Pushdown Automata

Robert M. Keller
Harvey Mudd College
1 October 2003

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 | x □ □*} where c □ □.
- Supposing □ ={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 | x □ □*} where c □ □.
  - 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**.

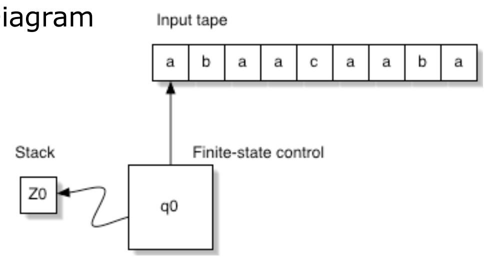
PDA Defined: (Q, □, □, q_0, Z_0, A, □)

- Q is a finite set of **control states**
- □ is the **input** alphabet
- □ is the **stack** alphabet
- q_0 is the **initial** control state
- Z_0 is the **initial** stack symbol
- A □ Q is the set of **accepting** control states
- □ is the **state transition relation** (or, in the case of a DPDA, **function**)

  □: Q x (□ □ {□}) x □ □ finite subsets of Q x □*

Say What?

PDA Diagram



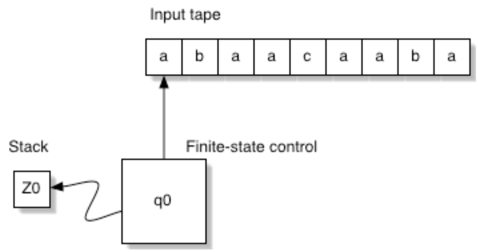
The **state** is generally of the form (q, □) □ Q x □*,
i.e. (control-state, stack-contents).
Sometimes this is called a **configuration** or **instantaneous description (ID)**, to distinguish it from the control state. I prefer to call it the state, which is what it is.

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

PDA Acceptance

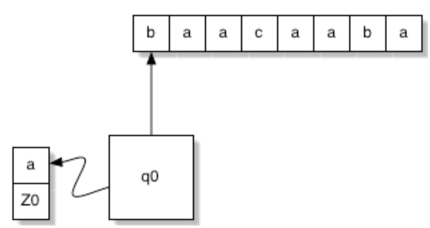
- There are different models for acceptance.
- **Final-state acceptance** means that the PDA accepts when it is in a **designated control state** after the input has been read.
- **Empty-stack acceptance** means that the PDA accepts when its **stack is empty** after the input has been read.
- The choice is a matter of convenience; the two can be shown **inter-convertible**.

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

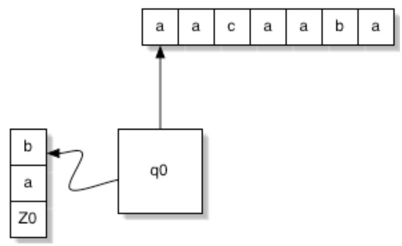


Note that control-state is in q_0 and will stay there until c is read.

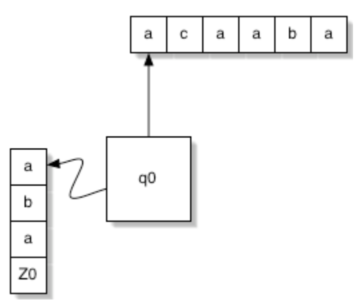
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



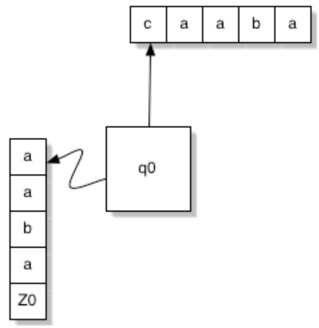
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



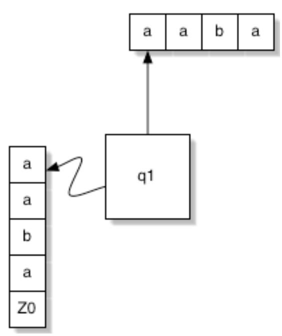
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



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

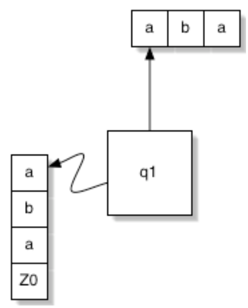


Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$

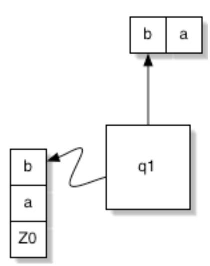


Note that control-state has changed to q_1 after reading c.

