Wanted:

- Similar to the DFA characterization of type 3 languages, we’d like a **machine characterization** of type 2 languages.

- We know that finite-state machines are inadequate.

- We need to add an extra memory component, one that permits **unbounded storage**.
A Proper Context-Free Language

- Example: \{xcx^R \mid x \in \mathcal{S}^*\} where c \in \mathcal{S}.

- Supposing \mathcal{S} = \{a, b\}, give a context-free grammar for this language.

- Is this language regular?

Stacks to the Rescue

- By adding a stack to a FSA, we can accept non-regular languages.

- Example: \{xcx^R \mid x \in \mathcal{S}^*\} where c \in \mathcal{S}.

  - Begin reading symbols.
  - Until we encounter a 'c', push the symbols read onto a stack.
  - After 'c' is encountered, pop the symbols from the stack, comparing with the next symbol read, if any.
  - Accept when the stack is empty.
  - Have to check for no c's, more than one c, etc. and reject these cases.
PDA’s

• The type of behavior described on the preceding slide is that of a (deterministic) PDA (DPDA).

• In general, PDA’s are allowed to be **non-deterministic**.

• The situation for PDA’s is different from FSA’s: there is **no subset construction**.

PDA Defined: \((Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)\)

• \(Q\) is a finite set of **control states**
• \(\Sigma\) is the **input** alphabet
• \(\Gamma\) is the **stack** alphabet
• \(q_0\) is the **initial** control state
• \(Z_0\) is the **initial** stack symbol
• \(A \subseteq Q\) is the set of **accepting** control states
• \(\delta\) is the **state transition relation** (or, in the case of a DPDA, **function**)

\[\delta: Q \times (\Sigma \times \{\Gamma\}) \times \Gamma^* \rightarrow \text{ finite subsets of } Q \times \Gamma^*\]
Say What?

PDA Diagram

The state is generally of the form \((q, \emptyset \in Q \times \emptyset^*)\), i.e. (control-state, stack-contents).

Sometimes this is called a configuration or instantaneous description (ID), to distinguish it from the control state. I prefer to call it the state, which is what it is.

The initial state is \((q_0, Z_0)\). We will describe \(\emptyset\) presently.

PDA Acceptance

- There are different models for acceptance.
  - Final-state acceptance means that the PDA accepts when it is in a designated control state after the input has been read.
  - Empty-stack acceptance means that the PDA accepts when its stack is empty after the input has been read.
- The choice is a matter of convenience; the two can be shown inter-convertible.
Movie of the PDA accepting a string in \( \{xcx^R \mid x \in \{a, b\}\} \)

Note that control-state is in \( q_0 \) and will stay there until \( c \) is read.

Movie of the PDA accepting a string in \( \{xcx^R \mid x \in \{a, b\}\} \)
Movie of the PDA accepting a string in \( \{xcx^R \mid x \in \{a, b\}^*\} \)

Movie of the PDA accepting a string in \( \{xcx^R \mid x \in \{a, b\}^*\} \)
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$

Note that control-state has changed to $q_1$ after reading $c$. 

Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$
Movie of the PDA accepting a string in \( \{ c x^R \mid x \in \{a, b\}^* \} \)

Movie of the PDA accepting a string in \( \{ c x^R \mid x \in \{a, b\}^* \} \)
Movie of the PDA accepting a string in \( \{xcx^R \mid x \in \{a, b\}\}^* \}

\( q_2 \) is the accepting control state.
The input is now empty.

So the PDA accepts the original input: abaacaaba.
Design of the PDA for the $xcx^R$ language

- $\mathcal{D}$: $Q \times (\{q_0, q_1\} \times \{a, b\}) \times \{Z_0\}$ finite subsets of $Q \times \{a, b\}^*$
  - $\mathcal{D}(q_0, \{a, b\} \times \{a, b, Z_0\}) = \{(q_0, \{a, b\} \times \{a, b, Z_0\})\}$
  - $\mathcal{D}(q_0, c, \{a, b, Z_0\}) = \{(q_0, \{c, Z_0\})\}$ for each $s \in \{a, b\}$
  - $\mathcal{D}(q_1, \{a, b\} \times \{a, b, Z_0\}) = \{(q_1, \{a, b\} \times \{a, b, Z_0\})\}$

- The top of the stack is the leftmost symbol in the string.
- The “for each” statements are just a way of abbreviating a larger number of separate transitions.
- For all other combinations, the RHS is empty, meaning that there is no transition defined. Unless the control state is accepting in this case, the input is rejected at this point.

