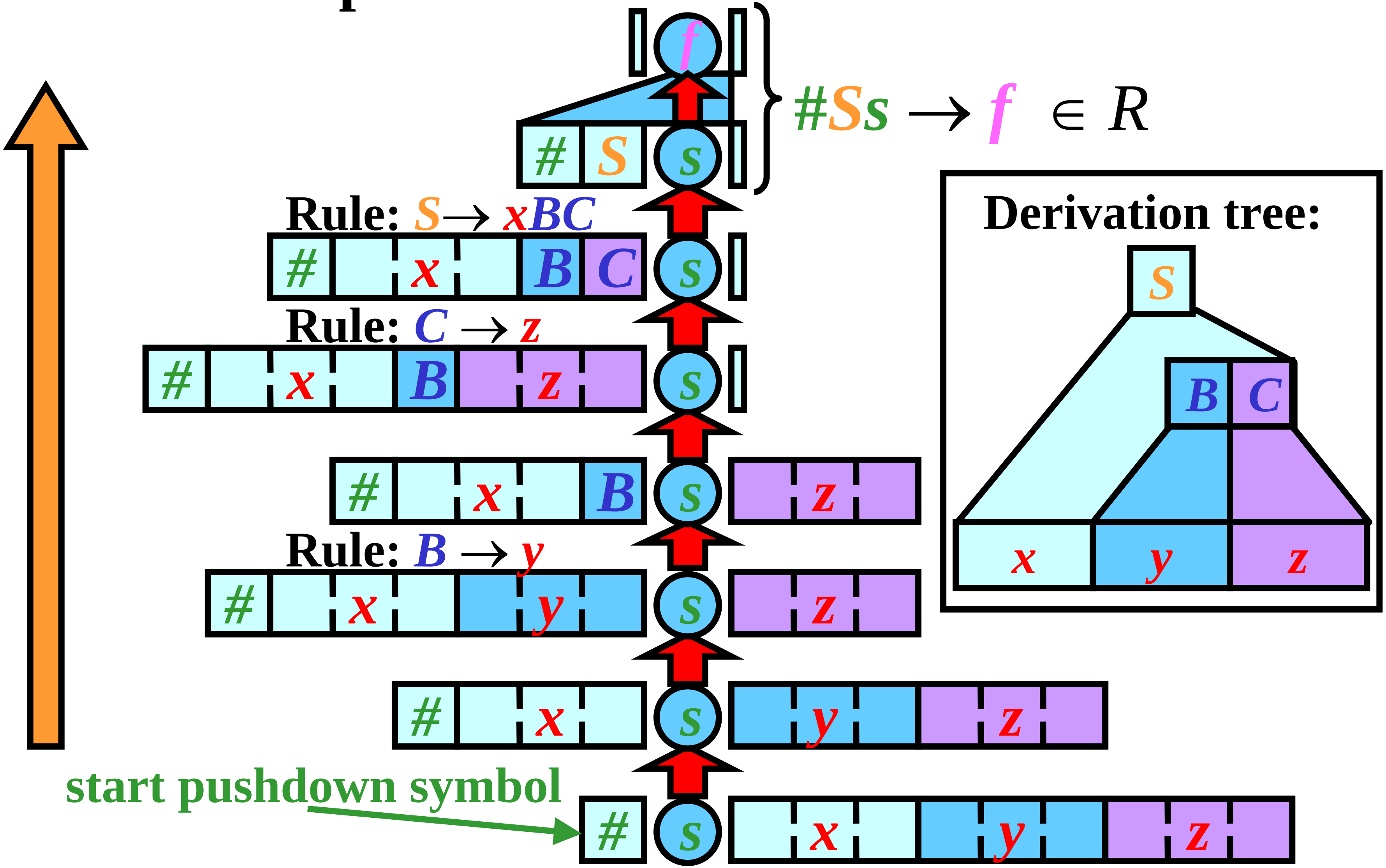
for every ***A*** → ***x*** ∈ *P* in *G*:

add ***xs*** → ***As*** to *R*;

3) *M* also contains the rule #*Ss* → *f* that takes *M* to a final state *f*

EPDAs as Models of Bottom-Up Parsers 2/2

Bottom-up construction of a derivation tree:



$\#Ss \rightarrow f \in R$

Rule: $S \rightarrow xBC$

Rule: $C \rightarrow z$

Rule: $B \rightarrow y$

start pushdown symbol

Derivation tree:

Algorithm: From CFG to EPDA

- **Input:** CFG $G = (N, T, P, \mathbf{S})$
- **Output:** EPDA $M = (Q, \Sigma, \Gamma, R, \mathbf{s}, \#, F)$; $L(G) = L(M)_f$

- **Method:**
- $Q := \{\mathbf{s}, f\}$;
- $\Sigma := T$;
- $\Gamma := N \cup T \cup \{\#\}$;
- Construction of R:
    - for every $\boldsymbol{a} \in \Sigma$, add $\mathbf{s}\boldsymbol{a} \rightarrow \boldsymbol{a}\mathbf{s}$ to R;
    - for every $\boldsymbol{A} \rightarrow \boldsymbol{x} \in P$, add $\boldsymbol{x}\mathbf{s} \rightarrow \boldsymbol{A}\mathbf{s}$ to R;
    - add $\#\mathbf{S}\mathbf{s} \rightarrow f$ to R;
- $F := \{f\}$;

From CFG to EPDA: Example 1/2

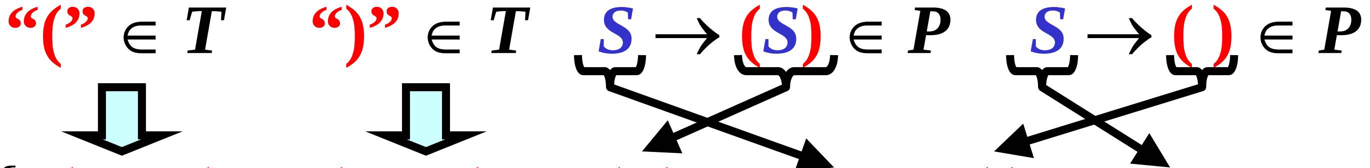
- $G = (N, T, P, S)$, where:

$N = \{S\}$, $T = \{(,)\}$, $P = \{S \to (S), S \to (\,)\}$

Objective: An EPDA M such that $L(G) = L(M)_f$

$M = (Q, \Sigma, \Gamma, R, s, \#, F)$ where:

$Q = \{s, f\}$; $\Sigma = T = \{(,)\}$; $\Gamma = N \cup T \cup \{\#\} = \{S, (,), \#\}$



“(” $\in T$ “)” $\in T$ $\underbrace{S} \to \underbrace{(S)} \in P$ $\underbrace{S} \to \underbrace{(\,)} \in P$

$R = \{\underbrace{s(\to (s, \quad s) \to)s,}_{\text{shift rules}} \quad \underbrace{(S)s \to Ss, \quad (\,)s \to Ss,}_{\text{reduction rules}} \quad \#Ss \to f\}$

