Context-free Grammars and Pushdown Automatas

Implementation of Computational Models
(TC2037)

Edgar Covantes Osuna, PhD
edgar.covantes@tec.mx

Tecnológico
de Monterrey

Growth and no-determinism in CFGs

LL and LR parsers

Let us remember that a CFG, for example, to generate words from the language $\{a^n b^n\}$:

1. $S \to aSb$
2. $S \to \varepsilon$

Where does the word **grow**? Derive using $w = \langle 1, 1, 1, 2 \rangle$.
Without using a given sequence, at the time to derive a word using the grammar rules, we have to choose between which of the available rules to use.

Growth and no-determinism in CFGs

LL and LR parsers

Let us remember that a CFG, for example, to generate words from the language $\{a^n b^n\}$:

1. $S \rightarrow aSb$
2. $S \rightarrow \varepsilon$

Where does the word **grow**? Derive using $w = \langle 1, 1, 1, 2 \rangle$.
Without using a given sequence, at the time to derive a word using the grammar rules, we have to choose between which of the available rules to use.

Growth and no-determinism in CFGs

LL and LR parsers

Let us remember that a CFG, for example, to generate words from the language $\{a^n b^n\}$:

1. $S \rightarrow aSb$
2. $S \rightarrow \varepsilon$

Where does the word **grow**? Derive using $w = \langle 1, 1, 1, 2 \rangle$.
Without using a given sequence, at the time to derive a word using the grammar rules, we have to choose between which of the available rules to use.

Growth and no-determinism in CFGs

LL and LR parsers

Let us remember that a CFG, for example, to generate words from the language $\{a^n b^n\}$:

1. $S \to aSb$
2. $S \to \varepsilon$

Where does the word **grow**? Derive using $w = \langle 1, 1, 1, 2 \rangle$.
Without using a given sequence, at the time to derive a word using the grammar rules, we have to choose between which of the available rules to use.

Growth and no-determinism in CFGs

LL and LR parsers

Let us remember that a CFG, for example, to generate words from the language $\{a^n b^n\}$:

1. $S \rightarrow aSb$
2. $S \rightarrow \varepsilon$

Where does the word **grow**? Derive using $w = \langle 1, 1, 1, 2 \rangle$.

Without using a given sequence, at the time to derive a word using the grammar rules, we have to choose between which of the available rules to use.

Growth and no-determinism in CFGs

LL and LR parsers

Let us remember that a CFG, for example, to generate words from the language $\{a^n b^n\}$:

1. $S \rightarrow aSb$
2. $S \rightarrow \varepsilon$

Where does the word **grow**? Derive using $w = \langle 1, 1, 1, 2 \rangle$.
Without using a given sequence, at the time to derive a word using the grammar rules, we have to choose between which of the available rules to use.

Growth and no-determinism in CFGs

LL and LR parsers

1. $S \to aSb$
2. $S \to \varepsilon$

How do we know what rules to use?

We can choose because we know the word that we want to obtain. A compiler cannot do that, so you need some other mechanism to **anticipate** what rule to use.

Generate the complete transition function of a pushdown automata for $\{a^n b^n\}$.

Growth and no-determinism in CFGs

LL and LR parsers

1. $S \to aSb$
2. $S \to \varepsilon$

How do we know what rules to use?

We can choose because we know the word that we want to obtain. A compiler cannot do that, so you need some other mechanism to **anticipate** what rule to use.

Generate the complete transition function of a pushdown automata for $\{a^n b^n\}$.

Growth and no-determinism in CFGs

LL and LR parsers

1. $S \to aSb$
2. $S \to \varepsilon$

How do we know what rules to use?

We can choose because we know the word that we want to obtain. A compiler cannot do that, so you need some other mechanism to **anticipate** what rule to use.

Generate the complete transition function of a pushdown automata for $\{a^n b^n\}$.

Growth and no-determinism in CFGs

LL and LR parsers

1. $S \to aSb$
2. $S \to \varepsilon$

How do we know what rules to use?

We can choose because we know the word that we want to obtain. A compiler cannot do that, so you need some other mechanism to **anticipate** what rule to use.

Generate the complete transition function of a pushdown automata for $\{a^nb^n\}$.

Growth and no-determinism in CFGs

LL and LR parsers

1. $S \to aSb$
2. $S \to \varepsilon$

How do we know what rules to use?

We can choose because we know the word that we want to obtain. A compiler cannot do that, so you need some other mechanism to **anticipate** what rule to use.

Generate the complete transition function of a pushdown automata for $\{a^n b^n\}$.

Growth and no-determinism in CFGs

LL and LR parsers

Tape	Pop	Move	Push
$aabb$	\$	R	A
abb	A	R	AA
bb	A	R	ε
b	A	R	ε
■	\$	N	ε

There are many rules that we are not using, and we are guided by the fact of looking *forward*. If we know that we are looking for $aabb$, then the most logic step is to use $S \to aSb$ two times, in order to *finish* with $S \to \varepsilon$.

This type of parsers, that read from left to right, and have a "prediction" window are known as *Left to right Leftmost derivation*, or LL for its acronym in English.

Growth and no-determinism in CFGs

LL and LR parsers

Tape	Pop	Move	Push
$aabb$	\$	R	A
abb	A	R	AA
bb	A	R	ε
b	A	R	ε
■	\$	N	ε

There are many rules that we are not using, and we are guided by the fact of looking *forward*. If we know that we are looking for $aabb$, then the most logic step is to use $S \to aSb$ two times, in order to *finish* with $S \to \varepsilon$.

This type of parsers, that read from left to right, and have a “prediction” window are known as *Left to right Leftmost derivation*, or LL for its acronym in English.

Growth and no-determinism in CFGs

LL and LR parsers

Tape	Pop	Move	Push
$aabb$	$\$$	R	A
abb	A	R	AA
bb	A	R	ε
b	A	R	ε
■	$\$$	N	ε

There are many rules that we are not using, and we are guided by the fact of looking *forward*. If we know that we are looking for $aabb$, then the most logic step is to use $S \to aSb$ two times, in order to *finish* with $S \to \varepsilon$.

This type of parsers, that read from left to right, and have a “prediction” window are known as *Left to right Leftmost derivation*, or LL for its acronym in English.

