Chapter 2: Context-Free Languages

Cardinality

- Def: A set is countable if it is

  (i) finite or

  (ii) countably infinite.

- Def: A set S is countable infinite if there is a bijection from N to S.
  - Note: Instead of N to S we can also say S to N.

    N = {0, 1, 2, 3, ...} -- the set of natural numbers.

- Def: A set that is not countable is uncountable.
  - Or, any infinite set with no bijection from N to itself.

Are all languages regular?

- Ans: No.
- How do we know this?
  - Ans: Cardinality arguments.
- Let C(DFA) = {M | M is a DFA}.
  - C(DFA) is a countable set. Why?
- Let AL = { L | L is a subset of Σ^*}.
  - AL is uncountable.

Examples

1. The set of chairs in this classroom is finite.
2. The set of even numbers is countably infinite.
3. The set of all subsets of N is uncountable.
   – Notation: 2^N

Exercises

- What is the cardinality of?
  1. Z - the set of integers.
  2. N X N = {(a,b) | a, b in N}.
  3. R - the set of real numbers.

Pumping Lemma

- First technique to show that specific given languages are not regular.
- Cardinality arguments show existence of languages that are not regular.
- There is a big difference between the two!

Statement of Pumping Lemma

If A is a regular language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

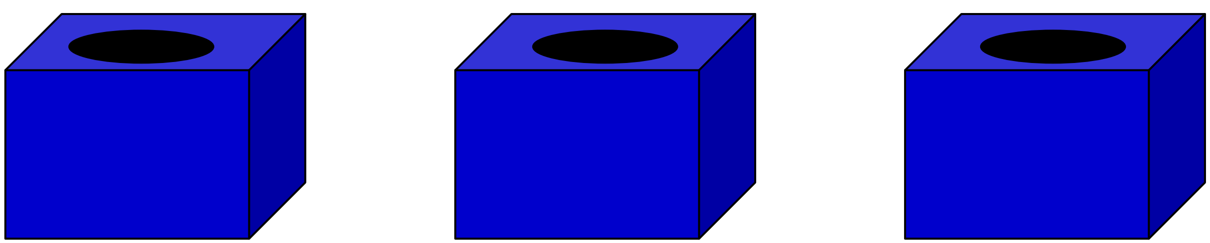
Proof of pumping lemma

- Idea: If a string w of length m is accepted by a DFA with n states, and $n < m$, then there is a cycle (repeated state) on the directed path from s to a final state labeled w.
  - Recall: directed path is denoted by $\delta^*(s,w)$.
  - Uses: Pigeon-hole principle.

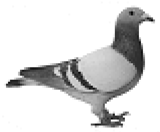
Pigeon-hole principle

4 pigeons

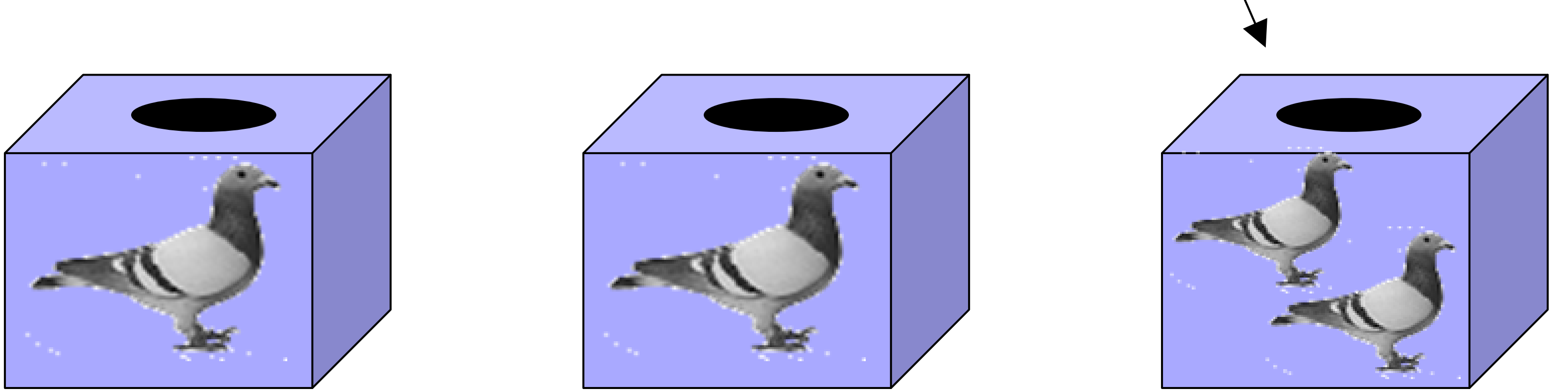
3 pigeonholes



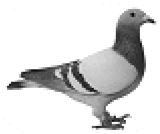
Pigeon-hole principle (cont'd)



A pigeonhole must
have two pigeons

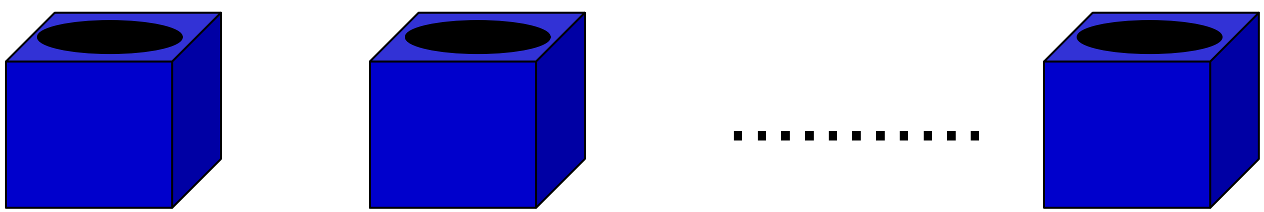


Pigeon-hole principle (contd.)



n $n > m$

m pigeonholes



Details of Proof of Pumping Lemma

Consider L - any infinite regular language.

1. L is regular $\rightarrow$ there is a DFA M with $L(M) = L$.
2. Let DFA have p states (say).
3. Let w in L be of length more than p.
   - Why does w exist?
   - Ans: because L is infinite.
4. $\delta^*(s,w) = f$ (some final state) must be

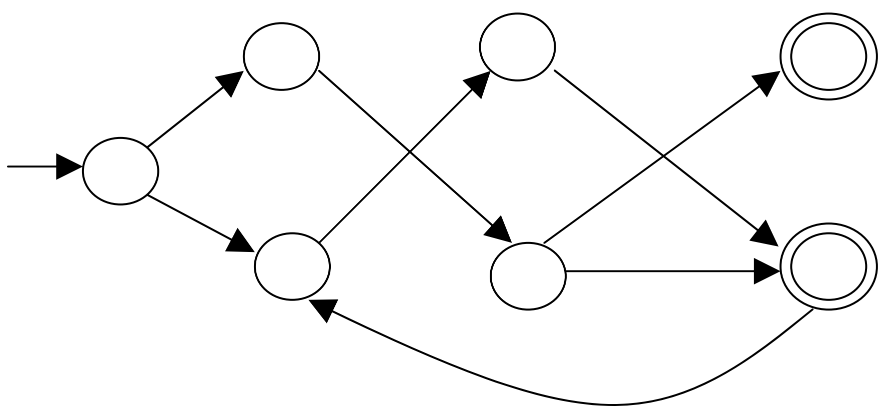
   $\delta^*(s, w = xyz) = \delta^*(q,yz) = \delta^*(q,z) = f$
5. So xy^nz in L for $n = 0, 1, 2, 3, \ldots$.

since $\delta^*(s, xy^nz) = \delta^*(q, y^nz) = \delta^*(q, y^{\{n-1\}}z) = \ldots = \delta^*(q,z) = f$.

Describing the pumping lemma

Take an infinite regular language L

DFA that accepts L



m

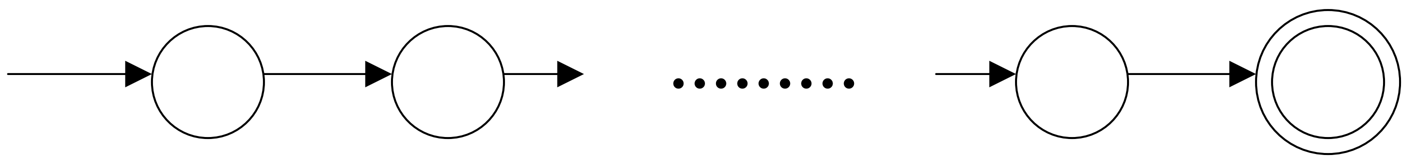
states

Describing the pumping lemma (contd.)

Take string w

with $w \in L$

There is a walk with label: w

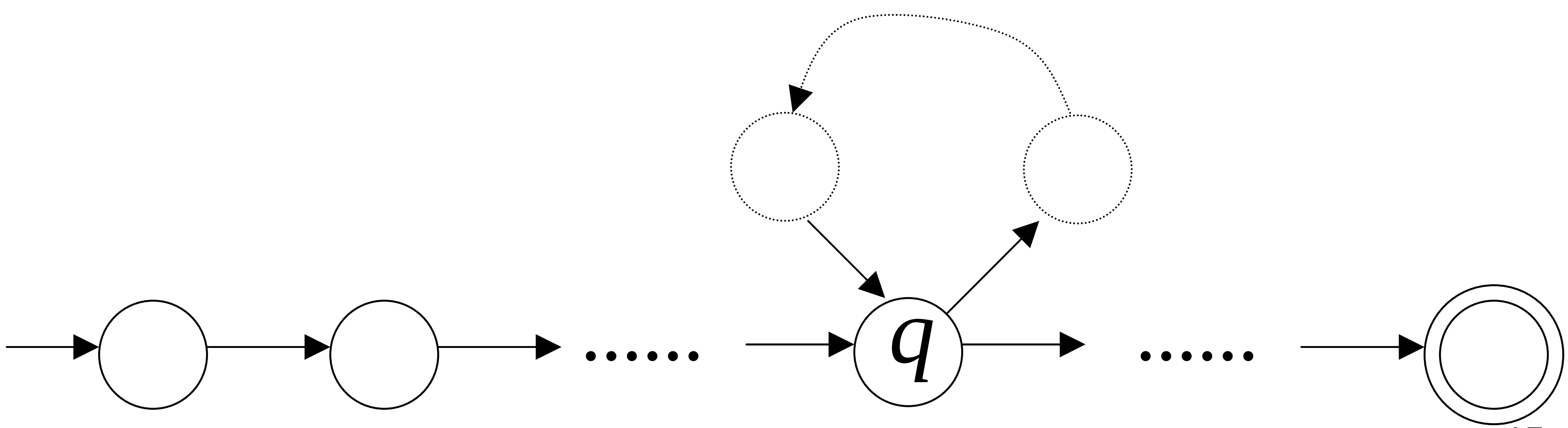


Describing the pumping lemma (contd.)

If string w has length $|w| \geq m$ number of states,

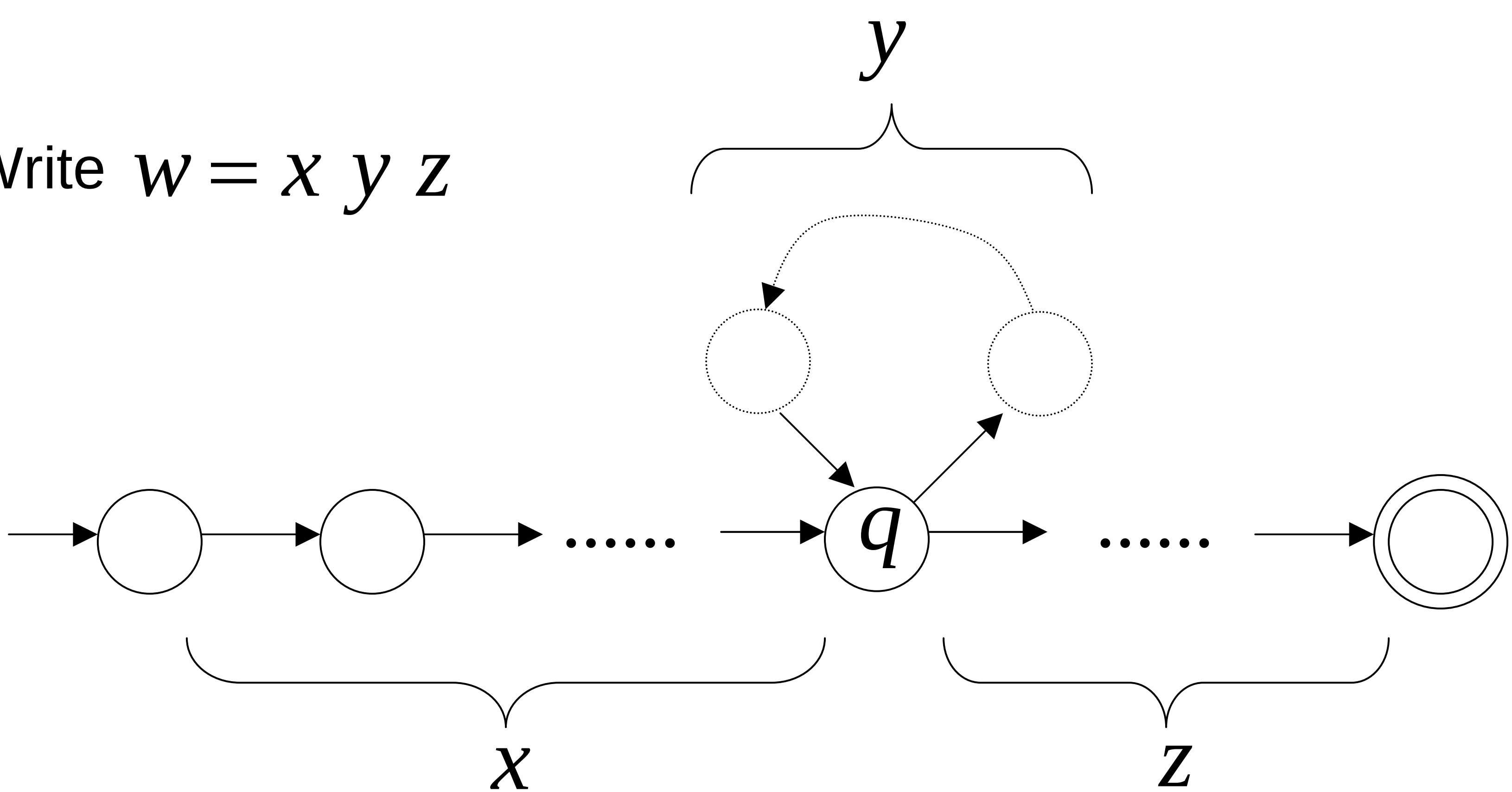
Then, from the pigeonhole principle:

A state q is repeated in the walk w



Describing the pumping lemma (contd.)