$F = \{f\}$

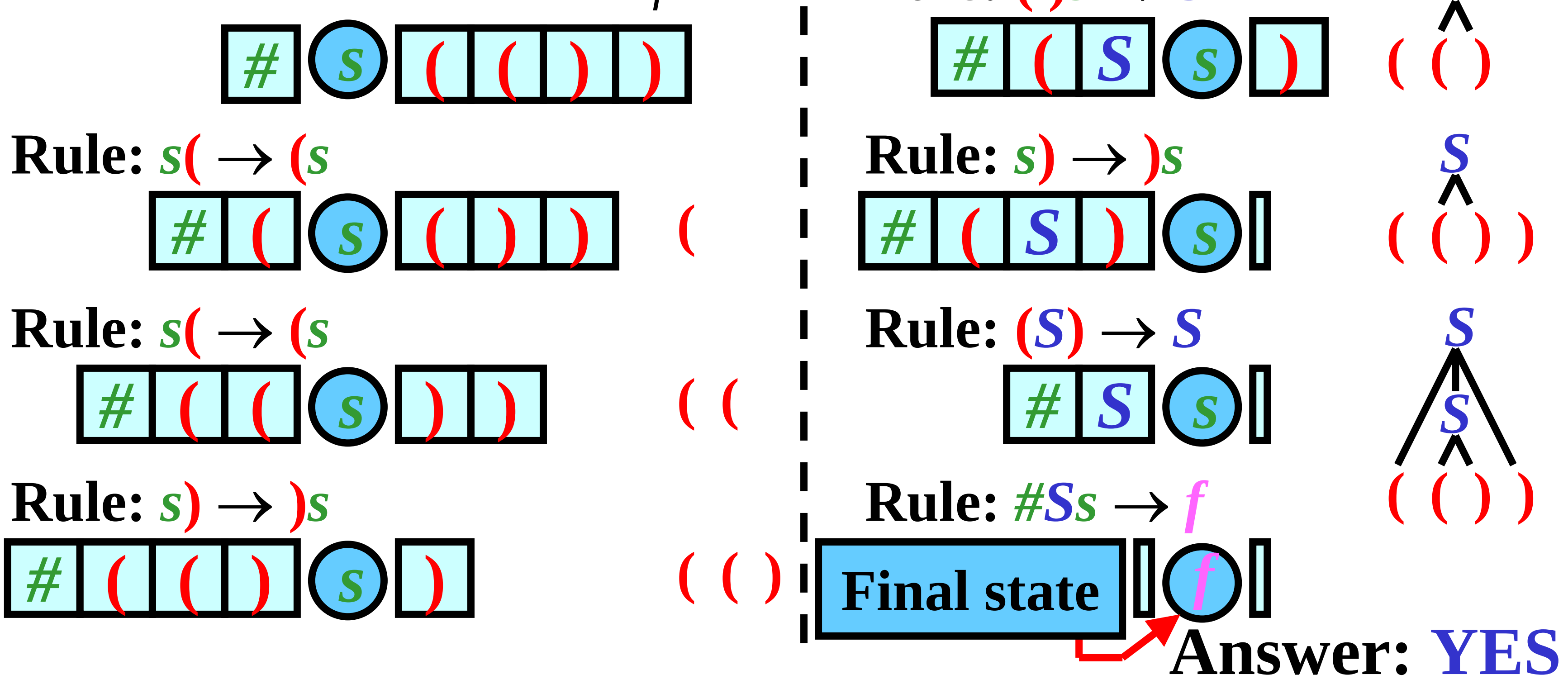
From CFG to EPDA: Example 2/2

$M = (Q, \Sigma, \Gamma, R, s, \#, F)$, where:

$Q = \{s, f\}$, $\Sigma = T = \{(,)\}$, $\Gamma = \{(,), S, \#\}$, $F = \{f\}$

$R = \{s(\to (s,\ s) \to)s,\ (S)s \to Ss,\ (\,)s \to Ss,\ \#Ss \to f\,\}$

Question: $((\,)) \in L(M)_f$?