Growth and no-determinism in CFGs

LL and LR parsers

On the contrary, there are parsers that read *backwards*, from right to left (regarding the grammar rules, first aSB and then $S \to$). In this way, the derivation tree runs backwards.

This kind of parsers are known as *Left to right Rightmost derivation*, or LR for its acronym in English.

But how do we convert a CFG to an PA and vice versa?

Growth and no-determinism in CFGs

LL and LR parsers

On the contrary, there are parsers that read *backwards*, from right to left (regarding the grammar rules, first aSB and then $S \rightarrow$). In this way, the derivation tree runs backwards.

This kind of parsers are known as *Left to right Rightmost derivation*, or LR for its acronym in English.

But how do we convert a CFG to an PA and vice versa?

Growth and no-determinism in CFGs

LL and LR parsers

On the contrary, there are parsers that read *backwards*, from right to left (regarding the grammar rules, first aSB and then $S \rightarrow$). In this way, the derivation tree runs backwards.

This kind of parsers are known as *Left to right Rightmost derivation*, or LR for its acronym in English.

But how do we convert a CFG to an PA and vice versa?

Multiple ways in the literature

LL and LR parsers

Like in the previous case, different people use different ways:

1. Brena (2003) difference between the type of parser, and convert a grammar of a certain type to an PA using an algorithm that depends on the type of the parser.
2. Ullman (1979), one of the authors of the most used book for this course worldwide, uses another version, only for LL parsers and is the one that we will use.

Multiple ways in the literature

LL and LR parsers

Like in the previous case, different people use different ways:

1. Brena (2003) difference between the type of parser, and convert a grammar of a certain type to an PA using an algorithm that depends on the type of the parser.
2. Ullman (1979), one of the authors of the most used book for this course worldwide, uses another version, only for LL parsers and is the one that we will use.

Equivalence of PA and CFGs

Let L be a language of a CFG G.

To build a PA M that accepts L, the following must be considered:

- M has **a state** q
- The **tape alphabet** of M, Σ is equal to the terminals of G
- The **stack alphabet** of $M, \Gamma \subseteq V \cup \Sigma$
- The **initial symbol (of the stack)** of M remains $\$$ (like before)

The **transition function** can be calculates with respect to it (like before)

The **accepting conditions** are:

- The stack is empty (like before)
- The tape reached the end (like before)

Equivalence of PA and CFGs

Let L be a language of a CFG G.

To build a PA M that accepts L, the following must be considered:

- M has **a state** q
- The **tape alphabet** of M, Σ is equal to the terminals of G
- The **stack alphabet** of $M, \Gamma \subseteq V \cup \Sigma$
- The **initial symbol (of the stack)** of M remains $\$$ (like before)

The **transition function** can be calculates with respect to it (like before)

The **accepting conditions** are:

- The stack is empty (like before)
- The tape reached the end (like before)

Equivalence of PA and CFGs

Let L be a language of a CFG G.

To build a PA M that accepts L, the following must be considered:

- M has **a state** q
- The **tape alphabet** of M, Σ is equal to the terminals of G
- The **stack alphabet** of $M, \Gamma \subseteq V \cup \Sigma$
- The **initial symbol (of the stack)** of M remains $\$$ (like before)

The **transition function** can be calculates with respect to it (like before)

The **accepting conditions** are:

- The stack is empty (like before)
- The tape reached the end (like before)

Equivalence of PA and CFGs

Let L be a language of a CFG G.

To build a PA M that accepts L, the following must be considered:

- M has **a state** q
- The **tape alphabet** of M, Σ is equal to the terminals of G
- The **stack alphabet** of $M, \Gamma \subseteq V \cup \Sigma$
- The **initial symbol (of the stack)** of M remains $\$$ (like before)

The **transition function** can be calculates with respect to it (like before)

The **accepting conditions** are:

- The stack is empty (like before)
- The tape reached the end (like before)

Equivalence of PA and CFGs

Let L be a language of a CFG G.

To build a PA M that accepts L, the following must be considered:

- M has **a state** q
- The **tape alphabet** of M, Σ is equal to the terminals of G
- The **stack alphabet** of $M, \Gamma \subseteq V \cup \Sigma$
- The **initial symbol (of the stack)** of M remains $\$$ (like before)

The **transition function** can be calculates with respect to it (like before)

The **accepting conditions** are:

- The stack is empty (like before)
- The tape reached the end (like before)

Equivalence of PA and CFGs

Let L be a language of a CFG G.

To build a PA M that accepts L, the following must be considered:

- M has **a state** q
- The **tape alphabet** of M, Σ is equal to the terminals of G
- The **stack alphabet** of $M, \Gamma \subseteq V \cup \Sigma$
- The **initial symbol (of the stack)** of M remains $\$$ (like before)

The **transition function** can be calculates with respect to it (like before)

The **accepting conditions** are:

- The stack is empty (like before)
- The tape reached the end (like before)

Equivalence of PA and CFGs

Let L be a language of a CFG G.

To build a PA M that accepts L, the following must be considered:

- M has **a state** q
- The **tape alphabet** of M, Σ is equal to the terminals of G
- The **stack alphabet** of $M, \Gamma \subseteq V \cup \Sigma$
- The **initial symbol (of the stack)** of M remains $\$$ (like before)

The **transition function** can be calculates with respect to it (like before)

The **accepting conditions** are:

- The stack is empty (like before)
- The tape reached the end (like before)

Equivalence of PA and CFGs

Let L be a language of a CFG G.

To build a PA M that accepts L, the following must be considered:

- M has **a state** q
- The **tape alphabet** of M, Σ is equal to the terminals of G
- The **stack alphabet** of $M, \Gamma \subseteq V \cup \Sigma$
- The **initial symbol (of the stack)** of M remains $\$$ (like before)

The **transition function** can be calculates with respect to it (like before)

The **accepting conditions** are:

- The stack is empty (like before)
- The tape reached the end (like before)

Equivalence of PA and CFGs

Let L be a language of a CFG G.