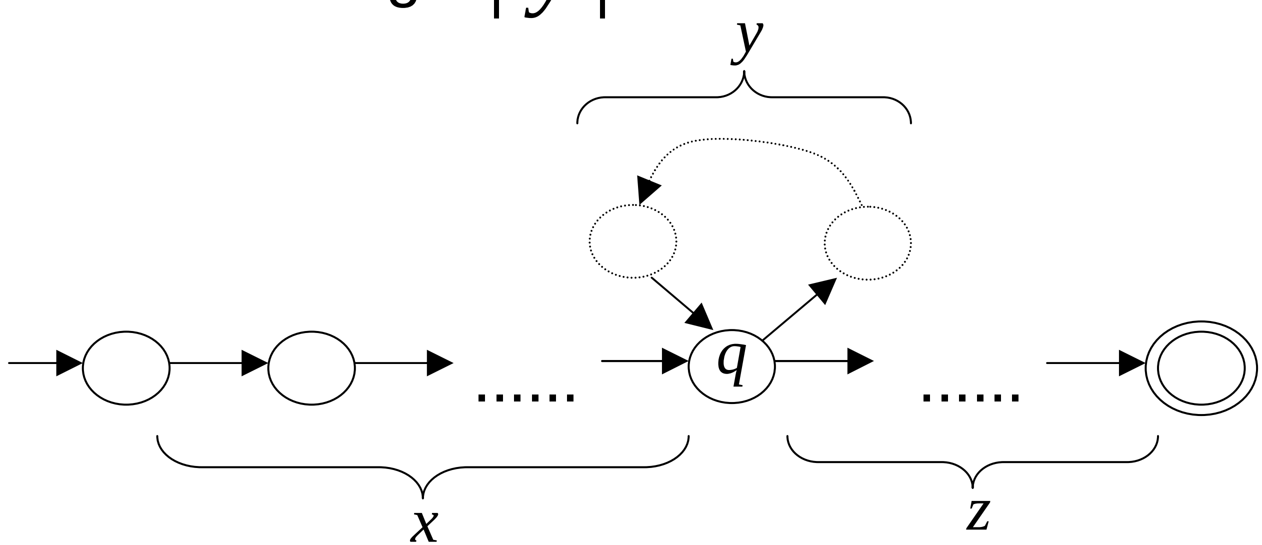
Write $w = x\ y\ z$



Describing the pumping lemma (cont'd)

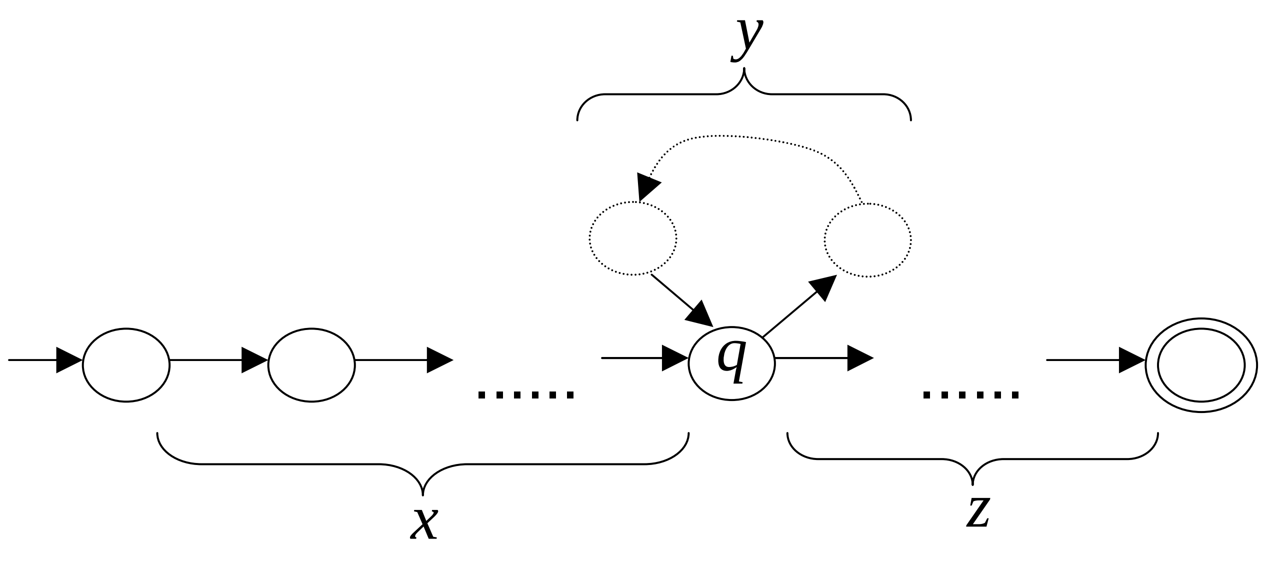
Observations : length $|x\ y| \leq m$ number of states

length $|y| \geq 1$



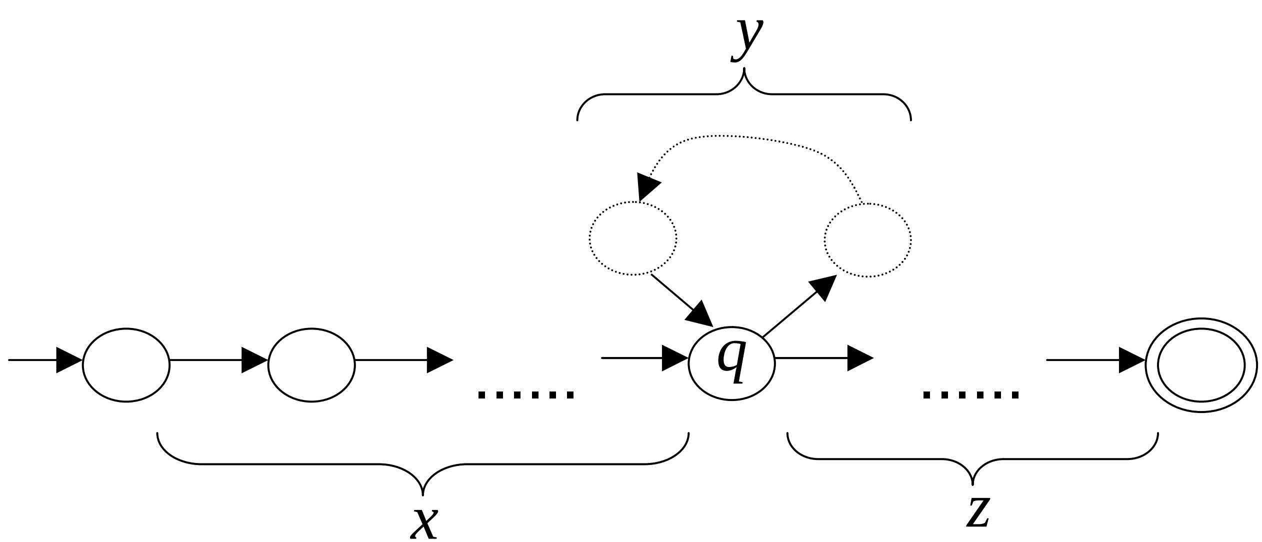
Describing the pumping lemma (contd.)

Observation: The string $x\ z$ is accepted



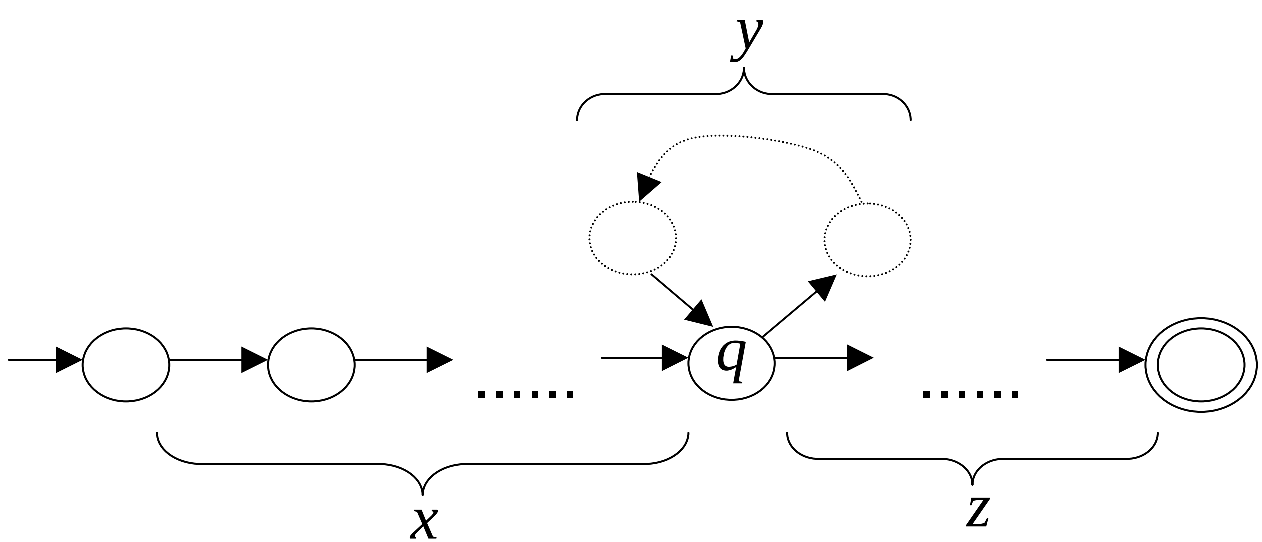
Describing the pumping lemma (contd.)

Observation: The string $x\ y\ y\ z$ is accepted.



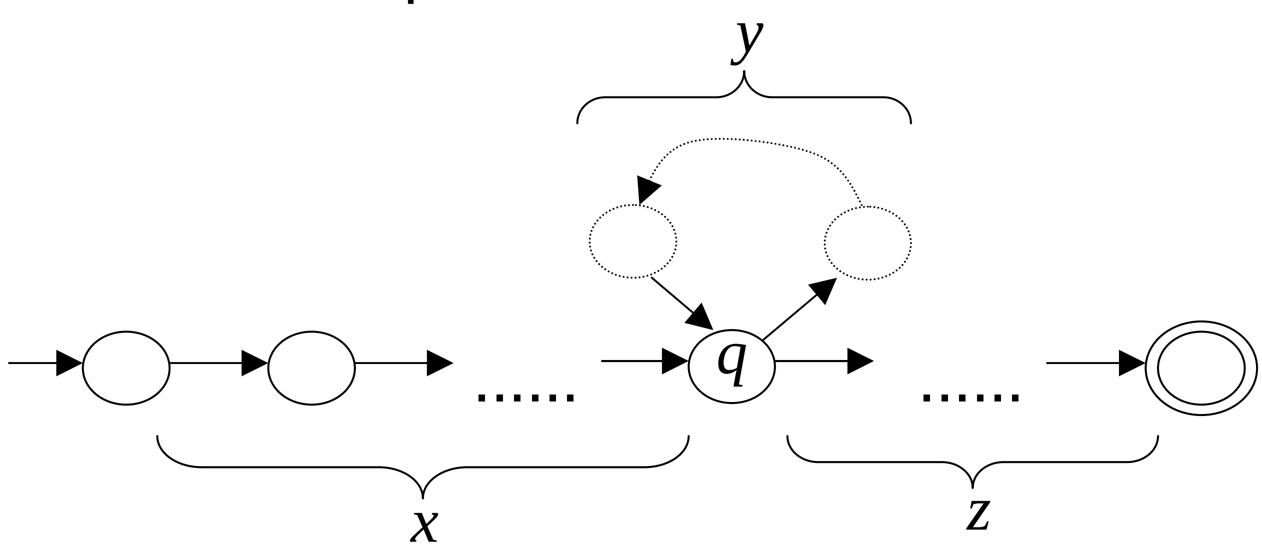
Describing the pumping lemma (contd.)

Observation: The string $x\ y\ y\ y\ z$ is accepted .



Describing the pumping lemma (cont'd.)

In General: The string $x\ y^i\ z \qquad i = 0, 1, 2, \ldots$
is accepted



Some Applications of Pumping Lemma

The following languages are not regular.

1. $\{a^n b^n \mid n \geq 0\}$.
2. $\{w = w^R \mid w \text{ in } \{a,b\}^*\}$ (language of palindromes).
3. $\{ww^R \mid w \text{ in } \{a,b\}^*\}$.
4. $\{a^{n^2} \mid n \geq 0\}$.

Tips of the trade -- Do not forget!

Closure properties can be used effectively for:

(1) shortening cumbersome Pumping lemma arguments.

- Example: {w in {a, b}* | w has equal a's and b's}.

(2) showing that certain languages are regular.

- Example: {w in {a, b}* | w begins with *a* and w contains a *b*}.

Pumping lemma applications

- Proving $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

Proof:

Assume L is regular. Certainly L is infinite and therefore the pumping lemma applies to L.

Let p be the constant for L (of the pumping lemma).

Pumping lemma applications (cont'd.)

To show there exist a string $w \in L$ of length at least p such that $\neg Q$ where Q is the rest of the statement of pumping lemma.

Let $w = a^p b^p$ such that $|w| \geq p$

write

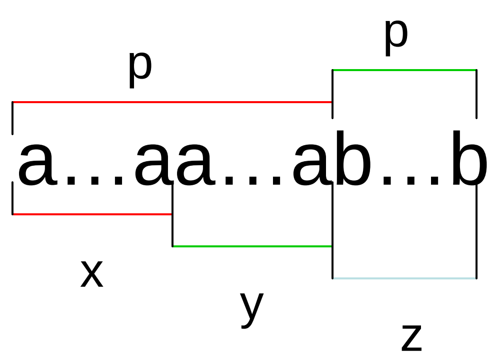
$$a^p b^p = xyz$$

But according to pumping lemma,

Pumping lemma (PL) applications (cont'd.)

PL statement (i) $\rightarrow$ $|xy| \leq p$

Therefore,



$x = a^k$, $y = a^m$ $m > 0$, $z = a^{p-k-m}b^p$

$xyz = a^p b^p$

Pumping lemma applications (cont'd.)

PL statement (ii) $\rightarrow xy^iz \in L \quad i = 0,1,2,3,\ldots$

Therefore,

$$xy^2z \in L$$

$$xy^2z = xyyz = a^k a^{2m} a^{p-k-m} b^p$$

$$= a^{p+m} b^p \in L$$

But,

$$L = \{a^n b^n \mid n \geq 0\}$$

which means $a^{p+m}b^p \notin L$ since $m > 0$.

CONTRADICTION !!

Pumping lemma applications (cont'd.)

Therefore our assumption that

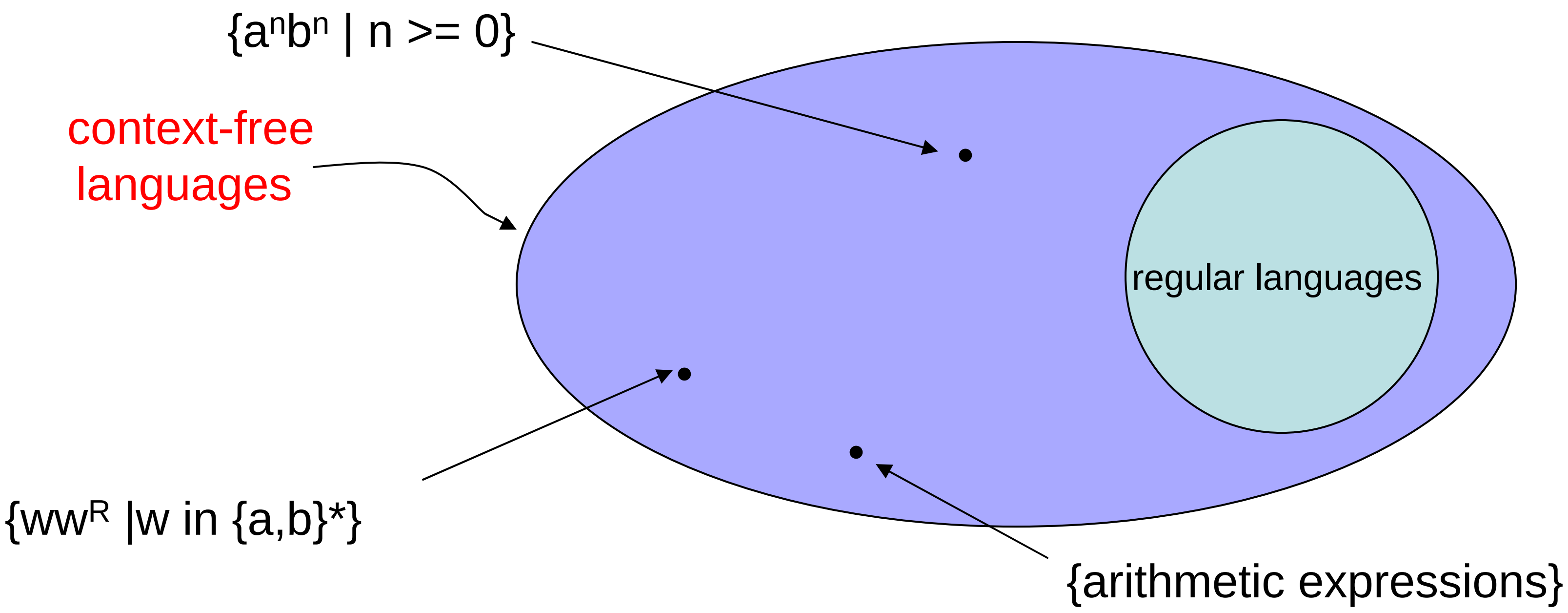
$L = \{a^n b^n \mid n \geq 0\}$ is a regular language cannot be true.

Using Pumping Lemma -- Very Important points

- Above example is a typical application of pumping lemma, to show that a language is not regular.
- You must choose string w so that w in L and $|w|$ is at least the pumping length.
  - Example: choosing w = aaabbb is wrong since we do not know the exact value of p.
- You must consider all possibilities for x, y and z such that $w = xyz$ and $|xy| \leq p$.
- The pumping lemma CANNOT be used to show that a language is regular, since it assumes that L is regular.