$\#\ s\ (\,(\,)\,)$

Rule: $s(\to (s$

$\#(\ s\ (\,)\,)$ $\quad ($

Rule: $s(\to (s$

$\#((\ s\ )\,)$ $\quad ((\,$

Rule: $s) \to)s$

$\#(()\ s\ )$ $\quad ((\,)$

Rule: $(\,)s \to S$

$\#(S\ s\ )$ $\quad S$ over $((\,)$

Rule: $s) \to)s$

$\#(S)\ s$ $\quad S$ over $((\,)\,)$

Rule: $(S) \to S$

$\#S\ s$ $\quad S$ over $(\,S\,)$, S over $((\,)\,)$

Rule: $\#Ss \to f$

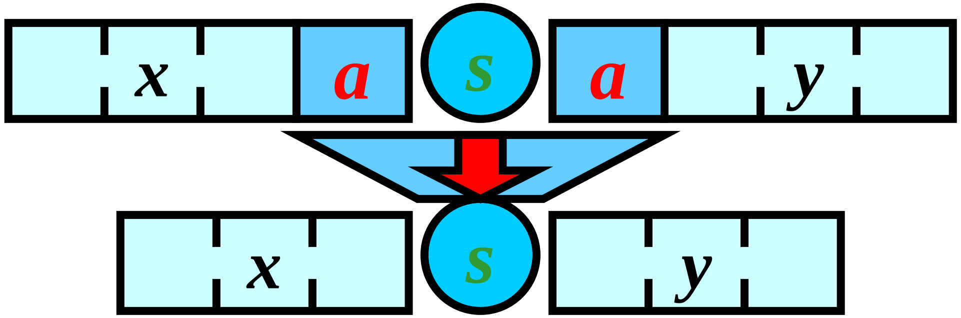
Final state f

Answer: YES

PDAs as Models of Top-Down Parsers 1/2

Gist: An PDA *M* underlies a top-down parser

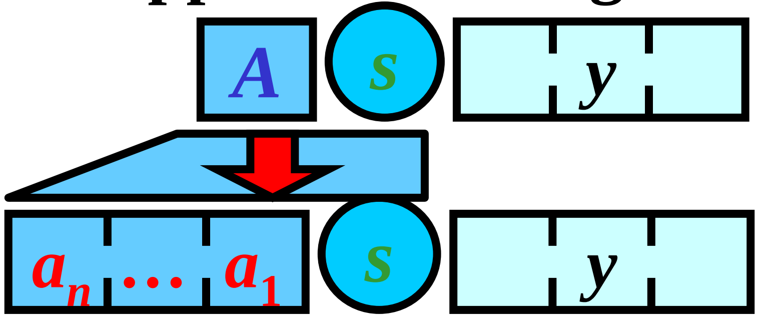
1) *M* contains *popping* rules that pops the top symbol from the pushdown and reads the input symbol if both coincide:



for every $a \in \Sigma$:

add $asa \rightarrow s$ to R;

2) *M* contains *expansion* rules that simulate the application of a grammatical rule:

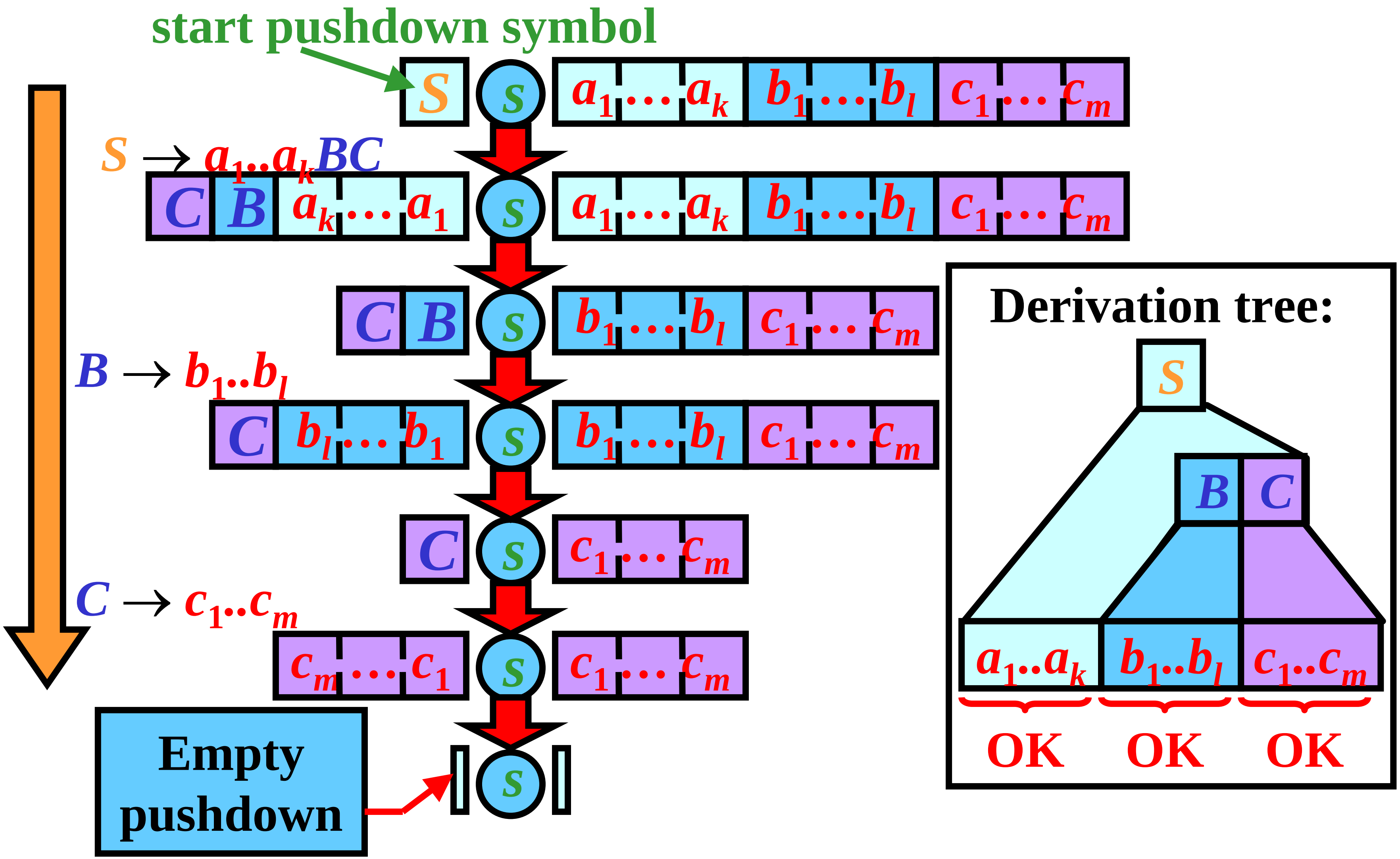


for every $A \rightarrow a_1 \ldots a_n \in P$ in G,

add $As \rightarrow \underbrace{a_n \ldots a_1}_{= \text{reversal}(a_1 \ldots a_n)} s$ to R;

PDAs as Models of Top-Down Parsers 2/2

Top-down construction of a derivation tree:



Algorithm: From CFG to PDA

- **Input:** CFG $G = (N, T, P, \mathbf{S})$
- **Output:** PDA $M = (Q, \Sigma, \Gamma, R, \mathbf{s}, \mathbf{S}, F)$; $L(G) = L(M)_{\varepsilon}$

- **Method:**
- $Q := \{\mathbf{s}\}$;
- $\Sigma := T$;
- $\Gamma := N \cup T$;
- Construction of R:
    - for every $\boldsymbol{a} \in \Sigma$, add $\boldsymbol{a}\mathbf{s}\boldsymbol{a} \rightarrow \mathbf{s}$ to R;
    - for every $\boldsymbol{A} \rightarrow \boldsymbol{x} \in P$, add $\boldsymbol{A}\mathbf{s} \rightarrow \boldsymbol{y}\mathbf{s}$ to R, where $\boldsymbol{y}$ = reversal($\boldsymbol{x}$);
- $F := \emptyset$;

From CFG to PDA: Example 1/2

- $G = (N, T, P, S)$, where:

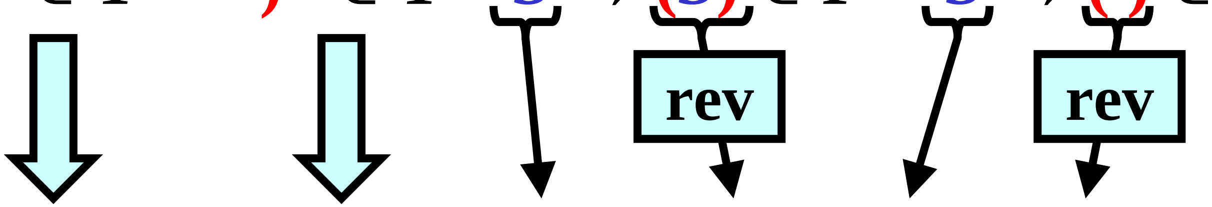
$N = \{S\}$, $T = \{(,)\}$, $P = \{S \rightarrow (S), S \rightarrow (\ )\}$

Objective: An PDA M such that $L(G) = L(M)_\varepsilon$

$M = (Q, \Sigma, \Gamma, R, s, S, F)$ where:

$Q = \{s\}$; $\Sigma = T = \{(,)\}$; $\Gamma = N \cup T = \{S, (,)\}$

“(” $\in$ *T* **“)” $\in$ *T*** $\underbrace{S} \rightarrow \underbrace{(S)} \in \boldsymbol{P}$ $\underbrace{S} \rightarrow \underbrace{(\ )} \in \boldsymbol{P}$



$$R = \{\underbrace{(s(\rightarrow s,\quad)s) \rightarrow s,}_{\textbf{popping rules}}\quad \underbrace{Ss \rightarrow)S(s,\quad Ss \rightarrow)(s}_{\textbf{expansion rules}}\}$$