To build a PA M that accepts L, the following must be considered:

- M has **a state** q
- The **tape alphabet** of M, Σ is equal to the terminals of G
- The **stack alphabet** of $M, \Gamma \subseteq V \cup \Sigma$
- The **initial symbol (of the stack)** of M remains $\$$ (like before)

The **transition function** can be calculates with respect to it (like before)

The **accepting conditions** are:

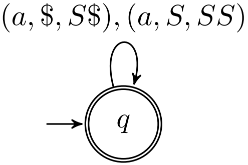
- The stack is empty (like before)
- The tape reached the end (like before)

Example: $\{a^n b^n\}$

Formal definition and design of PAs

1. $S \rightarrow aSb$
2. $S \rightarrow ab$

- $Q = \{q\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{\$, S\}$
- $\delta =$
  - $((q, a, \$)(q, S\$))$
  - $((q, a, S)(q, SS))$
  - $((q, b, S)(q, \varepsilon))$
  - $((q, \Box, \$), (q, \varepsilon))$
- $q = q$
- $F = \{q\}$



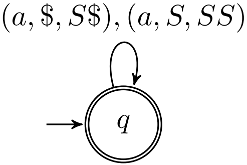
Is this approach correct? What would happen with $aabb$? And with abab?

Example: $\{a^n b^n\}$

Formal definition and design of PAs

1. $S \rightarrow aSb$
2. $S \rightarrow ab$

- $Q = \{q\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{\$, S\}$
- $\delta =$
  - $((q, a, \$)(q, S\$))$
  - $((q, a, S)(q, SS))$
  - $((q, b, S)(q, \varepsilon))$
  - $((q, \square, \$), (q, \varepsilon))$
- $q = q$
- $F = \{q\}$



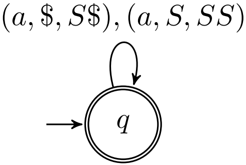
Is this approach correct? What would happen with $aabb$? And with abab?

Example: $\{a^n b^n\}$

Formal definition and design of PAs

1. $S \rightarrow aSb$
2. $S \rightarrow ab$

- $Q = \{q\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{\$, S\}$
- $\delta =$
  - $((q, a, \$)(q, S\$))$
  - $((q, a, S)(q, SS))$
  - $((q, b, S)(q, \varepsilon))$
  - $((q, \square, \$), (q, \varepsilon))$
- $q = q$
- $F = \{q\}$



Is this approach correct? What would happen with $aabb$? And with abab?

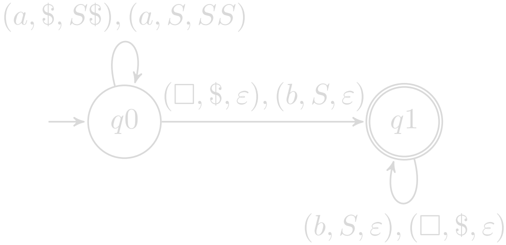
Example: $\{a^n b^n\}$

Formal definition and design of PAs

We have clearly exhausted the possible combinations between $\Sigma \times \Gamma$ ...

With which other set that has the PA can we make Cartesian product to obtain more computing possibilities?

With the **set of states**.



$(a, \$, S\$), (a, S, SS)$
$(\square, \$, \varepsilon), (b, S, \varepsilon)$
$q0$
$q1$
$(b, S, \varepsilon), (\square, \$, \varepsilon)$

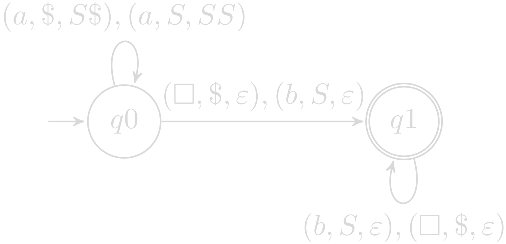
Example: $\{a^n b^n\}$

Formal definition and design of PAs

We have clearly exhausted the possible combinations between $\Sigma \times \Gamma$ ...

With which other set that has the PA can we make Cartesian product to obtain more computing possibilities?

With the **set of states**.



$(a, \$, S\$), (a, S, SS)$
$q0$
$(\Box, \$, \varepsilon), (b, S, \varepsilon)$
$q1$
$(b, S, \varepsilon), (\Box, \$, \varepsilon)$

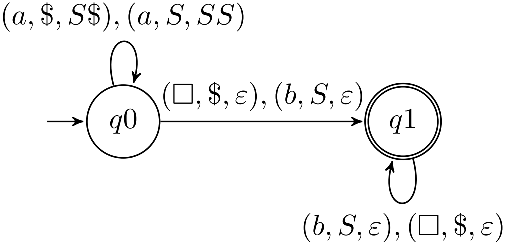
Example: $\{a^n b^n\}$

Formal definition and design of PAs

We have clearly exhausted the possible combinations between $\Sigma \times \Gamma$ ...

With which other set that has the PA can we make Cartesian product to obtain more computing possibilities?

With the **set of states**.



$(a, \$, S\$), (a, S, SS)$
$(\Box, \$, \varepsilon), (b, S, \varepsilon)$
$q0$
$q1$
$(b, S, \varepsilon), (\Box, \$, \varepsilon)$

Conversion between CFGs and PAs

Equivalence of PA and CFGs

Theorem

A language is context-free **if and only if** *some pushdown automaton recognises it.*

In order to prove an *if and only if* theorem, it is necessary to prove both directions:

- Context-free language recognised by some pushdown automaton.
- Pushdown automaton recognises some context-free language.

CFG to PA

Conversion between CFGs and PAs

Lemma

If a language is context-free, then some pushdown automaton recognises it.

The general idea or informal steps are:

1. Place the marker symbol \$ and the start variable in the stack.
2. Repeat the following steps forever:
   a. If the top of the stack is a variable symbol A, non-deterministically select one of the rules for A, and substitute A by the string on the right-hand side of the rule.
   b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If the match, repeat. If they do not match, reject on this branch on the non-determinism.
   c. If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if has all been read.

CFG to PA

Conversion between CFGs and PAs

Lemma

If a language is context-free, then some pushdown automaton recognises it.

The general idea or informal steps are:

1. Place the marker symbol \$ and the start variable in the stack.
2. Repeat the following steps forever:
   a. If the top of the stack is a variable symbol A, non-deterministically select one of the rules for A, and substitute A by the string on the right-hand side of the rule.
   b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If the match, repeat. If they do not match, reject on this branch on the non-determinism.
   c. If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if has all been read.