$\mathcal{D}$ can also be described as a graph

- $(a, \{a\}) / a$ for each $s \in \{a, b, Z_0\}$
- $(b, \{b\}) / b$ for each $s \in \{a, b, Z_0\}$
- $(c, \{c\}) / \epsilon$ for each $s \in \{a, b, Z_0\}$
- $(\epsilon, \{\epsilon\}) / \epsilon$ for each $s \in \{a, b\}$
- $(\epsilon, Z_0) / Z_0$

Note that the "for each" statements are just a way of abbreviating a larger number of transitions.
can also be described by “transition rules”

\[ q_0, a, \square a q_0, a \] for each \( \{a, b, Z_0\} \)
\[ q_0, b, \square b q_0, b \] for each \( \{a, b, Z_0\} \)
\[ q_0, c, \square c q_1, c \] for each \( \{a, b, Z_0\} \)
\[ q_1, \square, Z_0 \square q_2, Z_0 \]

The rules with “for each” are actually abbreviations for other rules with the \( \square \) and \( \square \) variables by literal symbols.

The important thing is that the total number of rules is finite.

The \( \square \) (turnstyle) and \( \square \) * notation.

- Let \( q, q' \square Q; \ x, x' \square \square^*; \ \square \square \square^* \)

\( (q, x, \square) \square (q', x', \square) \)
means that there is a 1-step transition of the PDA from “state” \( (q, x, \square) \) to \( (q', x', \square) \).

- \( \square \) * is the transitive closure of \( \square \), meaning:
  - \( (q, x, \square) \square (q, x, \square) \)
  - If \( (q, x, \square) \square (q', x', \square) \) and \( (q', x', \square) \square (q'', x'', \square) \),
    then \( (q, x, \square) \square (q'', x'', \square) \).
The key property ("stack property") of PDA’s:

- If \((q, x, \varepsilon) \xrightarrow{\varepsilon} (q', x', \varepsilon)\)
  
  where \(x, x' \in \Sigma^*; \varepsilon, \varepsilon' \in \Gamma^*\)

  then for any \(y \in \Sigma^*\) and \(z \in \Gamma^*\)

  also \((q, xy, \varepsilon) \xrightarrow{\varepsilon} (q', x'y, \varepsilon)\).

- In other words, steps that can take place with a given input and stack contents can also take place with additional input and lower stack contents, without depending upon or using the additional input or contents.

Use of Non-Determinism

- PDA’s are non-deterministic in the general case: Some of the range values of \(\delta\) can have more than one element.

- Unlike the situation with NFA’s, this non-determinism is more of a mathematical artifice.

- It can also be used to understand various parsing algorithms.
Guessing Metaphor

- Think back to NFA’s. One way to describe what they do in a given state with a given input is to “guess” which of several possible next states will ultimately be the “right” one (take the machine to an accepting state).

- This metaphor is extra helpful for PDA’s.

PDA accepting \{xx^R \mid x \in \{a, b\}^*\}

- While it appears similar to \{xcx^R \mid x \in \{a, b\}^*\}, this set is, in some sense, more difficult to recognize.

- The reason is that for a given input, we don’t have an indicator of when x ends and \( x^R \) starts.

- Here a PDA can use its non-determinism: It can guess at the point it thinks x has ended.
  - If it guessed right, it will get to an accepting state.
  - If it guessed wrong, nothing lost.
  - An important thing is that guessing never leads to an accepting state when the input is not in the language.
PDA accepting \( \{xx^R \mid x \in \{a, b\}^*\} \)

- Here’s how a PDA would work for this language:
  - It begins reading the input symbols.
  - For each symbol read, it would push the symbol onto its stack,
  - until it thinks it has reached the end of the \( x \) part of the input,
  - at which time it would transition to a new control state.
  - In the new control state, it would read input symbols, and pop the top
    of the stack, continuing as long as the symbol read matched the
    symbol on the stack.
  - Eventually it can guess has seen all of the input, and go to the
    accepting state, provided the top of the stack is \( Z_0 \).