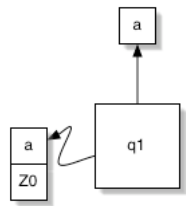
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



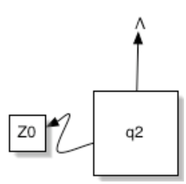
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



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



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



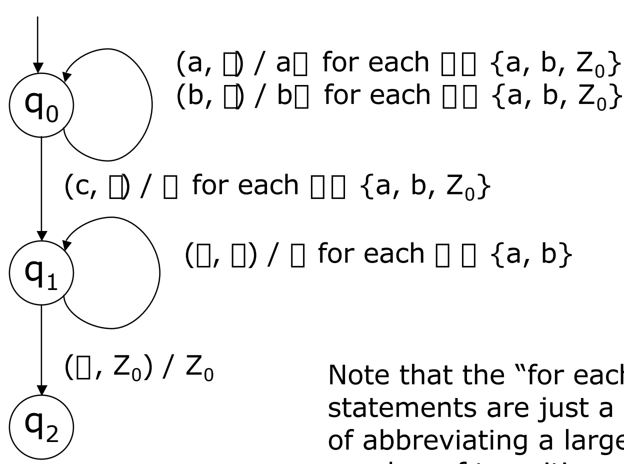
q_2 is the accepting control state.
The input is now empty.

So the PDA **accepts** the original input: abaacaaba.

Design of the PDA for the xcx^R language

- □: Q x (□ □ {□}) x □ □ finite subsets of Q x □*
  - □(q_0, □, □) = {(q_0, □ □)}, for each □ □ {a, b}, □□ {a, b, Z_0}
  - □(q_0, c, □) = {(q_1, □)}, for each □□ {a, b, Z_0}
  - □(q_1, □, □) = {(q_1, □)}, for each □ □ {a, b}
  - □(q_1, □, Z_0) = {(q_2, 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.

□ can also be described as a graph



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, □□ q_0, a□ for each □□ {a, b, Z_0}
q_0, b, □□ q_0, b□ for each □□ {a, b, Z_0}
q_0, c, □□ q_1, □ for each □□ {a, b, Z_0}
q_1, □, □□□ q_1, □ for each □□ {a, b, Z_0}, for each □ □ {a, b}
q_1, □, Z_0 □ q_2, Z_0

The rules with “for each” are actually abbreviations for other rules with the □ and □ variables by literal symbols.

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

The |□ (turnstyle) and |□ * notation.

- Let q, q’ □ Q; x, x’ □ □*; □, □’ □ □*

  (q, x, □) |□ (q’, x’, □’)

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

- |□ * is the **transitive closure** of |□ , meaning:
  - (q, x, □) |□ * (q, x, □)
  - If (q, x, □) |□ * (q’, x’, □’) and (q’, x’, □’) |□ * (q”, x”, □”),

    then (q, x, □) |□ * (q”, x”, □”).

The key property ("stack property") of PDA's:

- If (q, x, □) |□ * (q', x', □)

  where x, x' □ □*; □, □ □ □*

  then for any y □ □* and □ □ □*

  also (q, xy, □□) |□ * (q', x'y, □□).

- 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 □ 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.

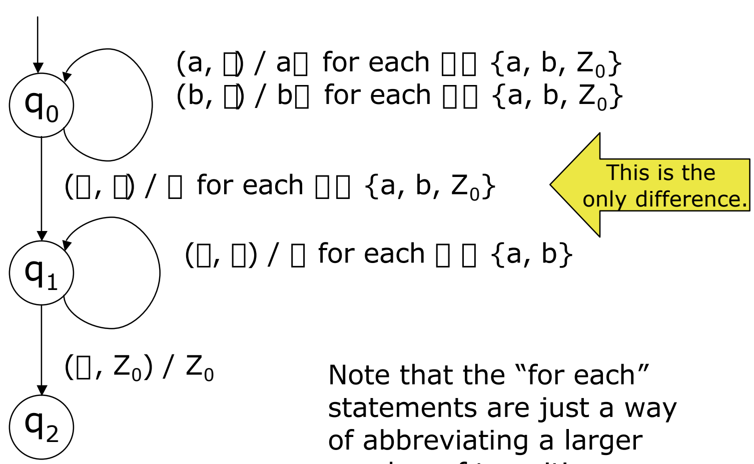
- The reason is that for a given input, we don't have an indicator of when x ends and x^R starts.

- 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 $\{xx^R \mid x$ □ $\{a, b\}^*\}$

- 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.