Context Free Languages

- Strictly bigger class than regular languages



A simple grammar for some sentences

<Sentence> $\rightarrow$ <noun> <verb> <object>

<noun> $\rightarrow$ Alice | John

<verb> $\rightarrow$ eats | eat | ate

<object> $\rightarrow$ apple | orange | mango

- The goal is to generate sentences in English over the English alphabet. Example:

<Sentence> $\Rightarrow$ <noun><verb><object> $\Rightarrow$ Alice <verb><object>
$\Rightarrow$ Alice eats<object> $\Rightarrow$ Alice eats apple

Context-free grammars (CFGs)

- Informally, a CFG is a finite set of rules.
- Each rule is of the form:

  <nonterminal symbol> $\rightarrow$ string over terminals and nonterminals.

- **Terminals**:- symbols that the desired strings should contain.
  - Example: {a...z, ' ',...}
- **Nonterminals**:- symbols to which rules can be applied.
  - Example: {<noun>, <verb>, ...}
- A special nonterminal is called **start** symbol.
  - Example: <Sentence>

A grammar for arithmetic expressions

- $E \rightarrow E + E \mid E * E \mid (E) \mid x \mid y$
- Start symbol - E.
- Terminals?
  - Ans: {x, y, *, +, (,)}
- Nonterminals?
  - Ans: {E}
- A derivation:

$$E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow x + E * E \Rightarrow x + x * E$$
$$\Rightarrow x + x * y$$

Another grammar for arithmetic expr's

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E) \mid x \mid y$

A derivation for $x + x * y$?

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow x + T \Rightarrow x + T * F$$
$$\Rightarrow x + F * F \Rightarrow x + x * F \Rightarrow x + x * y$$

Why two different grammars for arithmetic expressions?

Context Free Grammar Definition

- A CFG G = (V, T, P, S) where $V \cap T = \varnothing$,
  - V -- A finite set of symbols called nonterminals
  - T -- A finite set of symbols called terminals.
  - P is a finite subset of V X $(V \cup T)^*$ called productions or rules.
  - We write $A \rightarrow w$ whenever $(A, w) \in P$.
  - $S \in V$ -- start symbol.

Derivations and L(G)

- One step derivation:
  - $u \Rightarrow v$ if $u = xAy$, $v = xwy$ and $A \rightarrow w$ in P
- 0 or more steps derivation:
  - $u \Rightarrow^* v$ if $u = u_0 \Rightarrow u_1 \Rightarrow .... \Rightarrow u_n = v$ $(n \geq 0)$
- $L(G) = \{ w \text{ in } T^* \mid S \Rightarrow^* w \}$.
- A language L is context-free if there is a CFG G with L(G) = L.

Example:

$S \to aSb \mid \varepsilon$

Derivation:

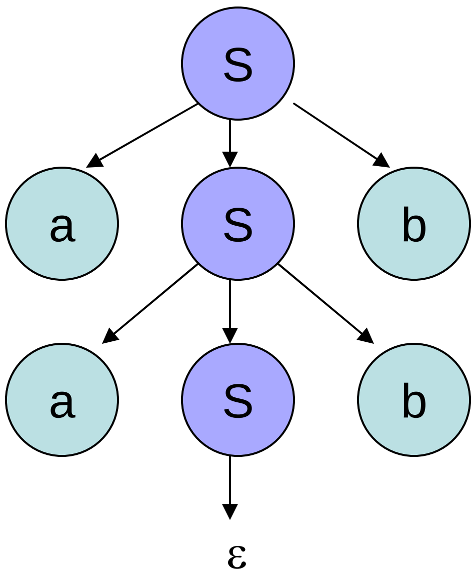
$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$.

L(G) = ?

- Ans: $\{a^n b^n \mid n \geq 0\}$

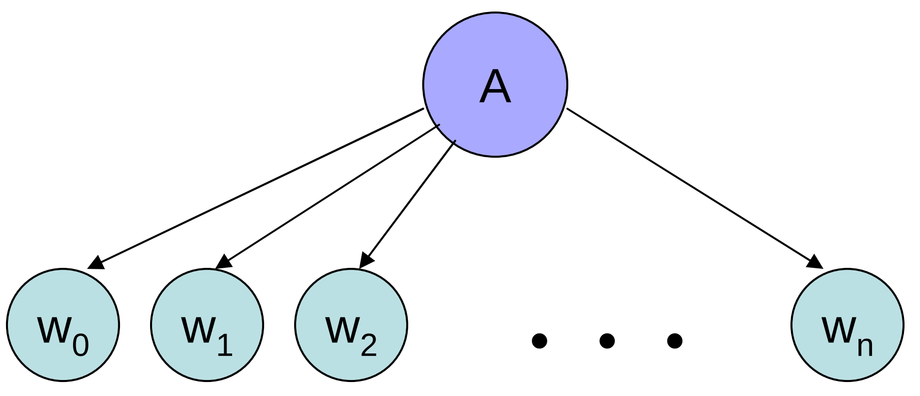
Parse trees

- All derivations can be shown in the form of trees.
- Order of rule application is lost.

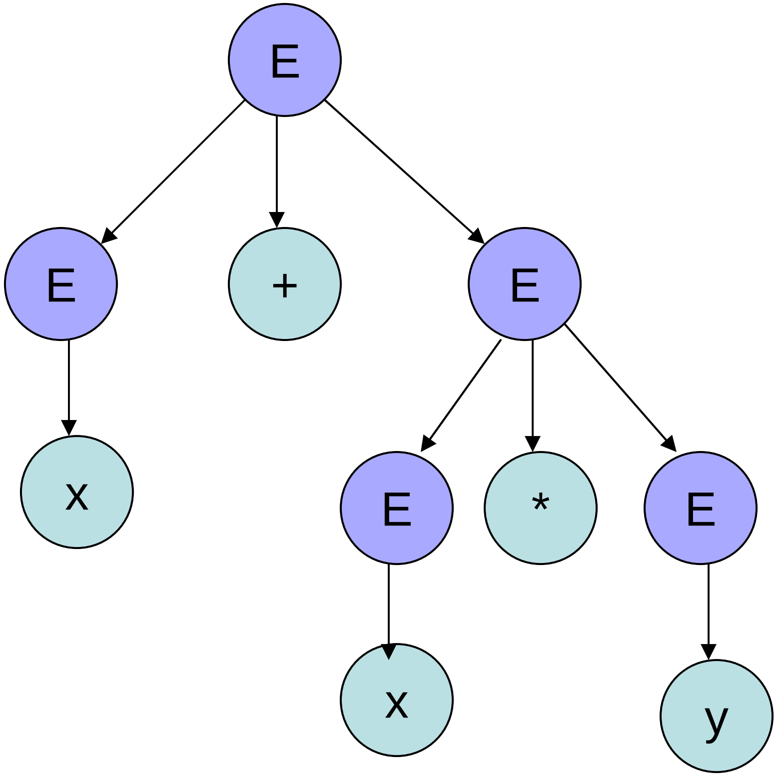


Parse trees [cont'd.]

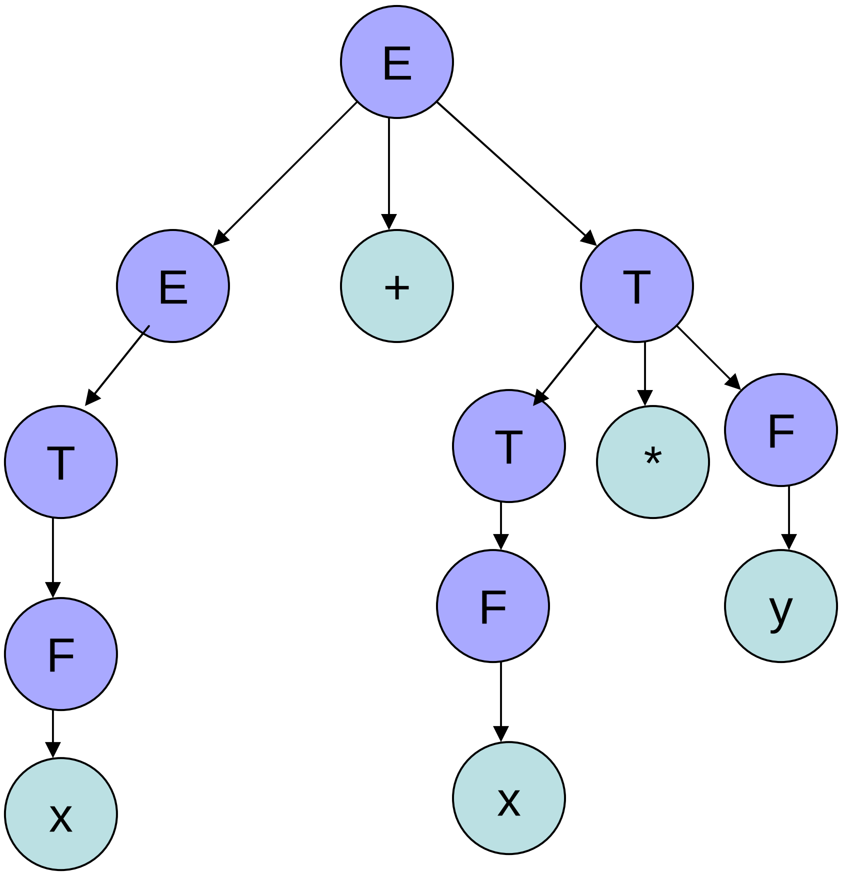
In general, if we apply rule $A \rightarrow w_0 w_1 \ldots w_n$, then we add nodes for w_i as children of node labeled A.



$$E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow x + E * E \Rightarrow x + x * E \Rightarrow x + x * y$$



$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow x + T$$
$$\Rightarrow x + T * F \Rightarrow x + F * F \Rightarrow x + x * F$$
$$\Rightarrow x + x * y$$



Leftmost and Rightmost Derivations

- Derivation is leftmost if the nonterminal replaced in every step is the leftmost nonterminal.
- Consider $E \Rightarrow E + E \Rightarrow E + x$.
  - Is it leftmost derivation?
- Derivation is rightmost if the nonterminal replaced in every step is the rightmost nonterminal.
- Consider $E \Rightarrow E + E \Rightarrow x + E$.
  - Is it rightmost derivation?

Ambiguity

- A CFG is ambiguous if there is a string with at least two leftmost derivations.
  - Example:

    $E \rightarrow E + E \mid E * E \mid (E) \mid x \mid y$ is ambiguous.
- A CFL is inherently ambiguous if every CFG that generates it is ambiguous.
  - Example:

    $\{a^n b^n c^m \mid n, m \geq 0\} \cup \{a^m b^n c^n \mid n, m \geq 0\}$

Chomsky Normal Form (CNF)

- Rules of CFG G are in one of two forms:

  (i) $A \to a$

  (ii) $A \to BC$, $B \neq S$ and $C \neq S$ (S is the start symbol)

  + Only one rule of the form $S \to \varepsilon$ is allowed if ε in L(G).
- Easier to reason with proofs.
- Leads to more efficient algorithms.
- Credited to Prof. Noam Chomsky at MIT.

Reading Assignment: Converting a CFG to CNF.

Exercises

- Are the following CFG's in CNF?

(i) $S \rightarrow aSb \mid \varepsilon$

(ii) $S \rightarrow aS \mid Sb \mid \varepsilon$

(iii) $S \rightarrow AS \mid SB \mid \varepsilon$

$A \rightarrow a$

$B \rightarrow b$

(iv) $S \rightarrow AS \mid SB$

$A \rightarrow a \mid \varepsilon$

$B \rightarrow b$

Closure properties of CFL's

- CFL's are closed under:
  (i) Union
  (ii) Concatenation
  (iii) Kleene Star
- What about intersection and complement?

The setting

- $L_1 = L(G_1)$ where

  $G_1 = (V_1, T, P_1, S_1)$
- $L_2 = L(G_2)$ where

  $G_2 = (V_2, T, P_2, S_2)$
- Assume wlog that $V_1 \cap V_2 = \varnothing$

Closure under Union -- Example

- $L_1 = \{ a^n b^n \mid n \geq 0 \}$
- $L_2 = \{ b^n a^n \mid n \geq 0 \}$
- G_1 ?
  - Ans: $S_1 \rightarrow aS_1b \mid \varepsilon$
- G_2 ?
  - Ans: $S_2 \rightarrow bS_2a \mid \varepsilon$
- How to make grammar for $L_1 \cup L_2$?
  - Ans: Idea: Add new start symbol S and rules

$$S \rightarrow S_1 \mid S_2$$

Closure under Union
General construction

- Let $G = (V, T, P, S)$ where
  - $V = V_1 \cup V_2 \cup \{ S \}$, (S is a new start symbol)
  - $S \notin V_1 \cup V_2$
  - $P = P_1 \cup P_2 \cup \{ S \rightarrow S_1 \mid S_2 \}$

Closure under concatenation Example

- $L_1 = \{ a^n b^n \mid n \geq 0 \}$
- $L_2 = \{ b^n a^n \mid n \geq 0 \}$
- Is $L_1 L_2 = \{ a^n b^{\{2n\}} a^n \mid n >= 0 \}$?
  - Ans: No! It is $\{ a^n b^{\{n+m\}} a^m \mid n, m \geq 0 \}$
- How to make grammar for $L_1 L_2$?
  - Idea: Add new start symbol and rule $S \rightarrow S_1 S_2$

Closure under concatenation General construction

- Let $G = (V, T, P, S)$ where
  - $V = V_1 \cup V_2 \cup \{ S \}$,
  - $S \notin V_1 \cup V_2$
  - $P = P_1 \cup P_2 \cup \{ S \rightarrow S_1S_2 \}$

S is a new start symbol and $S \rightarrow S_1S_2$ is a new rule.

Closure under kleene star Examples

- $L_1 = \{a^n b^n \mid n \geq 0\}$
- What is $(L_1)^*$?
  - Ans: $\{ a^{\{n1\}}b^{\{n1\}} \ldots a^{\{nk\}}b^{\{nk\}} \mid k \geq 0$ and $n^i \geq 0$ for all i $\}$
- $L_2 = \{ a^{\{n^2\}} \mid n \geq 1 \}$
- What is $(L_2)^*$?
  - Ans: a^*. Why?
- How to make grammar for $(L_1)^*$?
  - Idea: Add new start symbol S and rules $S \to SS_1 \mid \varepsilon$.

Closure under kleene star General construction

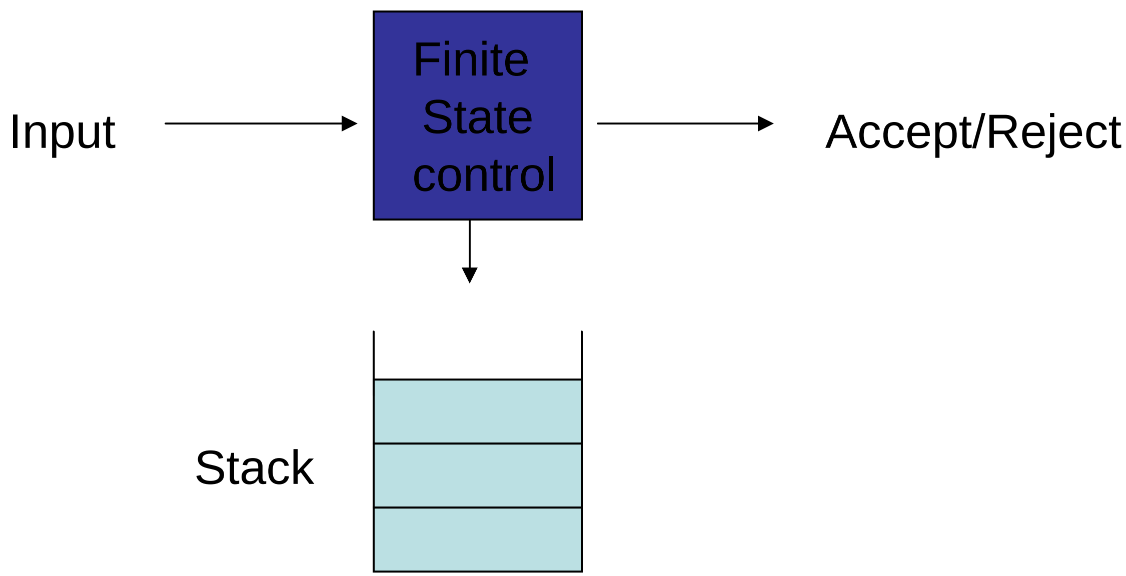
- Let $G = (V, T, P, S)$ where
  - $V = V_1 \cup \{ S \}$,
  - $S \notin V_1$
  - $P = P_1 \cup \{ S \to SS_1 \mid \varepsilon \}$

Tips for Designing CFG's

- Use closure properties -- divide and conquer
- Analyze strings – Is order important? Number? Do we need recursion?
- Flat vs. hierarchical?
- Are any possibilities (strings) missing?
- Is the grammar generating too many strings?