CFG to PA

Conversion between CFGs and PAs

Lemma

If a language is context-free, then some pushdown automaton recognises it.

The general idea or informal steps are:

1. Place the marker symbol \$ and the start variable in the stack.
2. Repeat the following steps forever:
   a. If the top of the stack is a variable symbol A, non-deterministically select one of the rules for A, and substitute A by the string on the right-hand side of the rule.
   b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If the match, repeat. If they do not match, reject on this branch on the non-determinism.
   c. If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if has all been read.

CFG to PA

Conversion between CFGs and PAs

Lemma

If a language is context-free, then some pushdown automaton recognises it.

The general idea or informal steps are:

1. Place the marker symbol \$ and the start variable in the stack.
2. Repeat the following steps forever:
   - a. If the top of the stack is a variable symbol A, non-deterministically select one of the rules for A, and substitute A by the string on the right-hand side of the rule.
   - b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If the match, repeat. If they do not match, reject on this branch on the non-determinism.
   - c. If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if has all been read.

CFG to PA

Conversion between CFGs and PAs

Lemma

If a language is context-free, then some pushdown automaton recognises it.

The general idea or informal steps are:

1. Place the marker symbol $ and the start variable in the stack.
2. Repeat the following steps forever:
   a. If the top of the stack is a variable symbol A, non-deterministically select one of the rules for A, and substitute A by the string on the right-hand side of the rule.
   b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If the match, repeat. If they do not match, reject on this branch on the non-determinism.
   c. If the top of stack is the symbol $\$$, enter the accept state. Doing so accepts the input if has all been read.

CFG to PA

Conversion between CFGs and PAs

Lemma

If a language is context-free, then some pushdown automaton recognises it.

The general idea or informal steps are:

1. Place the marker symbol \$ and the start variable in the stack.
2. Repeat the following steps forever:
   a. If the top of the stack is a variable symbol A, non-deterministically select one of the rules for A, and substitute A by the string on the right-hand side of the rule.
   b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If the match, repeat. If they do not match, reject on this branch on the non-determinism.
   c. If the top of stack is the symbol $\$$, enter the accept state. Doing so accepts the input if has all been read.

CFG to PA I

Conversion between CFGs and PAs

Formal details of the construction of the PA $M = (Q, \Sigma, \Gamma, \delta, q, F)$:

0. Let $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$, where q_{start} is the start state, q_{loop} is the state main body, q_{accept} is the only accepting state, and E is any other state needed in the q_{loop}. δ is defined as follows.
    1. The stack is initialised with the symbol $\$$ and the initial variable of the rule, e.g. S. Then implement step 1 in the informal description will look like $\delta((q_{start}, \varepsilon, \$)(q_{loop}, S\$))$.
    2. Now create the transitions for the q_{loop} from step 2.

Case a. The top of the stack contains a variable:

$$\delta((q_{loop}, \varepsilon, A)(q_{loop}, w)) \mid A \to w \text{ is a rule in R.}$$

Case b. The top of the stack contains a terminal:

$$\delta((q_{loop}, a, a)(q_{loop}, \varepsilon)).$$

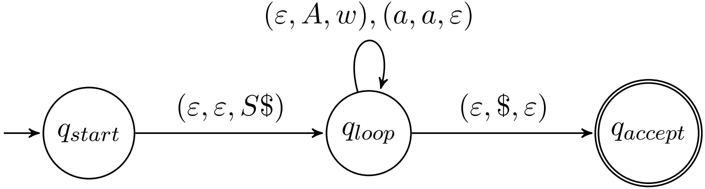
Case c. The empty stack symbol $\$$ is on the top of the stack:

$$\delta((q_{loop}, \varepsilon, \$)(q_{accept}, \varepsilon)).$$

CFG to PA II

Conversion between CFGs and PAs

3. The state diagram would look like this:



$(\varepsilon, A, w), (a, a, \varepsilon)$
q_{start}
$(\varepsilon, \varepsilon, S\$)$
q_{loop}
$(\varepsilon, \$, \varepsilon)$
q_{accept}

Example

CFG to PA

Let us construct a PA from the following CFG

$$
\begin{aligned}
S &\to aTb \mid b \\
T &\to Ta \mid \varepsilon
\end{aligned}
$$

The transition function is shown in the following diagram:

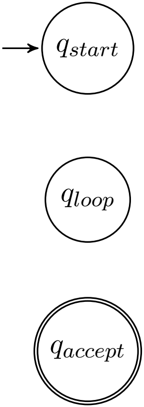
Example

CFG to PA

Let us construct a PA from the following CFG

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

The transition function is shown in the following diagram:



Rule (0)

Define $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$.

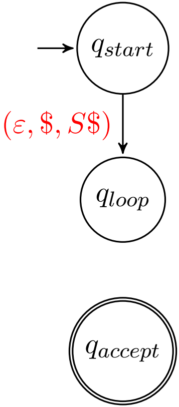
Example

CFG to PA

Let us construct a PA from the following CFG

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

The transition function is shown in the following diagram:



Rule (1)

Define $\delta((q_{start}, \varepsilon, \$)(q_{loop}, S\$))$

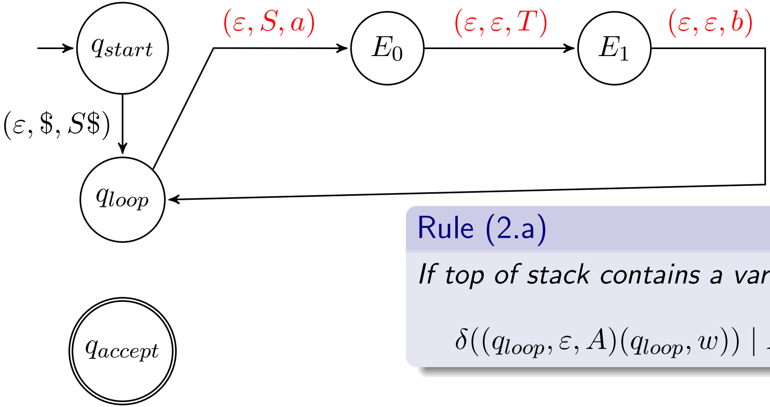
Example

CFG to PA

Let us construct a PA from the following CFG

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

The transition function is shown in the following diagram:



Rule (2.a)

If top of stack contains a variable:

$$\delta((q_{loop}, \varepsilon, A)(q_{loop}, w)) \mid A \to w.$$

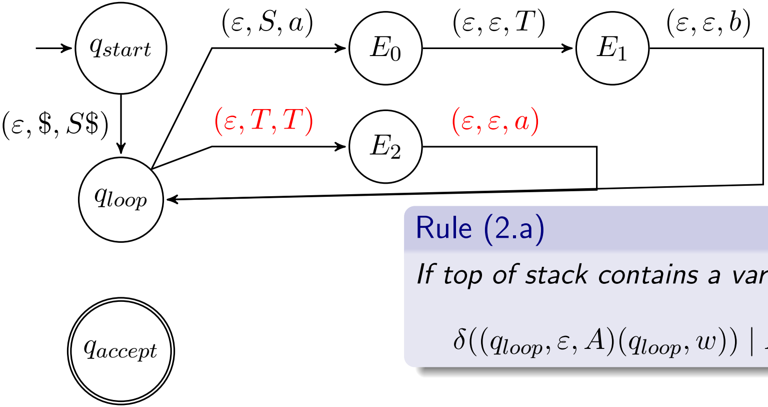
Example

CFG to PA

Let us construct a PA from the following CFG

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

The transition function is shown in the following diagram:



Rule (2.a)

If top of stack contains a variable:

$$\delta((q_{loop}, \varepsilon, A)(q_{loop}, w)) \mid A \to w.$$

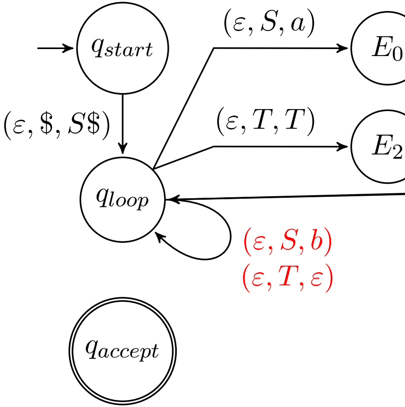
Example

CFG to PA

Let us construct a PA from the following CFG

$$S \rightarrow aTb \mid b$$
$$T \rightarrow Ta \mid \varepsilon$$

The transition function is shown in the following diagram:



Rule (2.a)

If top of stack contains a variable:

$$\delta((q_{loop}, \varepsilon, A)(q_{loop}, w)) \mid A \rightarrow w.$$

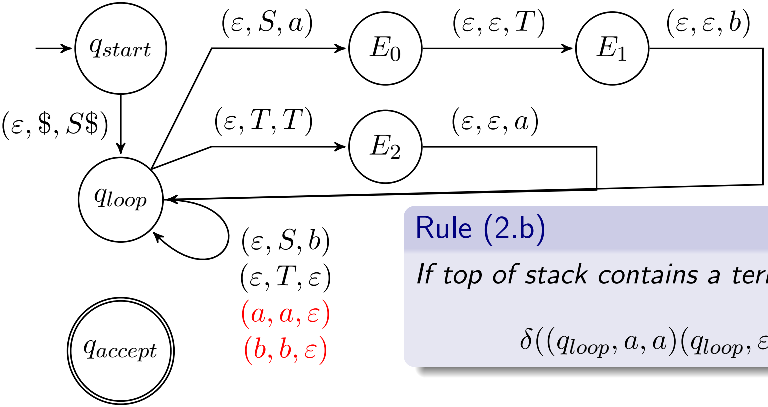
Example

CFG to PA

Let us construct a PA from the following CFG

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

The transition function is shown in the following diagram:



Rule (2.b)

If top of stack contains a terminal:

$$\delta((q_{loop}, a, a)(q_{loop}, \varepsilon))$$

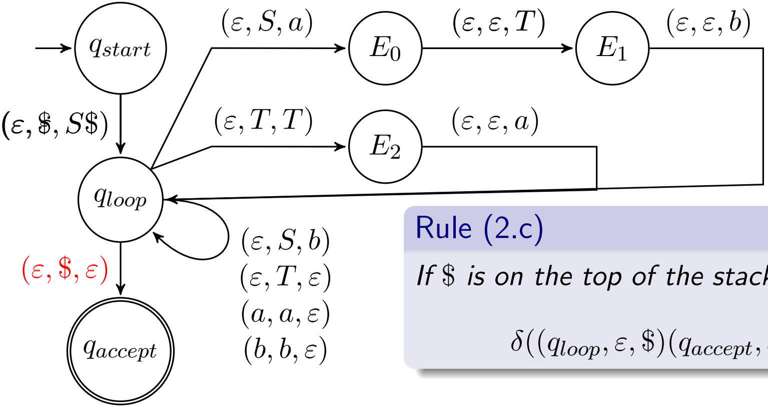
Example

CFG to PA

Let us construct a PA from the following CFG

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

The transition function is shown in the following diagram:



Rule (2.c)

If $\$$ is on the top of the stack:

$$\delta((q_{loop}, \varepsilon, \$)(q_{accept}, \varepsilon))$$

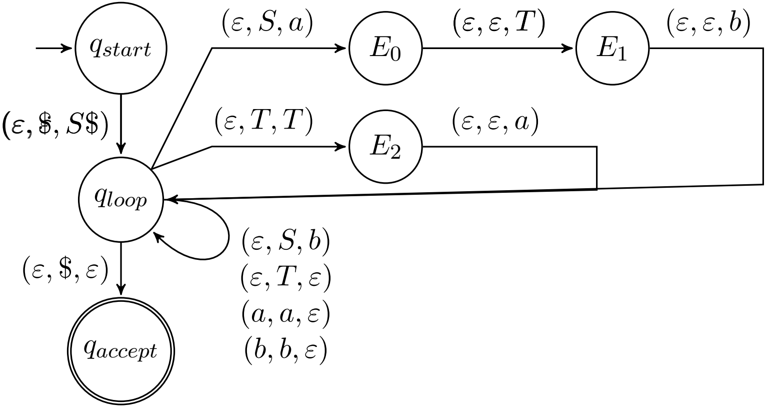
Example

CFG to PA

Let us construct a PA from the following CFG

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

The transition function is shown in the following diagram:



PA to CFG I

Conversion between CFGs and PAs

Lemma

If a pushdown automaton recognises some language, then it is context-free.