□ for $\{xx^R \mid x$ □ $\{a, b\}^*\}$



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

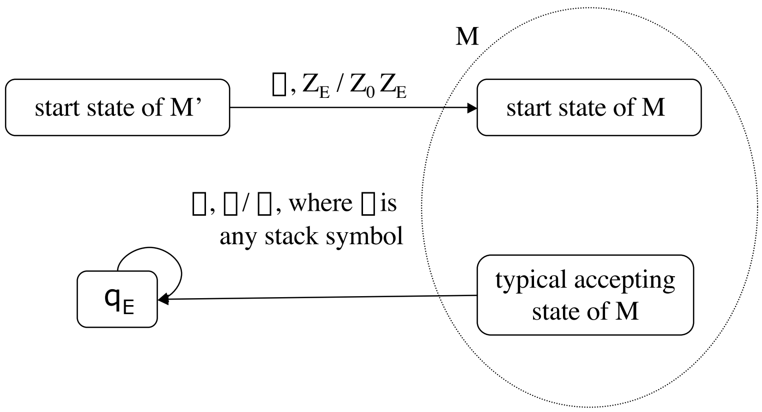
Interconvertibility of Acceptance Modes

- **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 □ Empty Stack

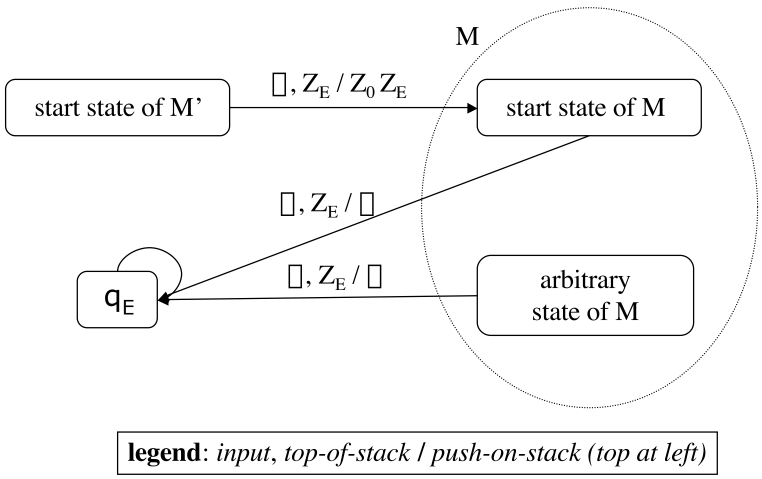


legend: *input*, *top-of-stack* / *push-on-stack (top at left)*

Interconvertibility of Acceptance Modes (2)

- **For every PDA M accepting by empty stack, there is a PDA M' accepting the same language by accepting state.** Proof:
- Create M' from M as follows: The empty stack symbol Z_E of M' is a new stack symbol not used in M.
- M' begins from a new initial state q_I, with a □ transition to the initial state of M, with which M' pushes Z_0 Z_E where Z_0 is the initial stack symbol of M.
- Introduce a new accepting state q_F as the only accepting state of M'.
- Add new □ transitions from each state of M to q_F conditioned on being Z_E on top of the stack.
- Whenever M would have emptied its stack, M' will see Z_E and can make a transition to the accepting state.

Empty Stack □ Accepting State



Leftmost/Rightmost Derivation Nomenclature

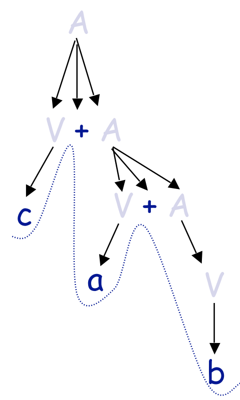
- A derivation in a context-free grammar is called leftmost if it is always the leftmost auxiliary that is replaced.

- 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

A → V | V + A
V → a | b | c



leftmost: $\underline{A} \Rightarrow \underline{V}+A \Rightarrow c+\underline{A} \Rightarrow c+\underline{V}+A \Rightarrow c+a+\underline{A} \Rightarrow c+a+\underline{V} \Rightarrow c+a+b$

rightmost: $\underline{A} \Rightarrow V+\underline{A} \Rightarrow V+V+\underline{A} \Rightarrow V+V+\underline{V} \Rightarrow V+\underline{V}+b \Rightarrow \underline{V}+a+b \Rightarrow c+a+b$

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 □ * x
  in a CFG
  corresponds to a series of moves
  (q, x, S) |□ * (q′, □, □)
  of some PDA accepting by empty stack.

CFL □ PDA Lemma

- Every context free language is accepted by some PDA.

1st Proof of the CFL $\rightarrow$ 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:
  $A \rightarrow \sigma B_1 B_2 B_3 \ldots B_n$,
  where is σ terminal and each B_i is auxiliary, add to M the transition:
  $(q, \sigma, A) \rightarrow (q, B_1 B_2 B_3 \ldots B_n)$
- The initial stack symbol is the start symbol S of G.
- We claim that for any $x \in \Sigma^*$
  $(q, x, S) \vdash^* (q, \varepsilon, \varepsilon)$ iff $S \Rightarrow^* x$
  This requires an inductive proof (given later).