Push Down Automaton (PDA)

- Language Acceptor Model for CFLs.
- It is an NFA with a stack.

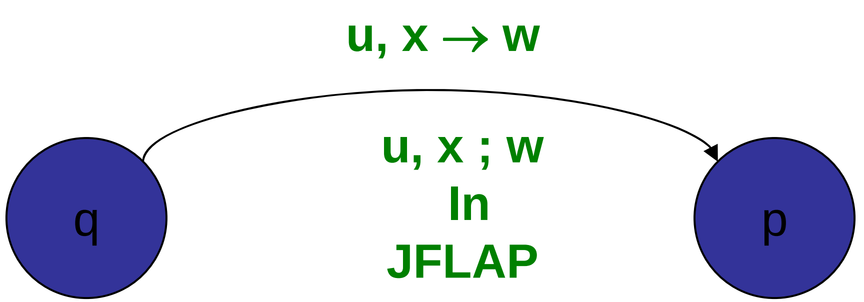


PDA (contd.)

- In one move the PDA can :
  - change state,
  - consume a symbol from the input tape or ignore it,
  - pop a symbol from the stack or ignore it,
  - push a symbol onto the stack or not.
- A string is accepted provided the machine when started in the start state consumes the string and reaches a final state.

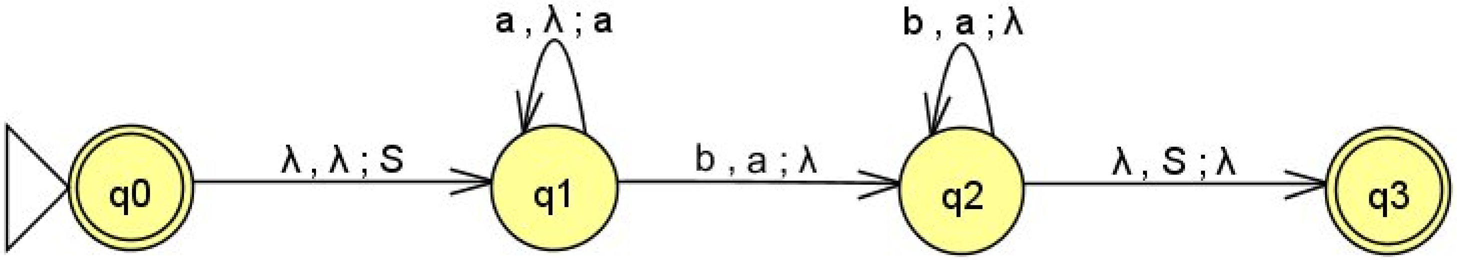
PDA (contd.)

- If PDA in state q can consume u, pop x from stack, change state to p, and push w on stack, we show it as

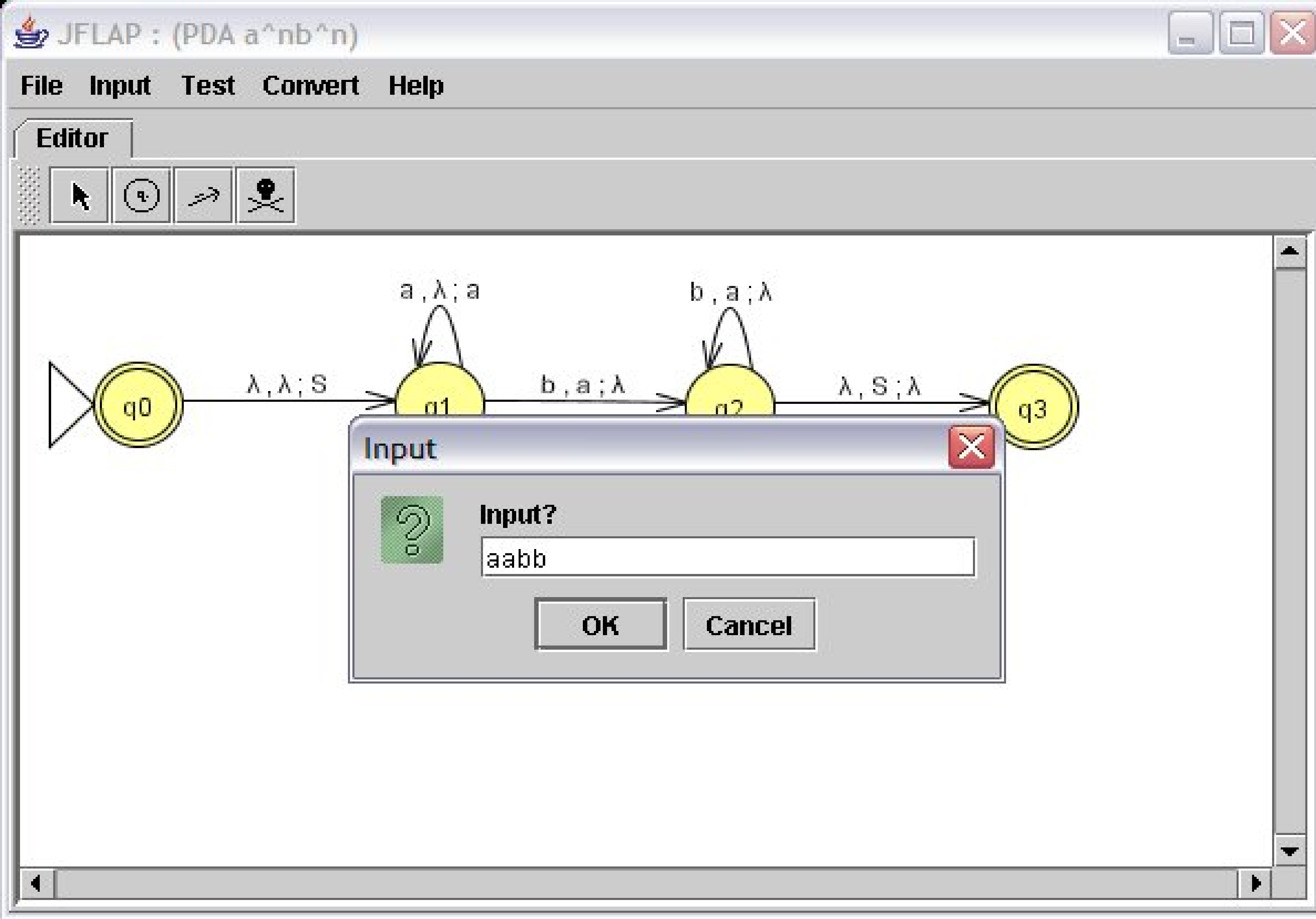


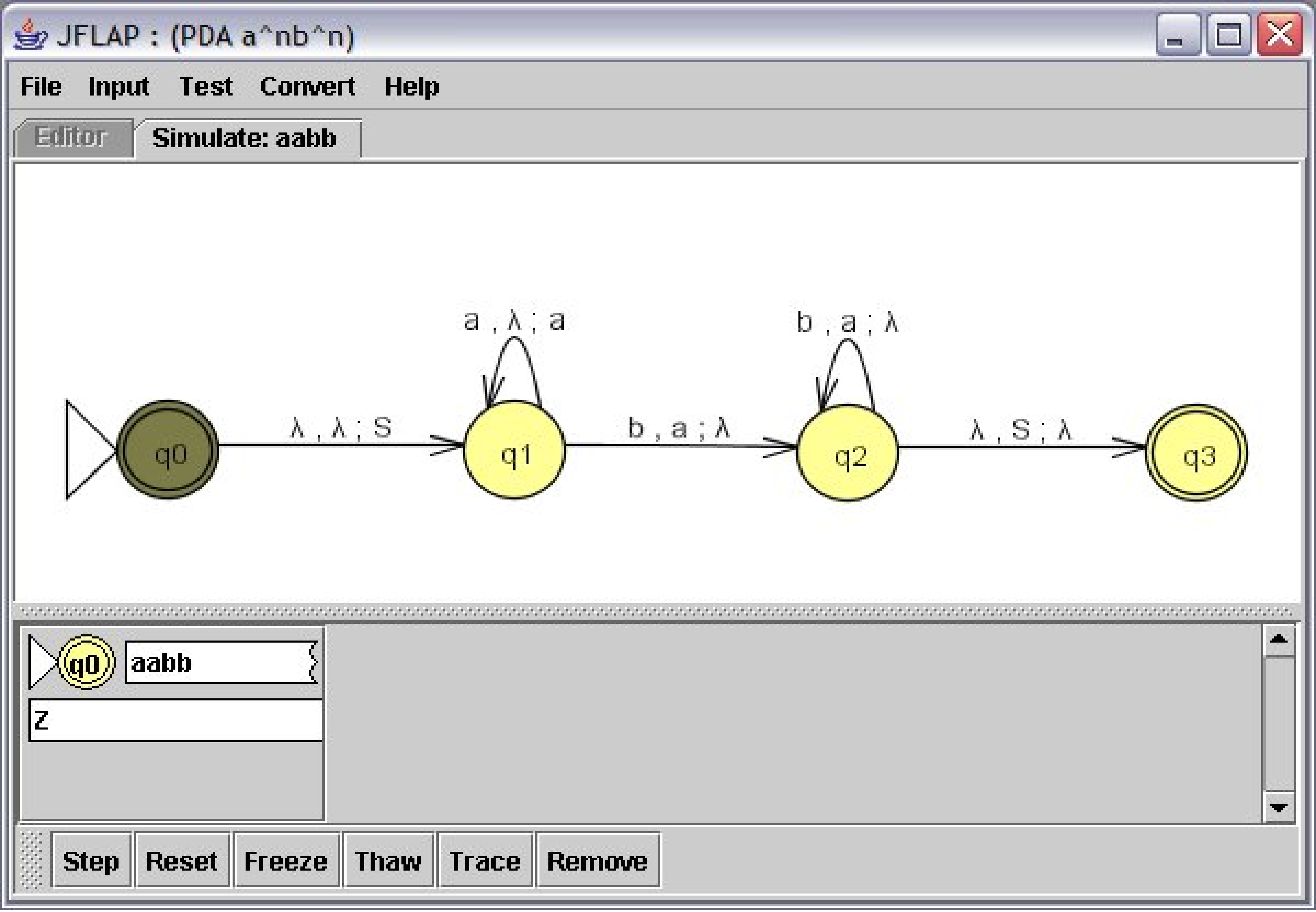
Example of a PDA

- PDA $L = \{a^n b^n \mid n \geq 0\}$

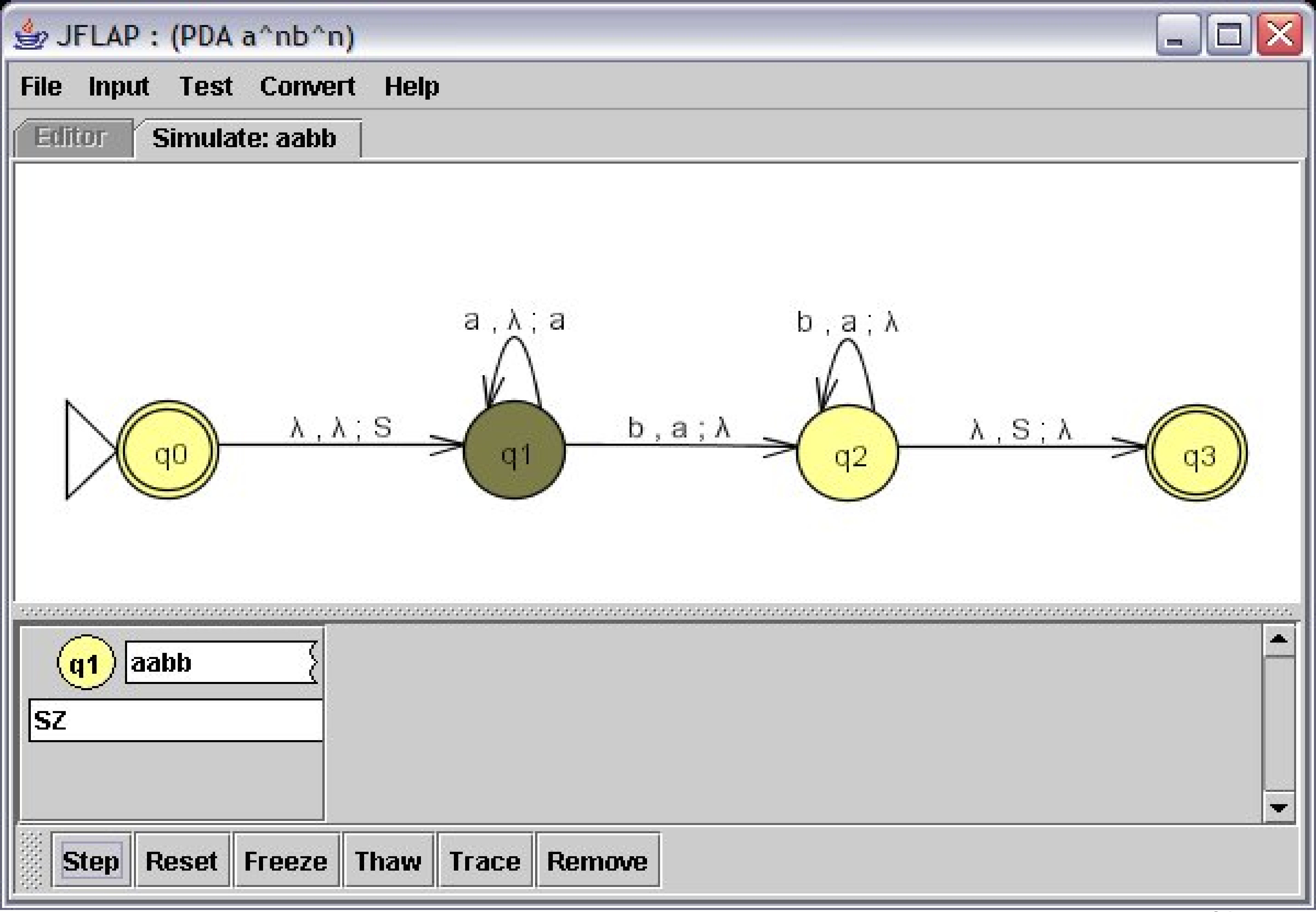


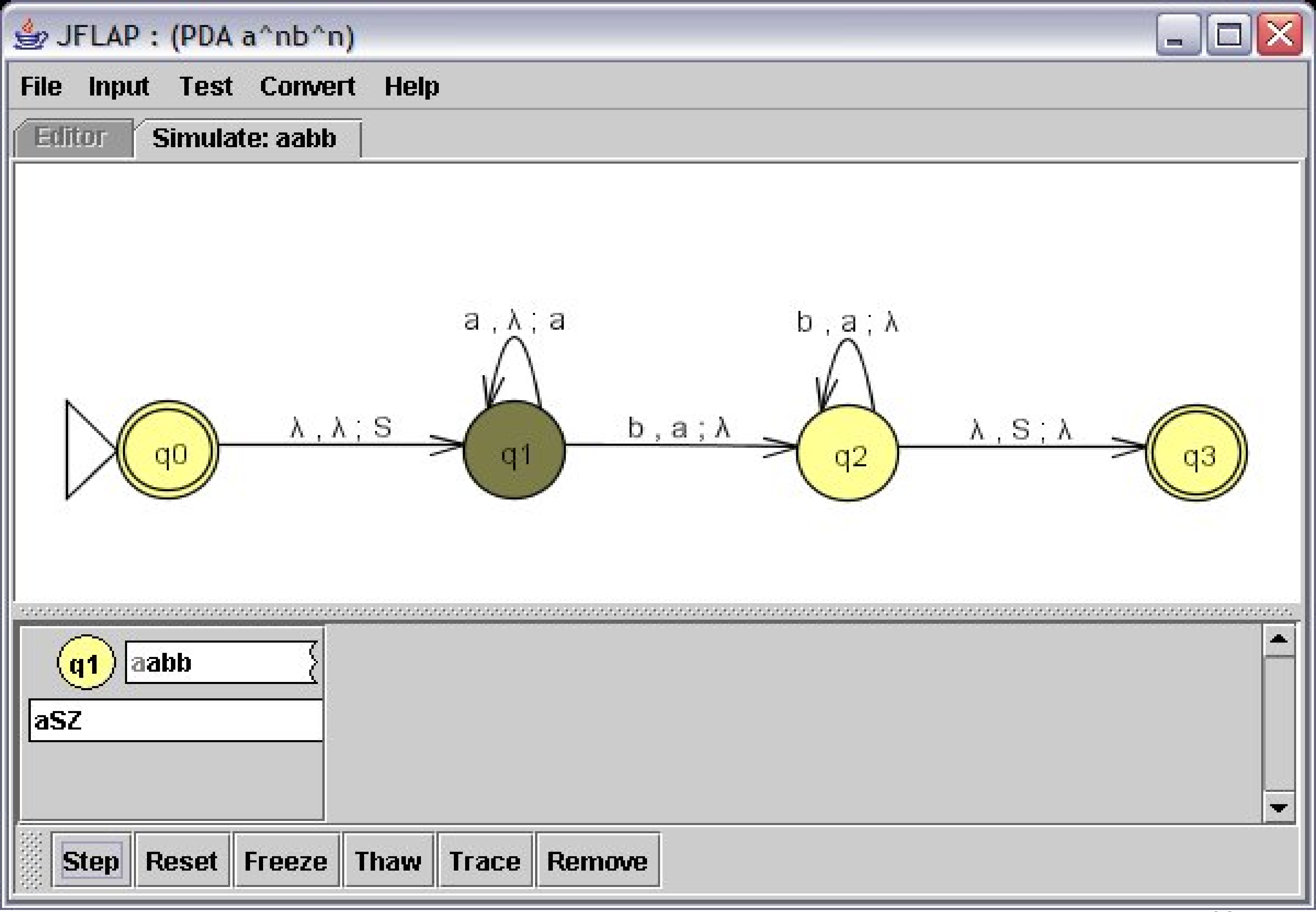
Push S to the stack in the beginning and then pop it at the end before accepting.