Given the PA $M = (Q, \Sigma_M, \Gamma, \delta, q, F)$ convert it to its equivalent context-free grammar $G = (V, \Sigma_G, R, S)$. The general idea or informal steps are:

1. Assign the corresponding Σ_G, with the tape language in M, in this case Σ_M, so $\Sigma_G = \Sigma_M$.
2. Assign the corresponding S in G to the initial stack symbol introduced in M, normally $S = S$.
3. Define the set of variables V. The first element in V we know is S. The next variables are all combinations between the states and stack symbols with the form $[Q\Gamma Q]$.

PA to CFG II

Conversion between CFGs and PAs

4. Define the set of prediction rules R. R is based on δ from M. There are 2 main transitions:
   1. Does not push anything into the stack after transition: $(q_0, a, \$)(q_0, \varepsilon)$.
   2. Push something into the stack: $(q_0, a, \$)(q_0, \$a)$.

We know that the rules will have the form of $[Q\Gamma Q]$ by step 3, now the transitions defined in step 4 can be converted into rules:

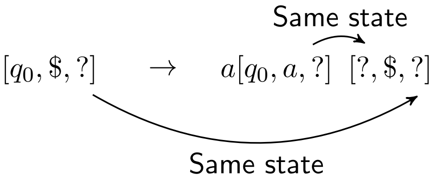
$$(q_0, a, \$)(q_0, \varepsilon) = [q_0, \$, q_0] \rightarrow a$$
$$(q_0, a, \$)(q_0, a\$) = [q_0, \$, ?] \rightarrow a[q_0, a, ?][?, \$, ?]$$

General way to define the rules from transitions

PA to CFG

For the rules where 2 elements are being pushed into the stack:

$$(q_0, a, \$)(q_0, a\$)$$



Same state

$$[q_0, \$, ?] \quad \rightarrow \quad a[q_0, a, ?]\ [?, \$, ?]$$

Same state

General way to define the rules from transitions

PA to CFG

For the rules where 3 elements are being pushed into the stack:

$$(q_0, a, \$)(q_1, axz)$$

$$\begin{tikzcd}
{[q_0, \$, ?]} \arrow[rrrr, bend right] & \rightarrow & a[q_0, a, ?] \arrow[r, bend left] & {[?, x, ?]} \arrow[r, bend left] & {[?, z, ?]}
\end{tikzcd}$$

General way to define the rules from transitions

PA to CFG

For the rules where 3 elements are being pushed into the stack:

$$(q_0, a, \$)(q_1, axz)$$

$$\begin{tikzcd}
{[q_0, \$, ?]} \arrow[rrrr, bend right] & \rightarrow & {a[q_0, a, ?]} \arrow[r, bend left] & {[?, x, ?]} \arrow[r, bend left] & {[?, z, ?]}
\end{tikzcd}$$

For the rules where n elements are being pushed into the stack is the same procedure.

Example

PA to CFG

From the previous example

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{\$, S, T\}$
- $\delta =$
  - $((q_0, \varepsilon, \$)(q_1, S\$))$
  - $((q_1, \varepsilon, S)(q_1, aTb))$
  - $((q_1, \varepsilon, T)(q_1, Ta))$
  - $((q_1, \varepsilon, S)(q_1, b))$
  - $((q_1, \varepsilon, T)(q_1, \varepsilon))$
  - $((q_1, a, a)(q_1, \varepsilon))$
  - $((q_1, b, b)(q_1, \varepsilon))$
  - $((q_1, \varepsilon, \$)(q_2, \varepsilon))$
- $q = q_0$
- $F = \{q_2\}$

Example

PA to CFG

From the previous example

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{\$, S, T\}$
- $\delta =$
  - $((q_0, \varepsilon, \$)(q_1, S\$))$
  - $((q_1, \varepsilon, S)(q_1, aTb))$
  - $((q_1, \varepsilon, T)(q_1, Ta))$
  - $((q_1, \varepsilon, S)(q_1, b))$
  - $((q_1, \varepsilon, T)(q_1, \varepsilon))$
  - $((q_1, a, a)(q_1, \varepsilon))$
  - $((q_1, b, b)(q_1, \varepsilon))$
  - $((q_1, \varepsilon, \$)(q_2, \varepsilon))$
- $q = q_0$
- $F = \{q_2\}$

1. $\Sigma_G = \Sigma = \{a, b\}$.
2. $S = S$.
3. Now we define V and R at the same time.

Example I

PA to CFG

Define V and R.
For $\delta(q_0, \varepsilon, \$) = (q_1, S\$)$ we have:

$$[q_0, \$, q_0] \to \varepsilon[q_1, S, q_0][q_0, \$, q_0]$$
$$[q_0, \$, q_0] \to \varepsilon[q_1, S, q_1][q_1, \$, q_0]$$
$$[q_0, \$, q_0] \to \varepsilon[q_1, S, q_2][q_2, \$, q_0]$$
$$[q_0, \$, q_1] \to \varepsilon[q_1, S, q_0][q_0, \$, q_1]$$
$$[q_0, \$, q_1] \to \varepsilon[q_1, S, q_1][q_1, \$, q_1]$$
$$[q_0, \$, q_1] \to \varepsilon[q_1, S, q_2][q_2, \$, q_1]$$
$$[q_0, \$, q_2] \to \varepsilon[q_1, S, q_0][q_0, \$, q_2]$$
$$[q_0, \$, q_2] \to \varepsilon[q_1, S, q_1][q_1, \$, q_2]$$
$$[q_0, \$, q_2] \to \varepsilon[q_1, S, q_2][q_2, \$, q_2]$$