\[
\begin{align*}
  \emptyset & \text{ for } \{xx^R \mid x \in \{a, b\}^*\} \\

  \begin{array}{c}
  \bullet q_0 \rightarrow \text{(a, } \emptyset \text{) / a} \\
  \text{for each } \emptyset \in \{a, b, Z_0\} \\
  \bullet (B, X) / X \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, X) / X \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (B, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet (a, Y) / Y \text{ for each } (a, b, Y) \\
  \text{for each } (a, b, Y) \\
  \bullet \emptyset \end{array}
\]

Note that the "for each" statements are just a way of abbreviating a larger
number of transitions.
Interconvertibility of Acceptance Modes

- **For every PDA M accepting by final state, there is a PDA M’ accepting the same language by empty stack.** Proof:
  - Create M’ from M as follows: Add a new state $q_E$ and a transition from each accepting state of M to $q_E$.
  - Add transitions from $q_E$ to itself which do nothing but pop symbols from the stack. This ensures that M’ can empty its stack whenever M would have accepted.
  - **However**, we must also ensure that M’ empties its stack only in this case; M could have emptied its stack, so M’ might do the same.

(Proving: For every PDA M accepting by accepting state, there is a PDA M’ accepting the same language by empty stack.)

- Make the initial stack symbol of M’ a new symbol $Z_E$ not used in M.
- Transitions of M can never remove $Z_E$.
- Where M might have emptied its stack, M’ will now have $Z_E$ on the stack.
- By design, M’ cannot remove $Z_E$ unless doing so from $q_E$.  

Accepting State $\emptyset$ Empty Stack

M

$\emptyset, Z'_E/ Z_0 Z_E$  

start state of M'  

$q_E$  

$\emptyset, \emptyset/ \emptyset$, where $\emptyset$ is any stack symbol  

typical accepting state of M

Legend: input, top-of-stack / push-on-stack (top at left)

Interconvertibility of Acceptance Modes (2)

- For every PDA $\mathcal{M}$ accepting by empty stack, there is a PDA $\mathcal{M}'$ accepting the same language by accepting state. Proof:
  - Create $\mathcal{M}'$ from $\mathcal{M}$ as follows: The empty stack symbol $Z'_E$ of $\mathcal{M}'$ is a new stack symbol not used in $\mathcal{M}$.
  - $\mathcal{M}'$ begins from a new initial state $q'_I$, with a $\emptyset$ transition to the initial state of $\mathcal{M}$, with which $\mathcal{M}'$ pushes $Z_0 Z_E$ where $Z_0$ is the initial stack symbol of $\mathcal{M}$.
  - Introduce a new accepting state $q_F$ as the only accepting state of $\mathcal{M}'$.
  - Add new $\emptyset$ transitions from each state of $\mathcal{M}$ to $q_F$ conditioned on being $Z'_E$ on top of the stack.
  - Whenever $\mathcal{M}$ would have emptied its stack, $\mathcal{M}'$ will see $Z'_E$ and can make a transition to the accepting state.
Empty Stack \[ \square \] Accepting State

![](diagram.png)

**Legend:** input, top-of-stack / push-on-stack (top at left)

Leftmost/Rightmost Derivation

**Nomenclature**

- A derivation in a context-free grammar is called **leftmost** if it is always the leftmost auxiliary that is replaced.

- A derivation in a context-free grammar is called **rightmost** if it is always the rightmost auxiliary that is replaced.

- Observation: For each derivation tree there is exactly one leftmost and one rightmost derivation.
Left/Rightmost Derivations

\[
\begin{align*}
A & \rightarrow V \mid V + A \\
V & \rightarrow a \mid b \mid c
\end{align*}
\]

leftmost: \[
A \rightarrow V+A \rightarrow c+A \rightarrow c+V+A \rightarrow c+a+V \rightarrow c+a+b
\]
rightmost: \[
A \rightarrow V+A \rightarrow V+V+A \rightarrow V+V+V \rightarrow V+b \rightarrow V+a+b \rightarrow c+a+b
\]

CFL Characterization Theorem

- A language is context-free iff there is a pushdown acceptor that recognizes it.
Parallel Between CFG and PDA

- Each derivation
  \[ S \xrightarrow{*} x \]
in a CFG
  corresponds to a series of moves
  \[(q, x, S) \xrightarrow{*} (q', \epsilon, \epsilon)\]
of some PDA accepting by empty stack.