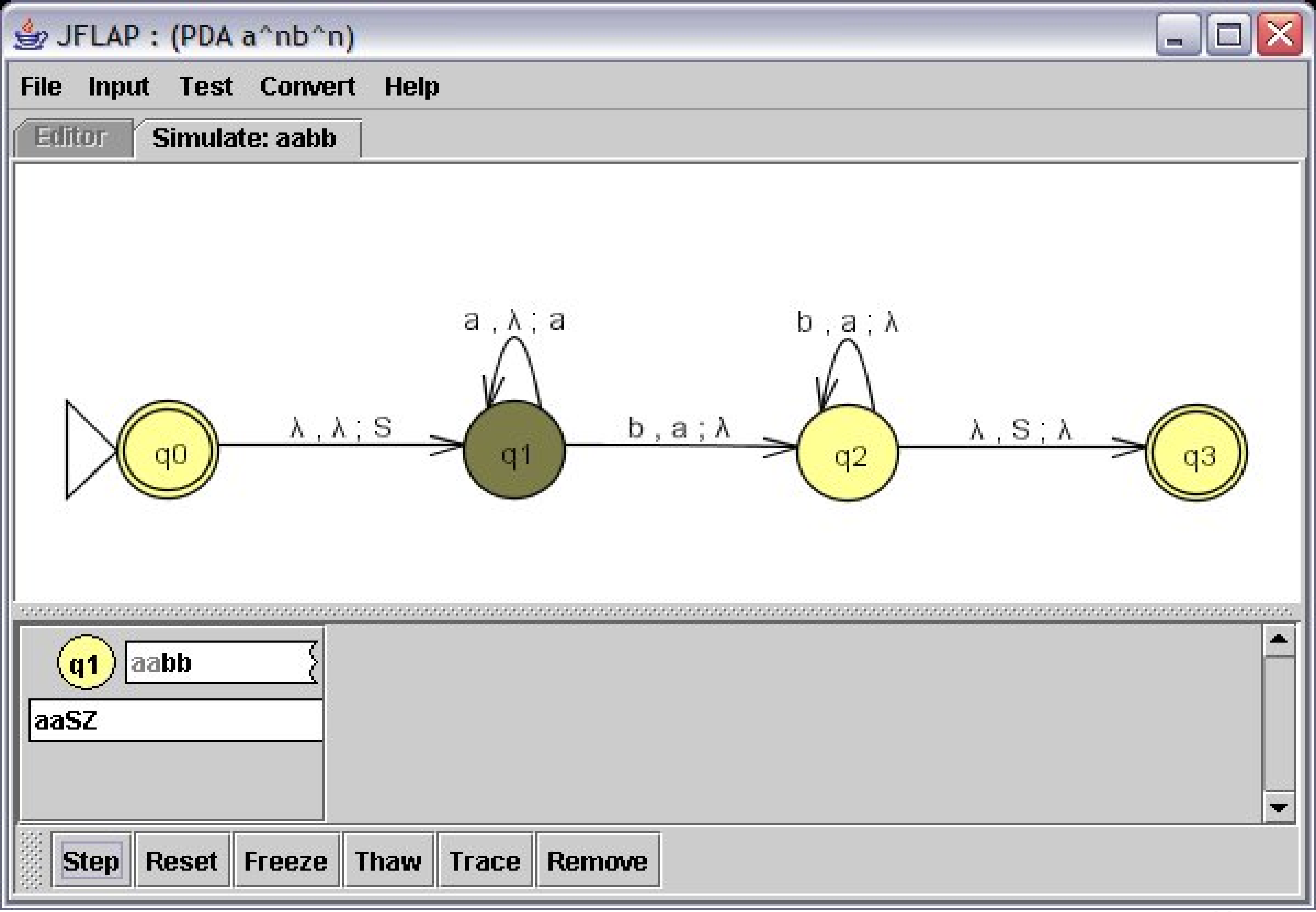

JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q3
Input
Input?
aabb
OK
Cancel

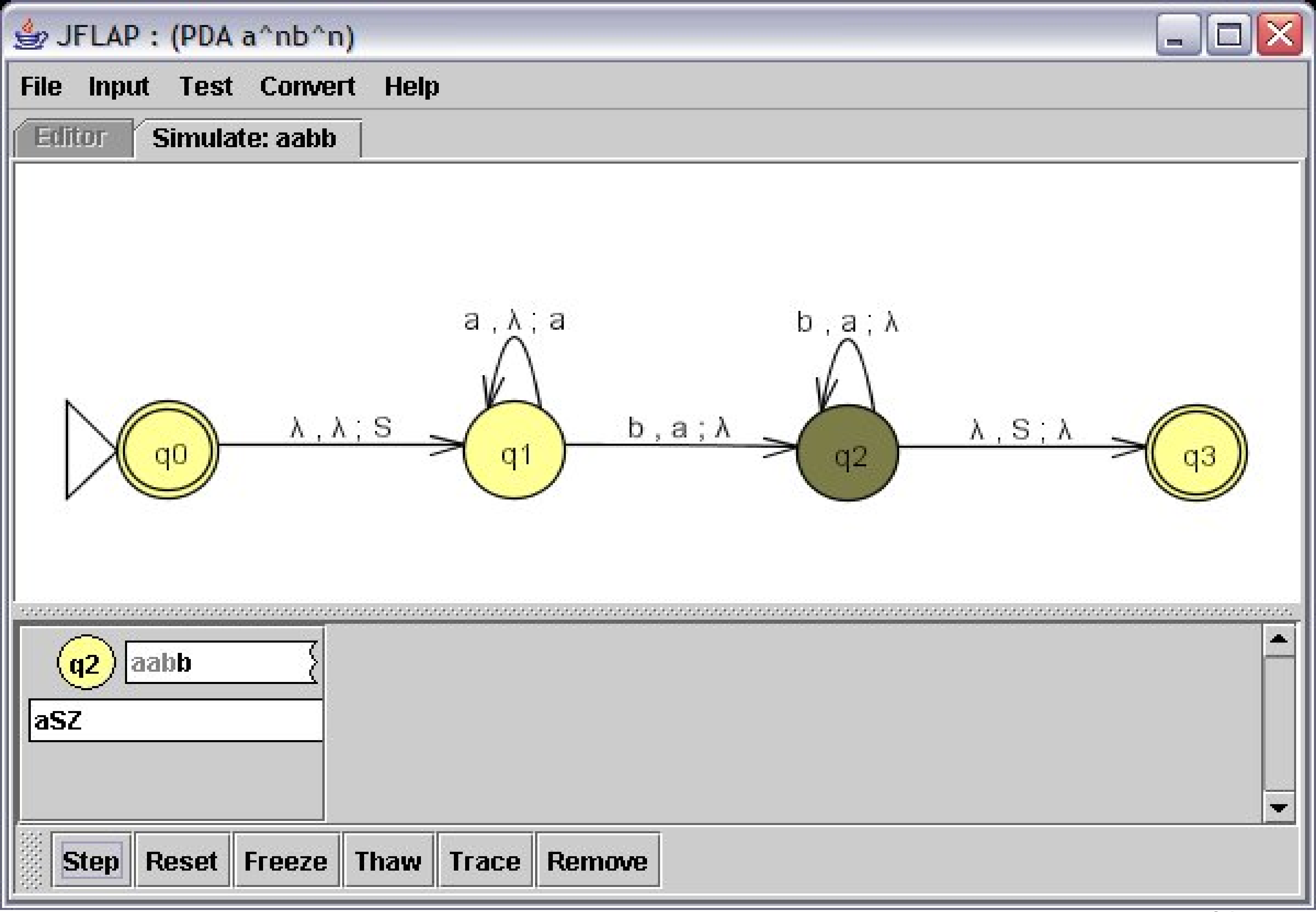


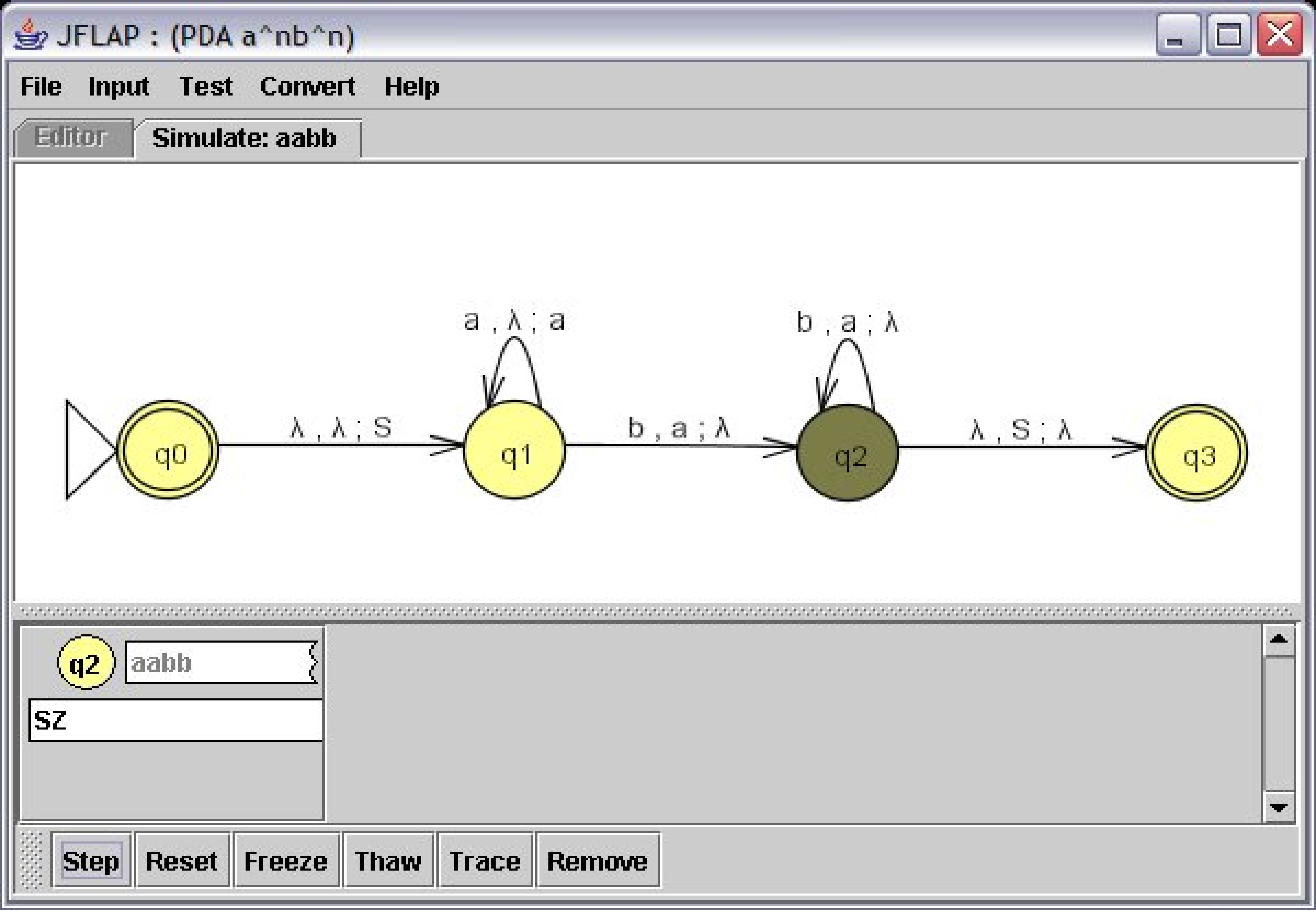
JFLAP : (PDA a^nb^n)
File
Input
Test
Convert
Help
Editor
Simulate: aabb
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q0
aabb
Z
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (PDA a^nb^n)
File
Input
Test
Convert
Help
Editor
Simulate: aabb
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q1
aabb
SZ
Step
Reset
Freeze
Thaw
Trace
Remove

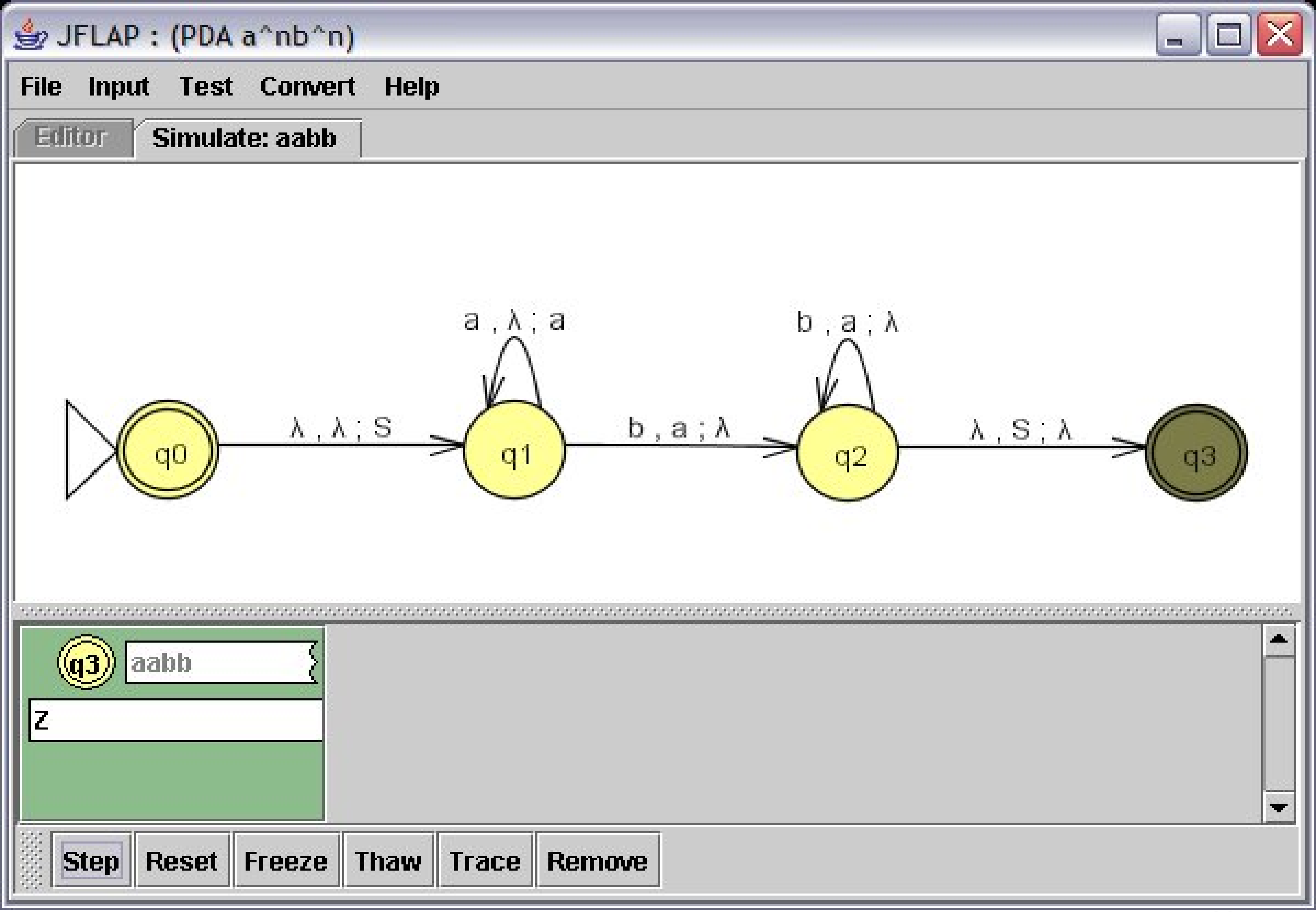

JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
Simulate: aabb
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q1
aabb
aSZ
Step
Reset
Freeze
Thaw
Trace
Remove