Example of CFG to Machine

Production	Transition
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, ε

Example accepting sequence

- Input: (()())
- Rules are:
  q, `(`, S □ q, T
  q, `(`, S □ q, ST
  q, `(`, S □ q, TS
- q, `(`, S □ q, STS
  q, `)`, T □ q, □
- State sequence:
  - q, (()()), S use rule: q, `(`, S □ q, ST to get
  - q, ()()), ST use rule: q, `(`, S □ q, TS to get
  - q,)()), TST use rule: q, `)`, T □ q, □ to get
  - q, ()), ST use rule: q, `(`, S □ q, T to get
  - q,)), TT use rule: q, `)`, T □ q, □ to get
  - q,), T use rule: q, `)`, T □ q, □ to get
  - q, □, □ 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 □ (T q, '(', S □ q, T
  S □ (ST q, '(', S □ q, ST
  S □ (TS q, '(', S □ q, TS
  S □ (STS q, '(', S □ q, STS
  T □) q, ')', T □ q, □
- (()()) | $\underline{\text{S}}$
- (()()) | ($\underline{\text{S}}$T
- (()()) | (($\underline{\text{T}}$ST
- (()()) | (()$\underline{\text{S}}$T
- (()()) | (()($\underline{\text{T}}$T
- (()()) | (()())$\underline{\text{T}}$
- (()()) | (()())

Red shows matched portions of input not remaining in input and not actually on the stack.

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, □, Z □ q, □ (□ □ □ □ {□})
      introduce a production
      Z □ □ □
- We claim that for any x □ □*
  (q, x, S) |□ * (q, □, □) iff S □ * x
  This requires an inductive proof (given later).

Proof that One State Suffices

- Assume WLOG that M = (Q, □, □, q_0, Z_0, {q_F}, □) 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′ = ({□}, □, □′, q, Z'_0, A′, □′)
  that accepts the same language:
  - □′ = Q x □ x Q
  - Each element of □′ is written <q Z q′> 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′, $B_1B_2B_3 \ldots B_n$) □ □(q, □, Z) include as transitions of M′:
    (□, <q_0 B_1 q_1> <q_1 B_2 q_2> … <q_{n-1} B_n q_n>) □ □′(□, □, <q_0 Z q_n>)
    for every choice of q_1, q_2 , …, q_n □ Q (choices not necessarily distinct).
    [If n = 0, then (□, <q_0 B_1 q_n>) □ □′(□, □, <q_0 Z q_n>).
  - 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 □ $x_1x_2x_3 \ldots x_n$ create transitions:

  q_0, □, x_n □ q_1, □
  q_1, □, x_{n-1} □ q_2, □
  q_2, □, x_{n-2} □ q_3, □
  . . .
  q_n, □, x_1 □ q_0, A

  where the q_i other q_0 than are distinct for each production.
- This pda simulates a **rightmost** derivation of an accepted input string.

Example of Bottom-Up

Production	Transitions
S □ (T	q_0, □, T □ q_1, □ q_1, □, (□ q_0, S
T □ S)	q_0, □,) □ q_2, □ q_2, □, S □ q_0, T
S □ ()	q_0, □,) □ q_3, □ q_3, □, (□ q_0, S
S □ SS	q_0, □, S □ q_4, □ q_4, □, S □ q_0, S
shift transitions for each □	q_0, (, □ □ q_0, (□ q_0,), □ □ q_0,)□
accept transitions	q_0, □, S □ q_5, □ q_5, □, Z_0 □ q_a, Z_0

Derivation: S □ (T □ (S) □ (SS) □ (S()) □ (()())

State Sequence
(()()) | q_0 | Z_0
(()()) | q_0 | (Z_0
(()()) | q_0 | ((Z_0
(()()) | q_0 |)((Z_0
(()()) | q_3 | ((Z_0
(()()) | q_0 | S(Z_0
(()()) | q_0 | (S(Z_0
(()()) | q_0 |)(S(Z_0
(()()) | q_3 | (S(Z_0
(()()) | q_0 | SS(Z_0
(()()) | q_4 | S(Z_0
(()()) | q_0 | S(Z_0
(()()) | q_0 |)S(Z_0
(()()) | q_2 | S(Z_0
(()()) | q_0 | T(Z_0
(()()) | q_1 | (Z_0
(()()) | q_0 | SZ_0
(()()) | q_5 | Z_0
(()()) | q_a | Z_0