Example II

PA to CFG

For $\delta(q_1, \varepsilon, S) = (q_1, aTb)$ we have:

$$
\begin{aligned}
[q_1, S, q_0] &\to \varepsilon[q_1, a, q_0][q_0, T, q_0][q_0, b, q_0] \\
[q_1, S, q_0] &\to \varepsilon[q_1, a, q_0][q_0, T, q_1][q_1, b, q_0] \\
[q_1, S, q_0] &\to \varepsilon[q_1, a, q_0][q_0, T, q_2][q_2, b, q_0] \\
[q_1, S, q_0] &\to \varepsilon[q_1, a, q_1][q_1, T, q_0][q_0, b, q_0] \\
[q_1, S, q_0] &\to \varepsilon[q_1, a, q_1][q_1, T, q_1][q_1, b, q_0] \\
[q_1, S, q_0] &\to \varepsilon[q_1, a, q_1][q_1, T, q_2][q_2, b, q_0] \\
[q_1, S, q_0] &\to \varepsilon[q_1, a, q_2][q_2, T, q_0][q_0, b, q_0] \\
[q_1, S, q_0] &\to \varepsilon[q_1, a, q_2][q_2, T, q_1][q_1, b, q_0] \\
[q_1, S, q_0] &\to \varepsilon[q_1, a, q_2][q_2, T, q_2][q_2, b, q_0]
\end{aligned}
$$

Example III

PA to CFG

For $\delta(q_1, \varepsilon, S) = (q_1, aTb)$ we have:

$$[q_1, S, q_1] \to \varepsilon[q_1, a, q_0][q_0, T, q_0][q_0, b, q_1]$$
$$[q_1, S, q_1] \to \varepsilon[q_1, a, q_0][q_0, T, q_1][q_1, b, q_1]$$
$$[q_1, S, q_1] \to \varepsilon[q_1, a, q_0][q_0, T, q_2][q_2, b, q_1]$$
$$[q_1, S, q_1] \to \varepsilon[q_1, a, q_1][q_1, T, q_0][q_0, b, q_1]$$
$$[q_1, S, q_1] \to \varepsilon[q_1, a, q_1][q_1, T, q_1][q_1, b, q_1]$$
$$[q_1, S, q_1] \to \varepsilon[q_1, a, q_1][q_1, T, q_2][q_2, b, q_1]$$
$$[q_1, S, q_1] \to \varepsilon[q_1, a, q_2][q_2, T, q_0][q_0, b, q_1]$$
$$[q_1, S, q_1] \to \varepsilon[q_1, a, q_2][q_2, T, q_1][q_1, b, q_1]$$
$$[q_1, S, q_1] \to \varepsilon[q_1, a, q_2][q_2, T, q_2][q_2, b, q_1]$$

Example IV

PA to CFG

For $\delta(q_1, \varepsilon, S) = (q_1, aTb)$ we have:

$$
\begin{aligned}
[q_1, S, q_2] &\to \varepsilon[q_1, a, q_0][q_0, T, q_0][q_0, b, q_2] \\
[q_1, S, q_2] &\to \varepsilon[q_1, a, q_0][q_0, T, q_1][q_1, b, q_2] \\
[q_1, S, q_2] &\to \varepsilon[q_1, a, q_0][q_0, T, q_2][q_2, b, q_2] \\
[q_1, S, q_2] &\to \varepsilon[q_1, a, q_1][q_1, T, q_0][q_0, b, q_2] \\
[q_1, S, q_2] &\to \varepsilon[q_1, a, q_1][q_1, T, q_1][q_1, b, q_2] \\
[q_1, S, q_2] &\to \varepsilon[q_1, a, q_1][q_1, T, q_2][q_2, b, q_2] \\
[q_1, S, q_2] &\to \varepsilon[q_1, a, q_2][q_2, T, q_0][q_0, b, q_2] \\
[q_1, S, q_2] &\to \varepsilon[q_1, a, q_2][q_2, T, q_1][q_1, b, q_2] \\
[q_1, S, q_2] &\to \varepsilon[q_1, a, q_2][q_2, T, q_2][q_2, b, q_2]
\end{aligned}
$$

Example V

PA to CFG

For $\delta(q_1, \varepsilon, T) = (q_1, Ta)$ we have:

$$
\begin{aligned}
[q_1, T, q_0] &\to \varepsilon[q_1, T, q_0][q_0, a, q_0] \\
[q_1, T, q_0] &\to \varepsilon[q_1, T, q_1][q_1, a, q_0] \\
[q_1, T, q_0] &\to \varepsilon[q_1, T, q_2][q_2, a, q_0] \\
[q_1, T, q_1] &\to \varepsilon[q_1, T, q_0][q_0, a, q_1] \\
[q_1, T, q_1] &\to \varepsilon[q_1, T, q_1][q_1, a, q_1] \\
[q_1, T, q_1] &\to \varepsilon[q_1, T, q_2][q_2, a, q_1] \\
[q_1, T, q_2] &\to \varepsilon[q_1, T, q_0][q_0, a, q_2] \\
[q_1, T, q_2] &\to \varepsilon[q_1, T, q_1][q_1, a, q_2] \\
[q_1, T, q_2] &\to \varepsilon[q_1, T, q_2][q_2, a, q_2]
\end{aligned}
$$

Example VI

PA to CFG

For $\delta(q_1, \varepsilon, S) = (q_1, b)$ we have:

$$[q_1, S, q_0] \to \varepsilon[q_1, b, q_0]$$
$$[q_1, S, q_1] \to \varepsilon[q_1, b, q_1]$$
$$[q_1, S, q_2] \to \varepsilon[q_1, b, q_2]$$

For $\delta(q_1, \varepsilon, T) = (q_1, \varepsilon)$ we have:

$$[q_1, T, q_1] \to \varepsilon$$

Example VII

PA to CFG

For $\delta(q_1, a, a) = (q_1, \varepsilon)$ we have:

$$[q_1, a, q_1] \to a$$

For $\delta(q_1, b, b) = (q_1, \varepsilon)$ we have:

$$[q_1, b, q_1] \to b$$

For $\delta(q_1, \varepsilon, \$) = (q_2, \varepsilon)$ we have:

$$[q_1, \$, q_2] \to \varepsilon$$

Example

PA to CFG

Now, we must start from one the following rules

$$S \to [q_0, \$, q_0] \qquad S \to [q_0, \$, q_1] \qquad S \to [q_0, \$, q_2].$$

From the previous step, there are many rules that we do not need, so we need to remove them and just focus on the ones that are helpful to construct the CFG.

- Eliminate the non-generating symbols.
- Eliminate the non-reachable rules.