$F = \emptyset$

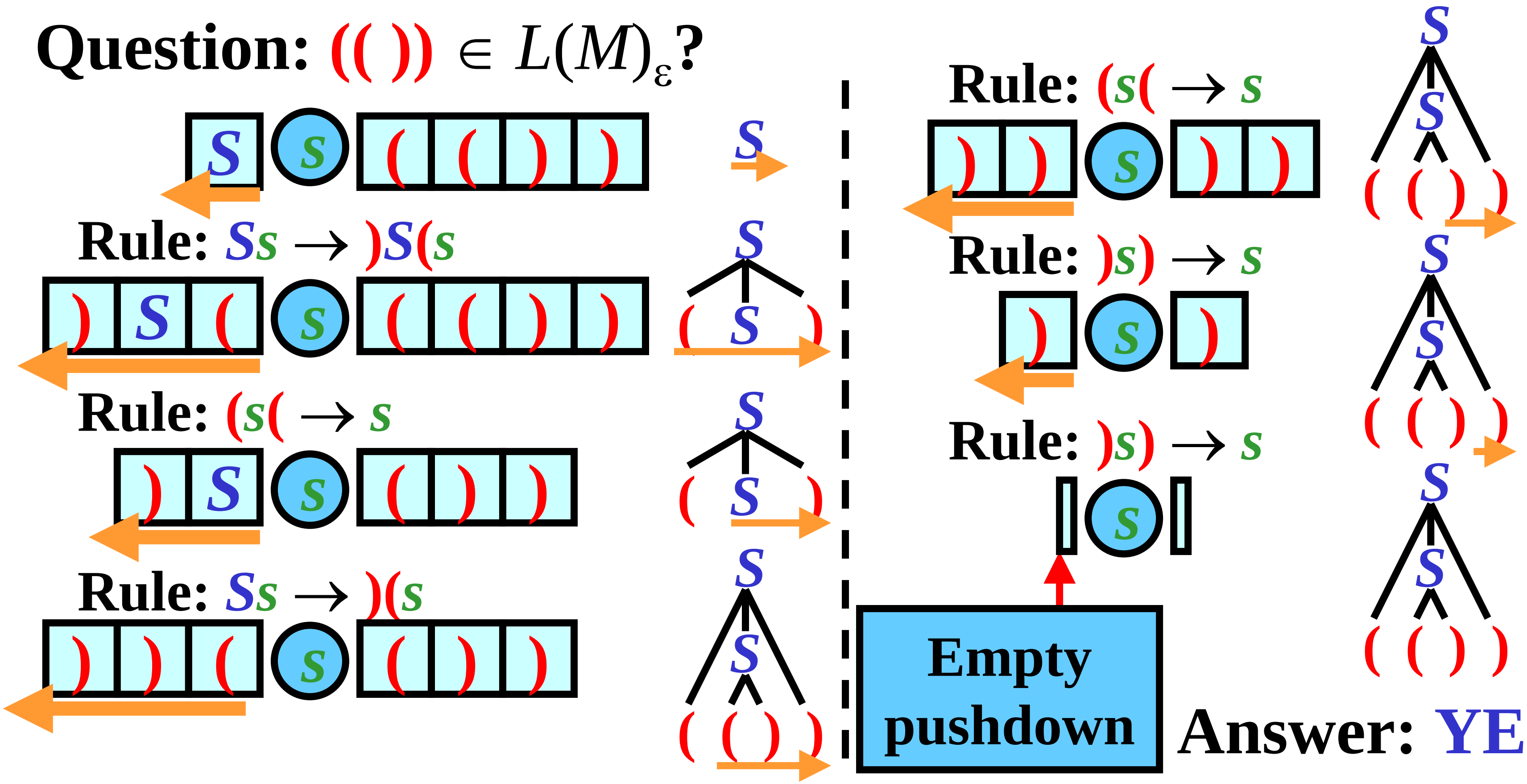
From CFG to PDA: Example 2/2

$M = (Q, \Sigma, \Gamma, R, s, S, F)$, where:

$Q = \{s\}, \Sigma = T = \{(,)\}, \Gamma = \{(,), S\}, F = \emptyset$

$P = \{(s(\rightarrow s, \quad)s) \rightarrow s, \quad Ss \rightarrow)S(s, \quad Ss \rightarrow)(s\ \}$

Question: (( )) $\in L(M)_\varepsilon$**?**



Rule: $Ss \rightarrow)S(s$

Rule: $(s(\rightarrow s$

Rule: $Ss \rightarrow)(s$

Rule: $(s(\rightarrow s$

Rule: $)s) \rightarrow s$

Rule: $)s) \rightarrow s$

Answer: YES

Models for Context-free Languages

Theorem: For every CFG G, there is an PDA M such that $L(G) = L(M)_\varepsilon$.

Proof: See the previous algorithm.

Theorem: For every PDA M, there is a CFG G such that $L(M)_\varepsilon = L(G)$.

Proof: See page 486 in [Meduna: Automata and Languages]

Conclusion: The fundamental models for context-free languages are

1) Context-free grammars 2) Pushdown automata