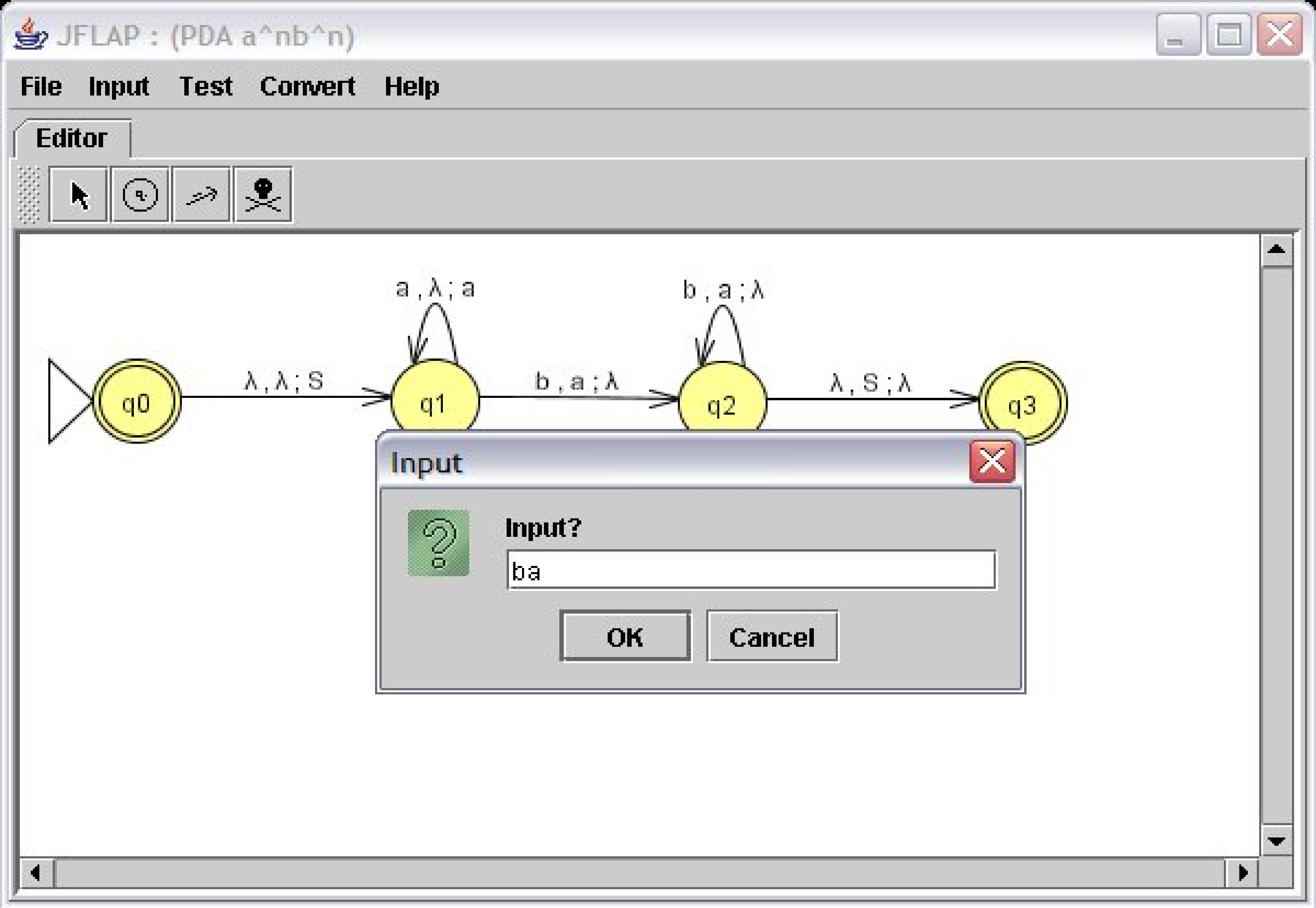

JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
Simulate: aabb
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q1
aabb
aaSZ
Step
Reset
Freeze
Thaw
Trace
Remove

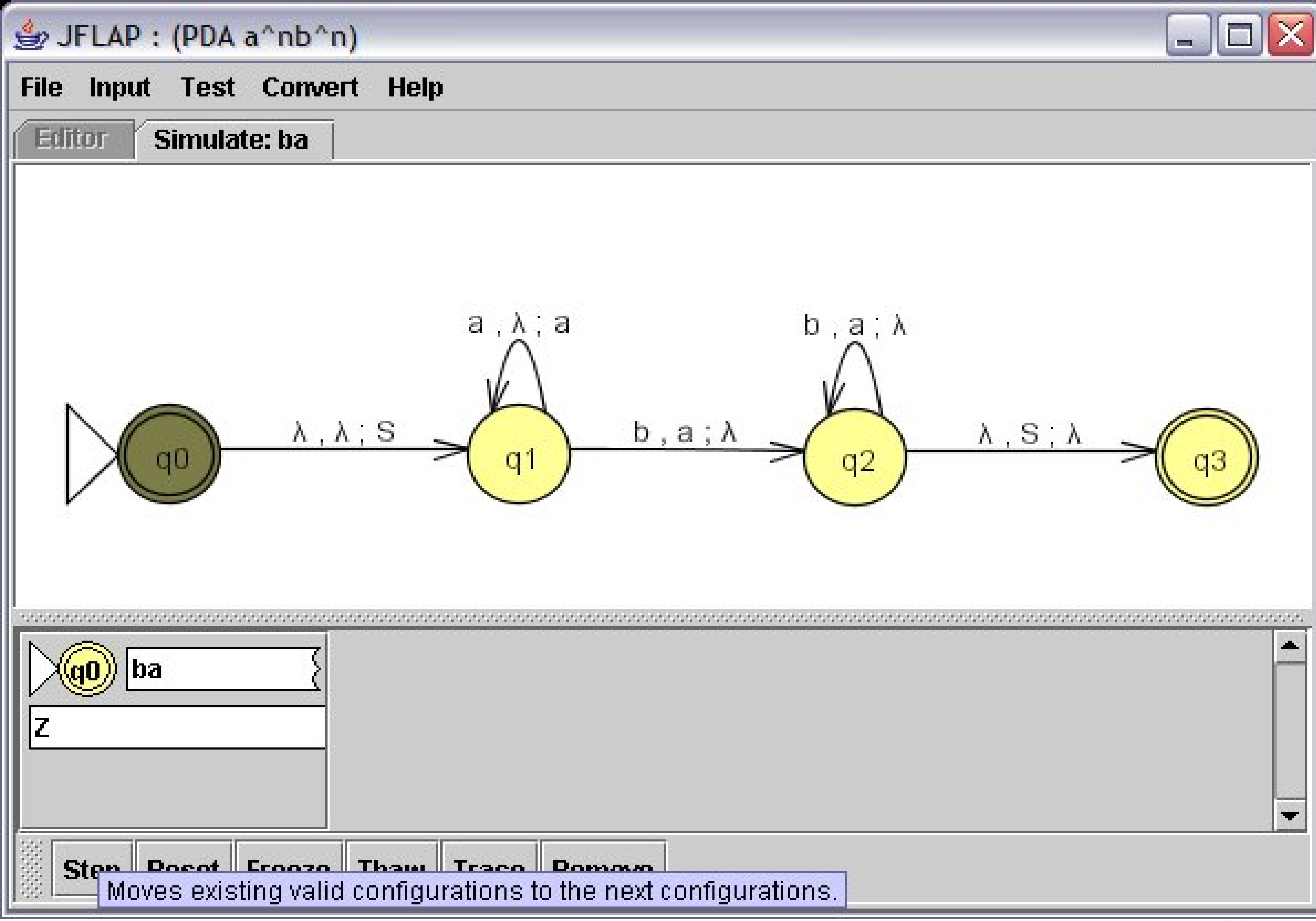

JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
Simulate: aabb
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q2
aabb
aSZ
Step
Reset
Freeze
Thaw
Trace
Remove

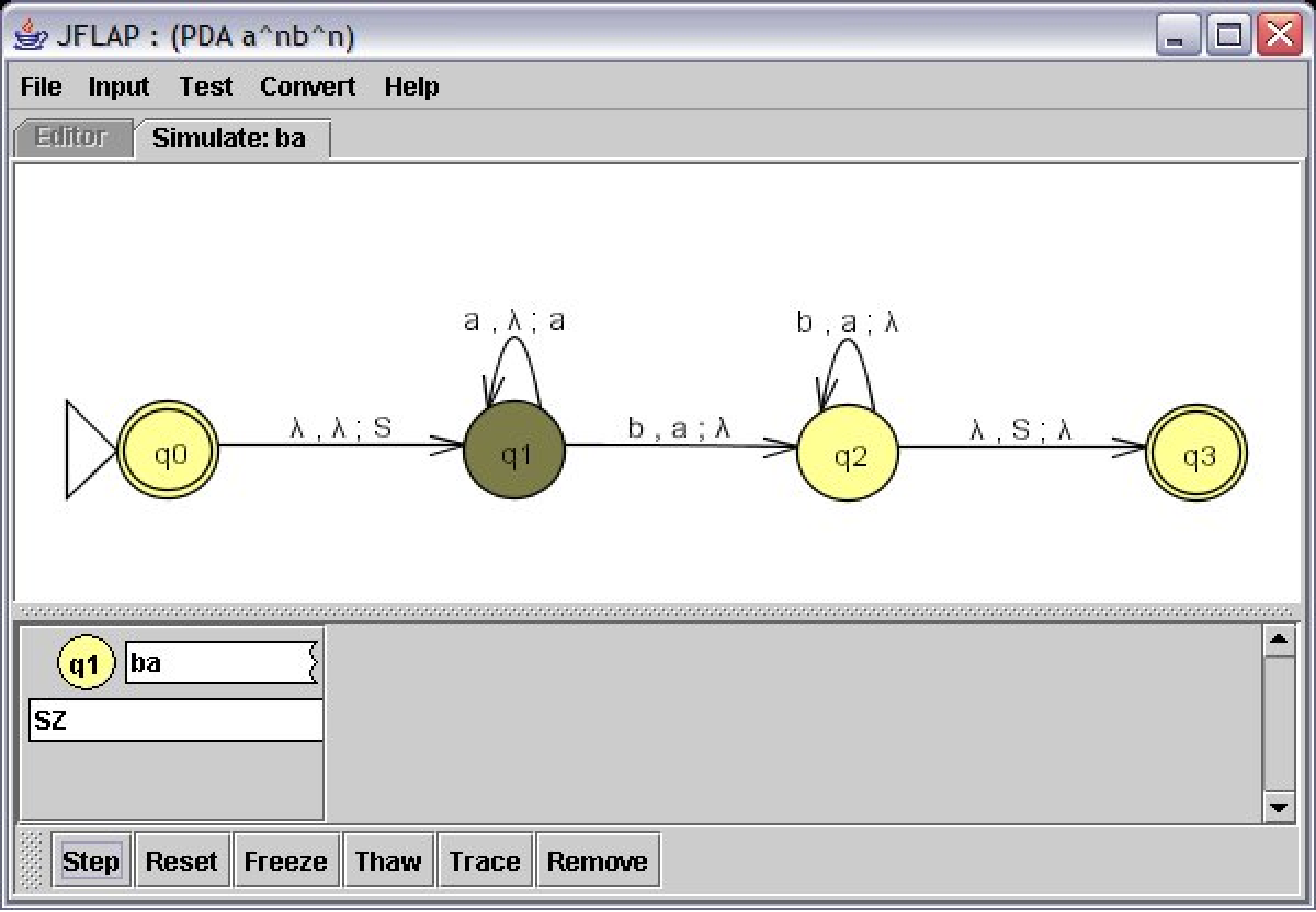


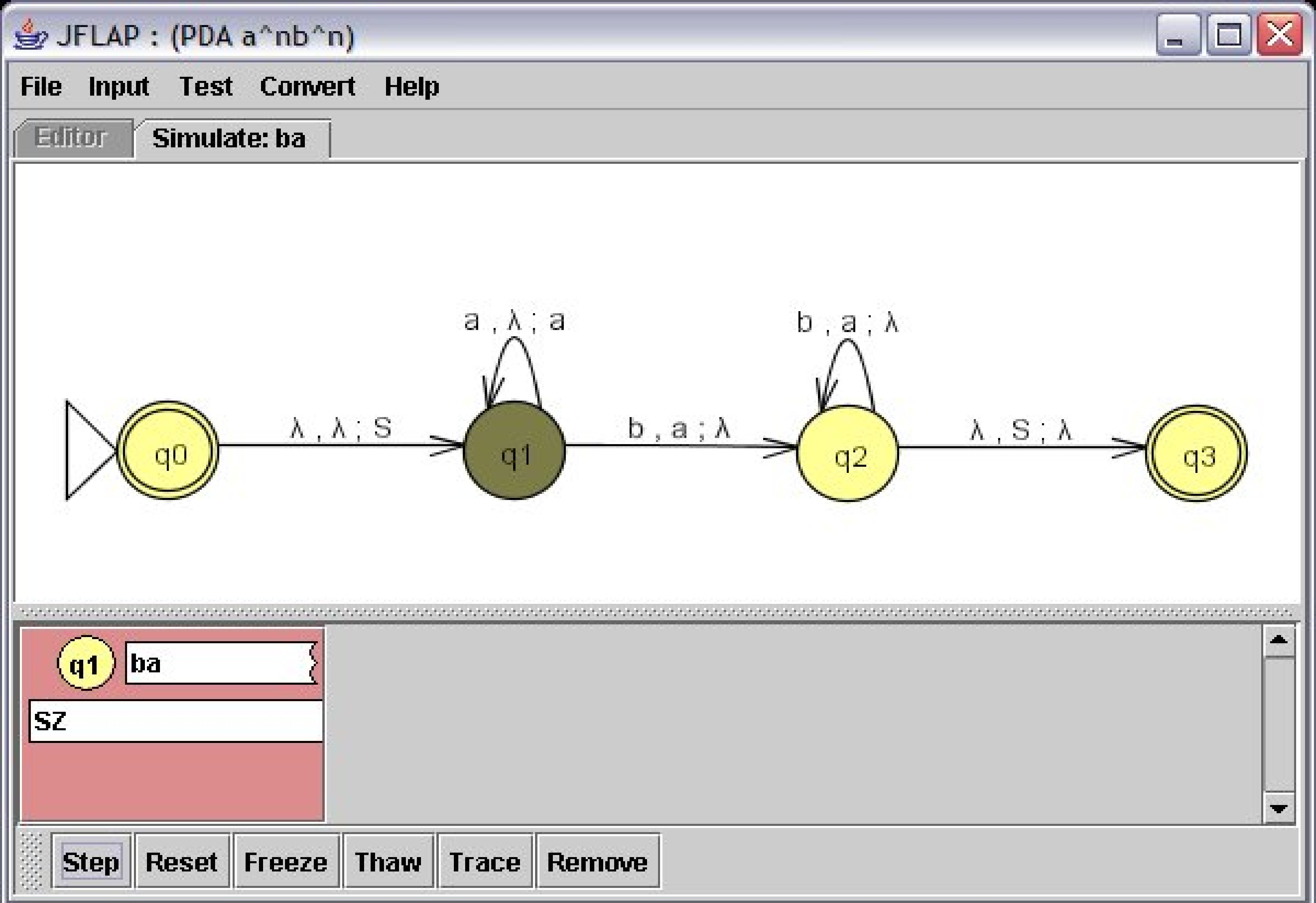
JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
Simulate: aabb
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q2
aabb
SZ
Step
Reset
Freeze
Thaw
Trace
Remove