Example

PA to CFG

Eliminate the non-generating rules.

$$[q_1, S, q_0] \to \varepsilon[q_1, b, q_0][q_0, T, q_0][q_0, a, q_0].$$

There is no rule with the form $[q_0, a, q_0] \to a[x, y, z]$, so all the rules containing $[q_0, a, q_0]$ can be eliminated (the same for $[q_1, a, q_0]$ and so on).

Eliminate the non-reachable rules.
If you remove for example:

$$[q_1, S, q_2] \to a[x, y, z]$$

Because there is no way to accomplish it, then all rules with the form

$$[x, y, z] \to a[q_1, S, q_2]$$

Can be eliminated since there is not rule to produce it.

Example

PA to CFG

Putting all together we have:

1. $\Sigma_G = \Sigma = \{a, b\}$.
2. $S = S$.
3. $V = \{S, [q_0, \$, q_2], [q_1, S, q_1], [q_1, \$, q_2], [q_1, a, q_1], [q_1, T, q_1], [q_1, b, q_1]\}$.
4. $R =$
    - $S \to [q_0, \$, q_2]$
    - $[q_0, \$, q_2] \to \varepsilon[q_1, S, q_1][q_1, \$, q_2]$
    - $[q_1, S, q_1] \to \varepsilon[q_1, a, q_1][q_1, T, q_1][q_1, b, q_1]$
    - $[q_1, S, q_1] \to \varepsilon[q_1, b, q_1]$
    - $[q_1, T, q_1] \to \varepsilon[q_1, T, q_1][q_1, a, q_1]$
    - $[q_1, T, q_1] \to \varepsilon$
    - $[q_1, a, q_1] \to a$
    - $[q_1, b, q_1] \to b$
    - $[q_1, \$, q_2] \to \varepsilon$

Example

PA to CFG

How can we avoid doing all the comparisons?

Is there a way to just create the rules needed?

- $\delta =$
  - $((q_0, \varepsilon, \$)(q_1, S\$))$
  - $((q_1, \varepsilon, S)(q_1, aTb))$
  - $((q_1, \varepsilon, T)(q_1, Ta))$
  - $((q_1, \varepsilon, S)(q_1, b))$
  - $((q_1, \varepsilon, T)(q_1, \varepsilon))$
  - $((q_1, a, a)(q_1, \varepsilon))$
  - $((q_1, b, b)(q_1, \varepsilon))$
  - $((q_1, \varepsilon, \$)(q_2, \varepsilon))$
- $R =$
  - $S \to [q_0, \$, q_2]$
  - $[q_0, \$, q_2] \to \varepsilon[q_1, S, q_1][q_1, \$, q_2]$
  - $[q_1, S, q_1] \to \varepsilon[q_1, a, q_1][q_1, T, q_1][q_1, b, q_1]$
  - $[q_1, S, q_1] \to \varepsilon[q_1, b, q_1]$
  - $[q_1, T, q_1] \to \varepsilon[q_1, T, q_1][q_1, a, q_1]$
  - $[q_1, T, q_1] \to \varepsilon$
  - $[q_1, a, q_1] \to a$
  - $[q_1, b, q_1] \to b$
  - $[q_1, \$, q_2] \to \varepsilon$

Do you observe a pattern? Can we create the rules immediately from the transitions?

Example

PA to CFG

How can we avoid doing all the comparisons?

Is there a way to just create the rules needed?

- $\delta =$
  - $((q_0, \varepsilon, \$)(q_1, S\$))$
  - $((q_1, \varepsilon, S)(q_1, aTb))$
  - $((q_1, \varepsilon, T)(q_1, Ta))$
  - $((q_1, \varepsilon, S)(q_1, b))$
  - $((q_1, \varepsilon, T)(q_1, \varepsilon))$
  - $((q_1, a, a)(q_1, \varepsilon))$
  - $((q_1, b, b)(q_1, \varepsilon))$
  - $((q_1, \varepsilon, \$)(q_2, \varepsilon))$
- $R =$
  - $S \to [q_0, \$, q_2]$
  - $[q_0, \$, q_2] \to \varepsilon[q_1, S, q_1][q_1, \$, q_2]$
  - $[q_1, S, q_1] \to \varepsilon[q_1, a, q_1][q_1, T, q_1][q_1, b, q_1]$
  - $[q_1, S, q_1] \to \varepsilon[q_1, b, q_1]$
  - $[q_1, T, q_1] \to \varepsilon[q_1, T, q_1][q_1, a, q_1]$
  - $[q_1, T, q_1] \to \varepsilon$
  - $[q_1, a, q_1] \to a$
  - $[q_1, b, q_1] \to b$
  - $[q_1, \$, q_2] \to \varepsilon$

Do you observe a pattern? Can we create the rules immediately from the transitions?

Example

PA to CFG

How can we avoid doing all the comparisons?

Is there a way to just create the rules needed?

- $\delta =$
  - $((q_0, \varepsilon, \$)(q_1, S\$))$
  - $((q_1, \varepsilon, S)(q_1, aTb))$
  - $((q_1, \varepsilon, T)(q_1, Ta))$
  - $((q_1, \varepsilon, S)(q_1, b))$
  - $((q_1, \varepsilon, T)(q_1, \varepsilon))$
  - $((q_1, a, a)(q_1, \varepsilon))$
  - $((q_1, b, b)(q_1, \varepsilon))$
  - $((q_1, \varepsilon, \$)(q_2, \varepsilon))$
- $R =$
  - $S \rightarrow [q_0, \$, q_2]$
  - $[q_0, \$, q_2] \rightarrow \varepsilon[q_1, S, q_1][q_1, \$, q_2]$
  - $[q_1, S, q_1] \rightarrow \varepsilon[q_1, a, q_1][q_1, T, q_1][q_1, b, q_1]$
  - $[q_1, S, q_1] \rightarrow \varepsilon[q_1, b, q_1]$
  - $[q_1, T, q_1] \rightarrow \varepsilon[q_1, T, q_1][q_1, a, q_1]$
  - $[q_1, T, q_1] \rightarrow \varepsilon$
  - $[q_1, a, q_1] \rightarrow a$
  - $[q_1, b, q_1] \rightarrow b$
  - $[q_1, \$, q_2] \rightarrow \varepsilon$

Do you observe a pattern? Can we create the rules immediately from the transitions?