CFL \(\subseteq\) PDA Lemma

- Every context free language is accepted by some PDA.
1st Proof of the CFL ⊆ PDA Lemma: 
**top-down** or **produce-match** technique

- Assume L is a context-free language. Then L has a context-free grammar G in Greibach normal form.
- (It is easy to dispose of the Greibach normal form assumption, but we keep it initially for simplicity).
- Construct a PDA M with one control state q.
- This machine will accept by **empty-stack**.
- Each production, which has the form:
  \[ A \rightarrow \alpha B_1 B_2 B_3 \ldots B_n \]
  where \( \alpha \) is terminal and each \( B_i \) is auxiliary, add to M the transition:
  \[ (q, \alpha, A) \rightarrow (q, B_1 B_2 B_3 \ldots B_n) \]
- The initial stack symbol is the start symbol \( S \) of G.
- We claim that for any \( x \in L \):
  \[ (q, x, S) \Rightarrow^* (q, \lambda, \lambda) \iff S \Rightarrow^* x \]
  This requires an inductive proof (given later).

**Example of CFG to Machine**

- **Production** | **Transition**
  - \( S \rightarrow (T) \) | \( q, \langle , S \rangle q, T \)
  - \( S \rightarrow (ST) \) | \( q, \langle , S \rangle q, ST \)
  - \( S \rightarrow (TS) \) | \( q, \langle , S \rangle q, TS \)
  - \( S \rightarrow (STS) \) | \( q, \langle , S \rangle q, STS \)
  - \( T \rightarrow ) \) | \( q, \rangle , T \rangle q, \rangle \)
Example accepting sequence

- Input: ((()))
- Rules are:
  - \( q, '(', S \rightarrow q, T \)
  - \( q, '(', S \rightarrow q, ST \)
  - \( q, '(', S \rightarrow q, TS \)
  - \( q, ')', S \rightarrow q, STS \)
  - \( q, ')', S \rightarrow q, T \)
- State sequence:
  - \( q, ((())), S \rightarrow \text{use rule: } q, '(', S \rightarrow q, ST \text{ to get } q, ()()), ST \)
  - \( q, ()(), ST \rightarrow \text{use rule: } q, ')', T \rightarrow q, \epsilon \text{ to get } q, )() \)
  - \( q, ()(), ST \rightarrow \text{use rule: } q, ')', T \rightarrow q, \epsilon \text{ to get } q, )) \)
  - \( q, T \rightarrow \text{use rule: } q, ')', T \rightarrow q, \epsilon \text{ to get } q, ) \)
  - \( q, \epsilon, \epsilon \rightarrow \text{accept by empty stack} \)

What’s going on here?

- The PDA is simulating a leftmost derivation of a string.
- In a leftmost derivation, the leftmost auxiliary is always rewritten.
  - The symbols to the left of that auxiliary in the derived string are the ones that have been read.
  - The symbols to the right are the stack contents.
Simulating Leftmost Derivation

- $S \rightarrow (T\;') q, \; ', S \rightarrow q, T$
- $S \rightarrow (ST) q, \; ', S \rightarrow q, ST$
- $S \rightarrow (TS) q, \; ', S \rightarrow q, TS$
- $S \rightarrow (STS) q, \; ', S \rightarrow q, STS$
- $T \rightarrow ) q, \; ', T \rightarrow q, )$

- \text{Red shows matched portions of input}

- (()) | $S$
- Not remaining in input and
- (()) | (ST)
- Not actually on the stack.
- (()) | (TST)
- (()) | (ST)
- (()) | (T)
- Underscore shows the LHS symbol
- (()) | (())
- Being rewritten.
- (()) | (())

Exercise

- Show that Greibach normal form is not essential for the lemma and proof technique.

- (Allow terminals as well as auxiliaries on the stack.
  Allow $\;'$ to be either $\;'$ or an element of $\;$.)
PDA ▷ CFL Lemma

- For every PDA M, there is a context-free grammar G such that L(G) = L(M).
- Proof outline:
  - Assume that M has just one control state q and acceptance is by empty stack.
  - Reverse the construction of the CFL ▷ PDA Lemma; for each transition create a corresponding grammar rule:
    - The stack symbols will become auxiliaries in the grammar.
    - The initial stack symbol is S, the start symbol.
  - For each transition rule of the form \( q, s, Z \rightarrow q, g \) intro,
    - introduce a production \( Z \rightarrow s g \)
- We claim that for any \( x \in S^* \):
  - \( (q, x, S) \in L \) iff \( S \triangleright x \)
  - This requires an inductive proof (given later).