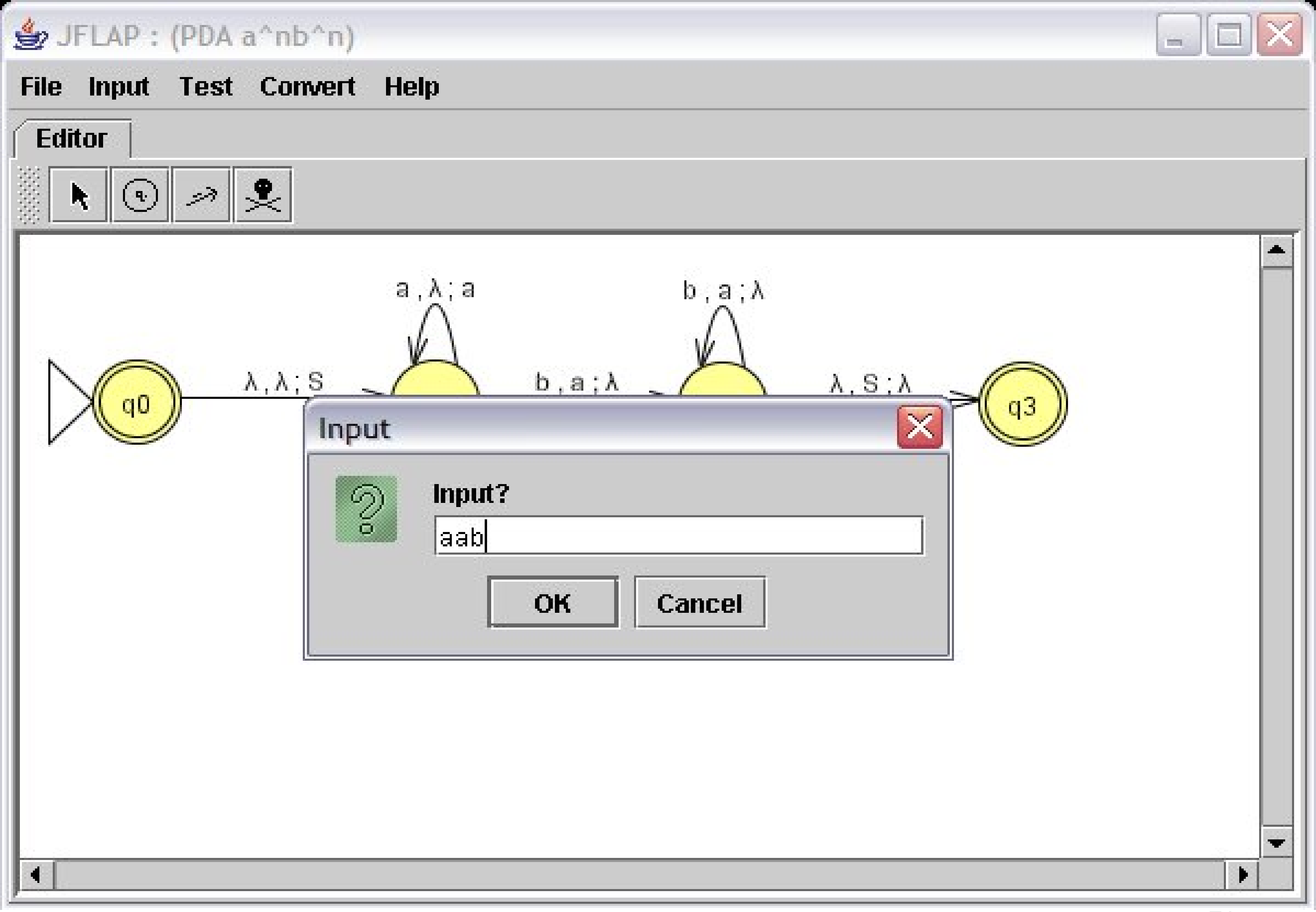

JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
Simulate: aabb
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q3
aabb
Z
Step Reset Freeze Thaw Trace Remove

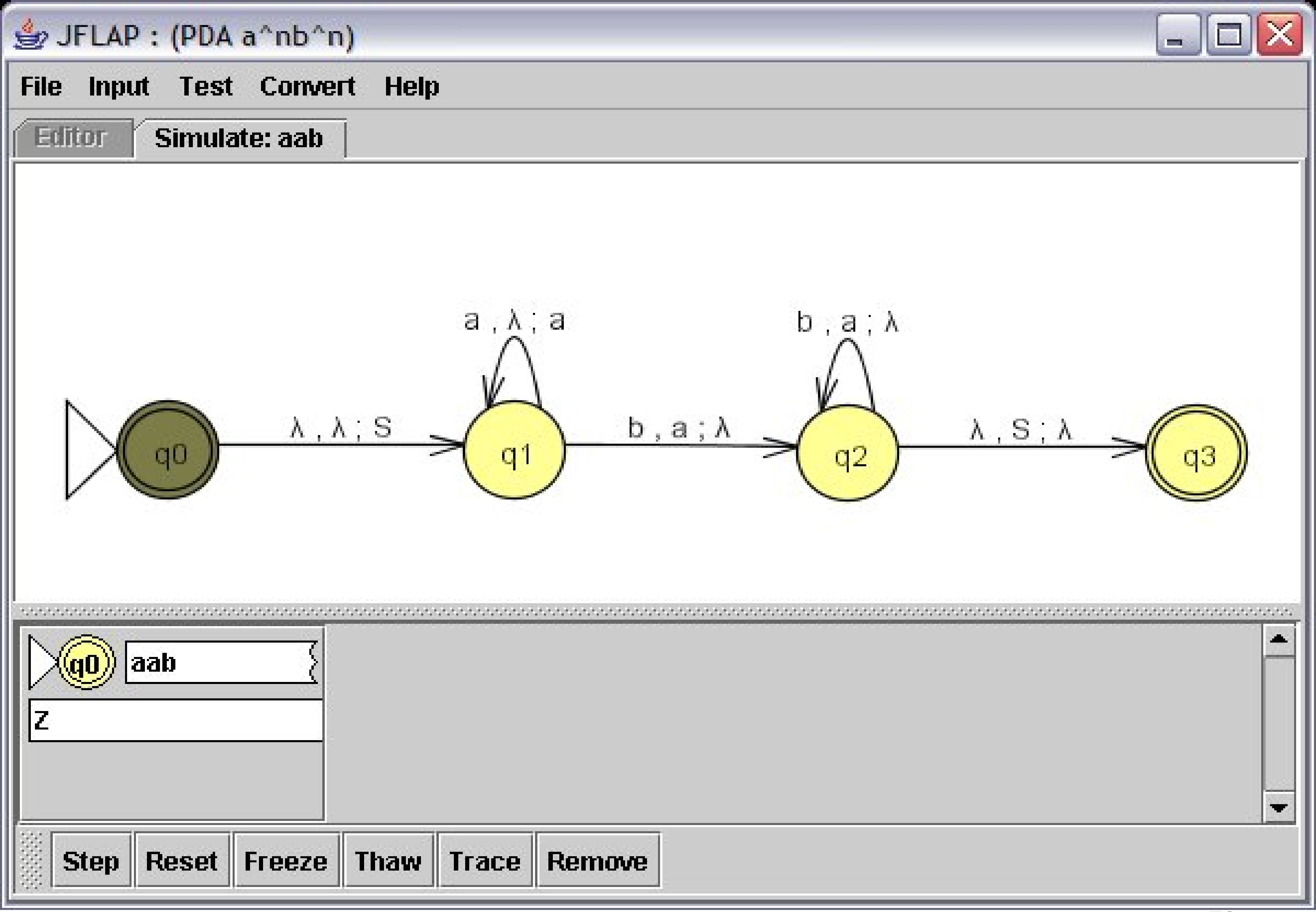

JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
Input
Input?
ba
OK
Cancel

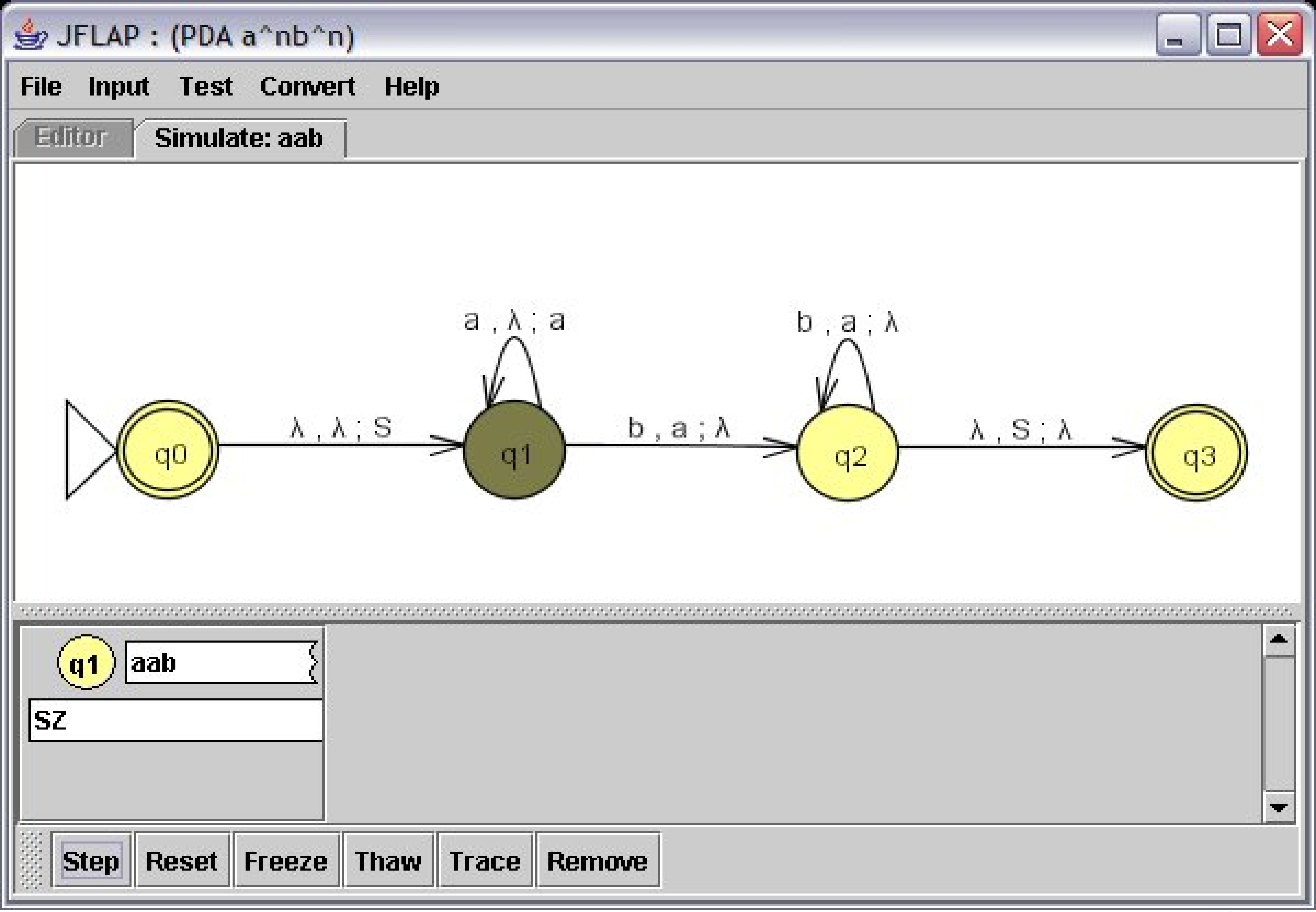

JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
Simulate: ba
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q0
ba
Z
Moves existing valid configurations to the next configurations.

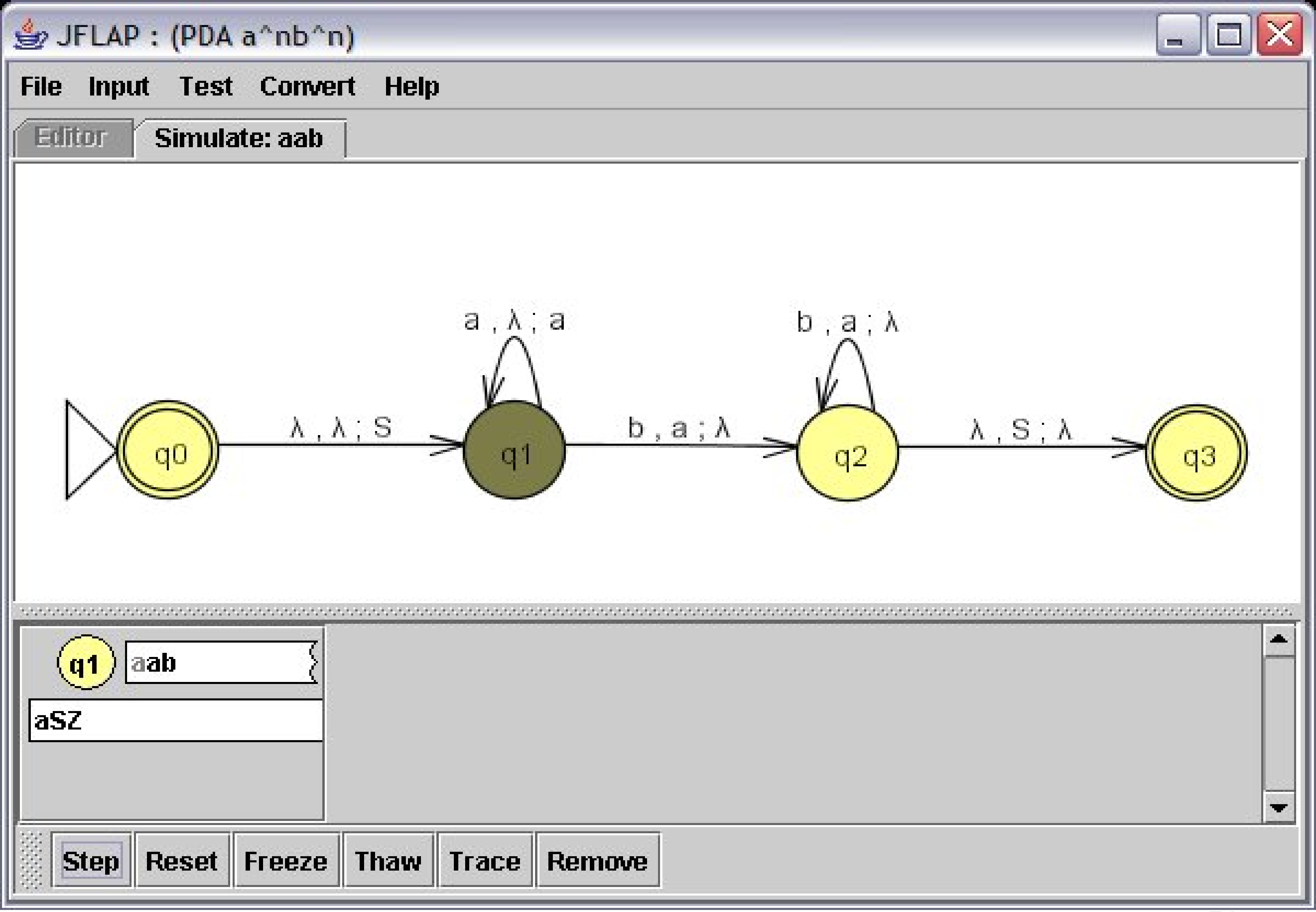

JFLAP : (PDA a^nb^n)
File
Input
Test
Convert
Help
Editor
Simulate: ba
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q1
ba
SZ
Step
Reset
Freeze
Thaw
Trace
Remove