Proof that One State Suffices

- Assume WLOG that M = \( (Q, \square, \square, q_0, Z_0, \{q_F\}, \square) \) is a pda accepting by empty stack, where \( q_F \) is a single final state, from which M empties its stack.
- Construct a new pda \( M' = (\{\square\}, \square, q, Z_0', A', \square) \) that accepts the same language:
  - \( A' = Q \times \square \times Q \)
  - Each element of \( A' \) is written \( \langle q, Z, q' \rangle \) but is really just a single symbol.
  - The initial stack symbol of \( M' \) is \( Z_0' = <q_0, Z_0, q_F> \).
  - For each transition of M: \( (q', B_1 B_2 B_3 ... B_n, \square, \square, \square, q, Z) \) include as transitions of \( M' \):
    - \( \langle q, q_0, B_1, q_1, q_2 ... q_{n-1}, B_n, q_{n+1}, \rangle \rightarrow \langle q, q_0, Z, q_{n+1}, \rangle \)
  - For every choice of \( q_1, q_2, ..., q_n \in Q \) (choices not necessarily distinct).
  - That \( M' \) accepts the same language as M needs to be proved by induction.
2nd Proof of the CFL PDA Lemma: bottom-up or shift-reduce technique

- Assume L is a context-free language. Then L has a context-free grammar G.
- Create a pda that accepts by final state.
- For each terminal symbol , create a transition:
  \[ q_0, \, , \, q_0, \]
  These have the effect of shifting the input string onto the stack (and reversing it in the process).
- For each production \( A \rightarrow x_1x_2x_3...x_n \) create transitions:
  \[ q_0, \, L, \, x_n \rightarrow q_1, \, L \]
  \[ q_1, \, L, \, x_{n-1} \rightarrow q_2, \, L \]
  \[ q_2, \, L, \, x_{n-2} \rightarrow q_3, \, L \]
  \[ \ldots \]
  \[ q_n, \, L, \, x_1 \rightarrow q_0, \, A \]
  where the \( q_i \) other than are distinct for each production.
- This pda simulates a rightmost derivation of an accepted input string.

Example of Bottom-Up Production Transitions

<table>
<thead>
<tr>
<th>Production</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow (T) )</td>
<td>( q_0, , L, , , , q_1, , L ) ( S \rightarrow , L, , q_0, , Z_0 )</td>
</tr>
<tr>
<td>( T \rightarrow S )</td>
<td>( q_0, , L, , , , q_2, , L ) ( q_2, , L, , S \rightarrow q_0, , T )</td>
</tr>
<tr>
<td>( S \rightarrow () )</td>
<td>( q_0, , L, , , , q_3, , L ) ( q_3, , L, , ( \rightarrow q_0, , S )</td>
</tr>
<tr>
<td>( S \rightarrow SS )</td>
<td>( q_0, , L, , S \rightarrow q_4, , L ) ( q_4, , L, , S \rightarrow q_0, , S )</td>
</tr>
<tr>
<td>shift transitions for each</td>
<td>( q_0, , , , q_0, , ( )</td>
</tr>
<tr>
<td>accept transitions</td>
<td>( q_0, , Z_0 \rightarrow q_5, , Z_0 )</td>
</tr>
</tbody>
</table>

State Sequence

\( ((()) | q_0 | Z_0 \)
\( () | q_0 | (Z_0 \)
\( () | q_0 | ((Z_0 \)
\( () | q_0 | ))(Z_0 \)
\( () | q_0 | )S(Z_0 \)
\( () | q_0 | SS(Z_0 \)
\( () | q_0 | )SS(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | )S(Z_0 \)
\( () | q_0 | )S(Z_0 \)
\( () | q_0 | )S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)
\( () | q_0 | S(Z_0 \)

Derivation: \( S \rightarrow (T \rightarrow (S) \rightarrow (SS) \rightarrow (S()) \rightarrow ())) \)