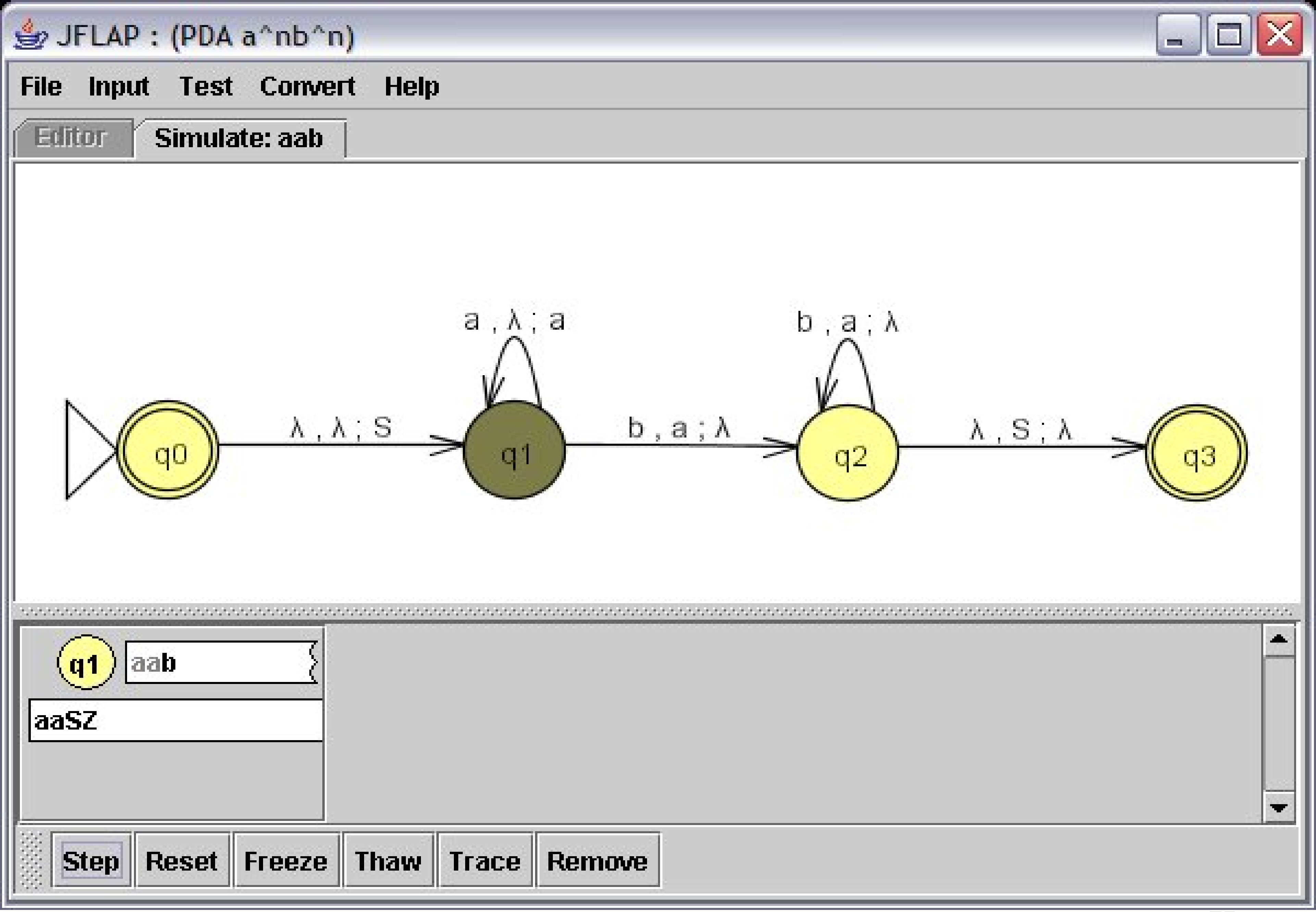

JFLAP : (PDA a^nb^n)
File
Input
Test
Convert
Help
Editor
Simulate: ba
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q1
ba
SZ
Step
Reset
Freeze
Thaw
Trace
Remove

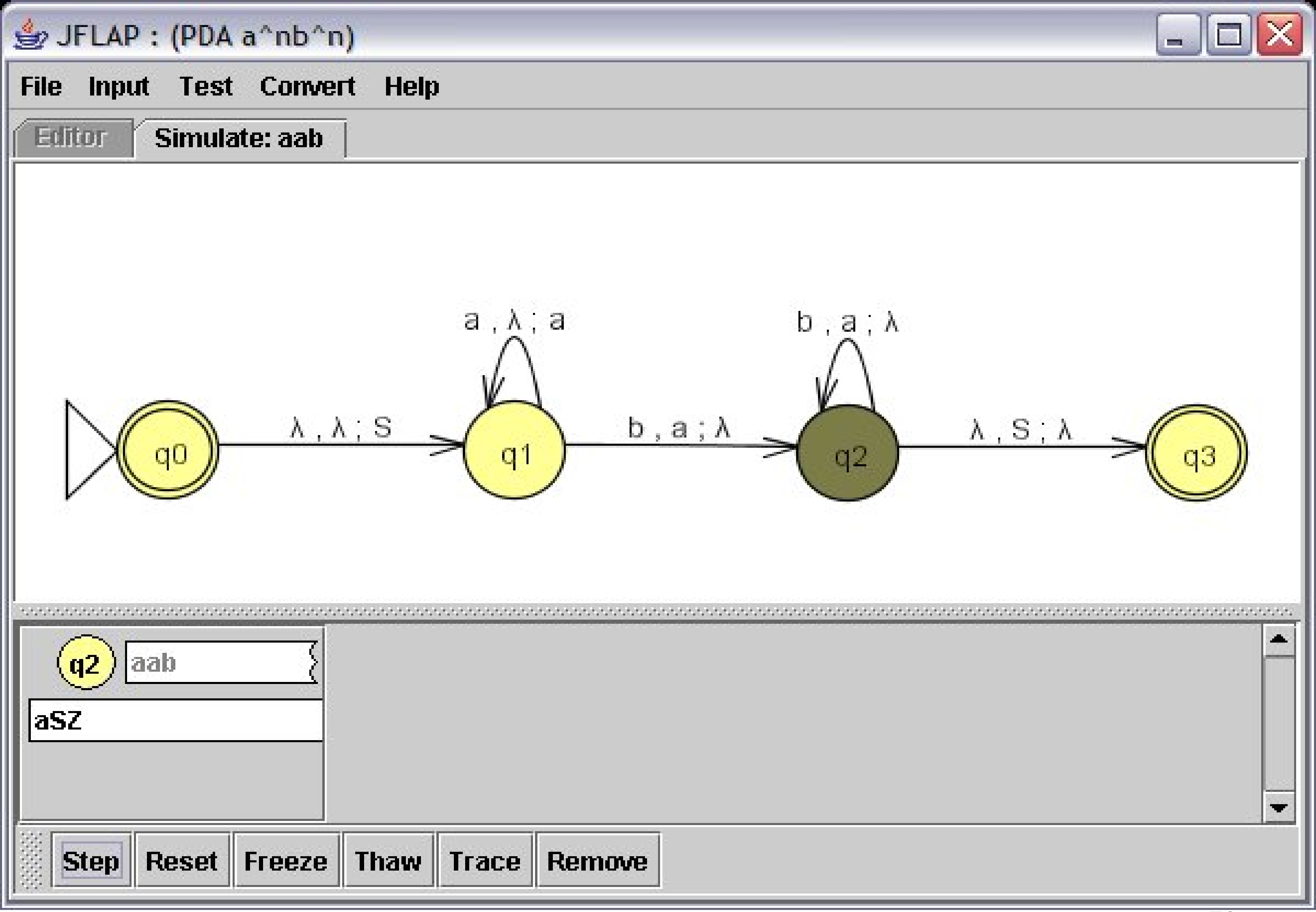

JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q3
Input
Input?
aab
OK
Cancel

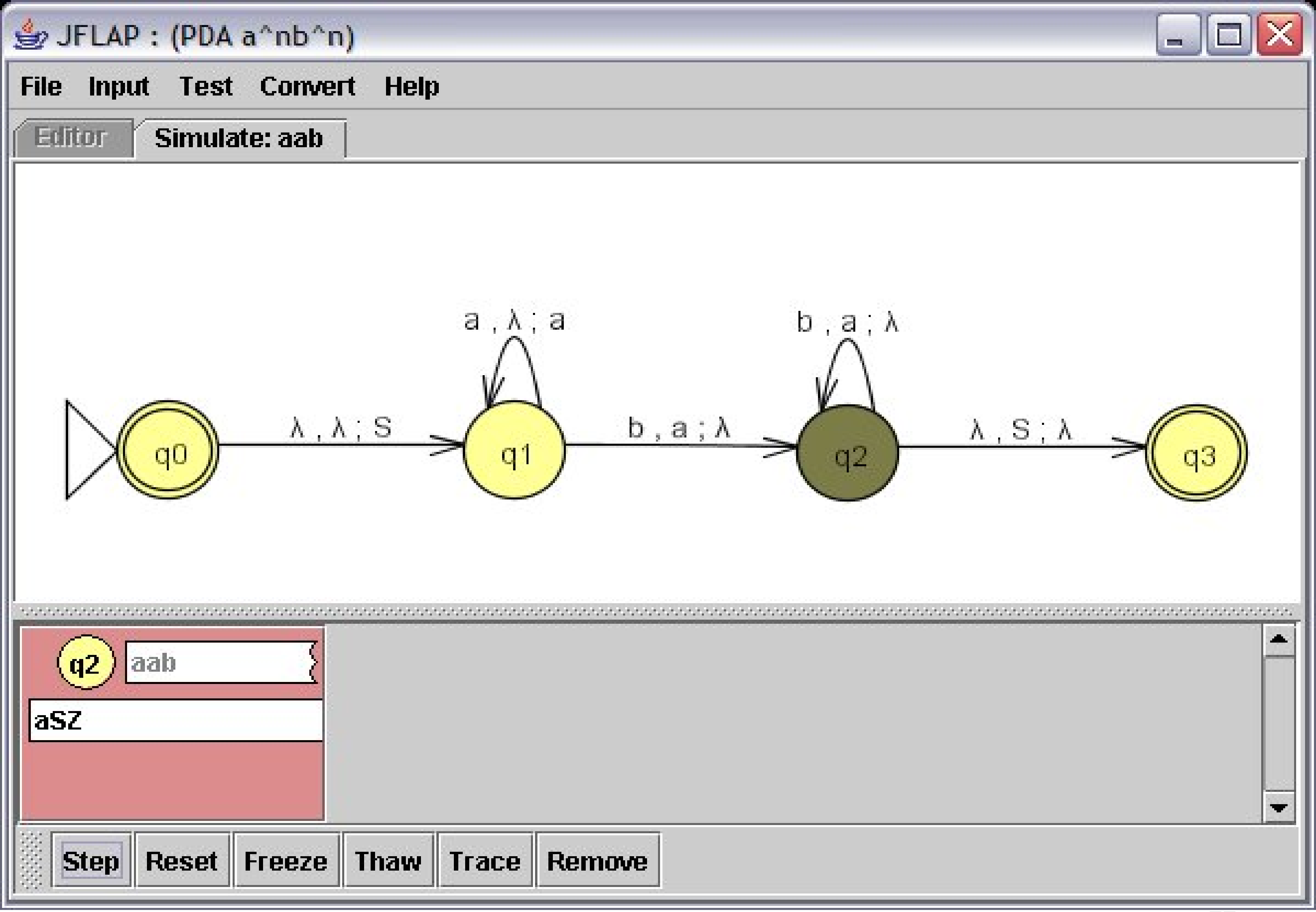

JFLAP : (PDA a^nb^n)
File
Input
Test
Convert
Help
Editor
Simulate: aab
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q0
aab
Z
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (PDA a^nb^n)
File
Input
Test
Convert
Help
Editor
Simulate: aab
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q1
aab
SZ
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
Simulate: aab
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q1
aab
aSZ
Step Reset Freeze Thaw Trace Remove


JFLAP : (PDA a^nb^n)
File
Input
Test
Convert
Help
Editor
Simulate: aab
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q1
aab
aaSZ
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
Simulate: aab
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q2
aab
aSZ
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (PDA a^nb^n)
File Input Test Convert Help
Editor
Simulate: aab
a , λ ; a
b , a ; λ
λ , λ ; S
b , a ; λ
λ , S ; λ
q0
q1
q2
q3
q2
aab
aSZ
Step
Reset
Freeze
Thaw
Trace
Remove

Definition of PDA

- Formally, a PDA M = (K, Σ, Γ, Δ, s, F), where
  - K -- finite set of states
  - Σ -- is the input alphabet
  - Γ -- is the tape alphabet
  - s $\in$ K -- is the start state
  - F $\subseteq$ K -- is the set of final states
  - $\Delta \subseteq (K \times \Sigma_\varepsilon \times \Gamma_\varepsilon) \times (K \times \Gamma_\varepsilon)$

Definition of L(M)

- Define Δ^* as:

  (1) $\Delta^*(q, \varepsilon, \varepsilon) = \{(q, \varepsilon, \varepsilon)\} \cup \{(p, \varepsilon, \varepsilon) \mid ((q, \varepsilon, \varepsilon), (p, \varepsilon)) \in \Delta\}$

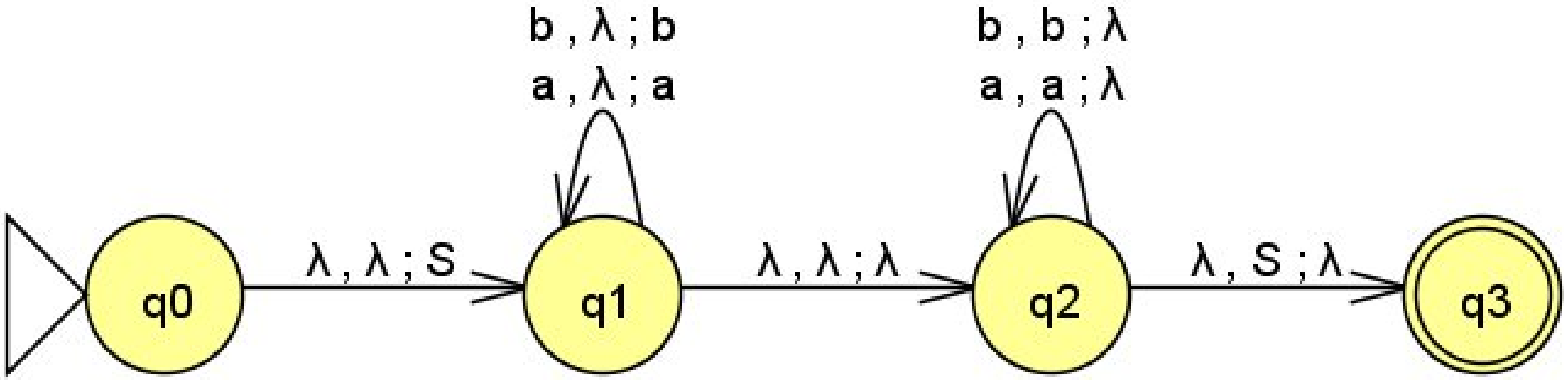
  (2) $\Delta^*(q, uv, xy) = \bigcup \{\Delta^*(p, v, wy) \mid ((q, u, x), (p, w)) \in \Delta\}$

  i.e., first compute Δ^* for all successor configurations and then take the union of all those sets.

- M accepts w if (f, ε, x) in $\Delta^*(s, w, \varepsilon)$
- Alternative: if $(f, \varepsilon, \varepsilon)$ in $\Delta^*(s, w, \varepsilon)$ [we use]
- $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$

Example

- What is L(M)?



Push S to the stack in the beginning and then pop it at the end before accepting.

PDA's and CFG's

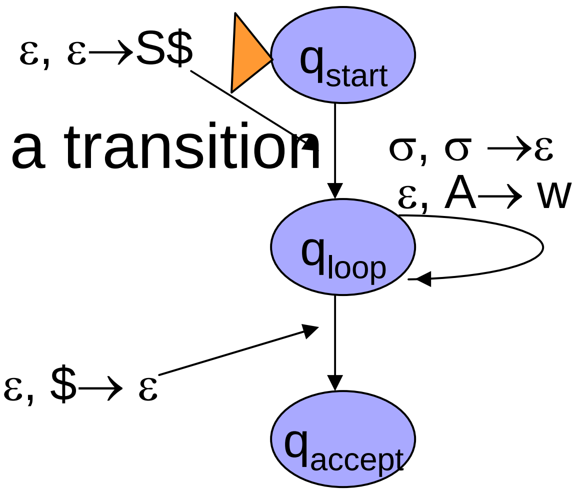
- For every CFG G there is a PDA M such that L(G) = L(M)
- For every PDA M there is a CFG G such that L(M) = L(G)

CFG → PDA

- Given CFG $G = (V, \Sigma, R, S)$
- Let PDA $M = (Q, \Sigma, \Sigma \cup V \cup \{\$\}, \Delta, q_{start}, \{q_{accept}\})$
  - $Q = \{q_{start}, q_{loop}, q_{accept}\}$
- Δ contains transitions for the form
  1. $((q_{start}, \varepsilon, \varepsilon), (q_{loop}, S\$)) \in \Delta$
  2. For each rule $A \to w \in R(G)$ there is a transition

     $((q_{loop}, \varepsilon, A), (q_{loop}, w)) \in \Delta$***
  3. For each symbol $\sigma \in \Sigma$

     $((q_{loop}, \sigma, \sigma), (q_{loop}, \varepsilon)) \in \Delta$
  4. $((q_{loop}, \varepsilon, \$), (q_{accept}, \varepsilon)) \in \Delta$

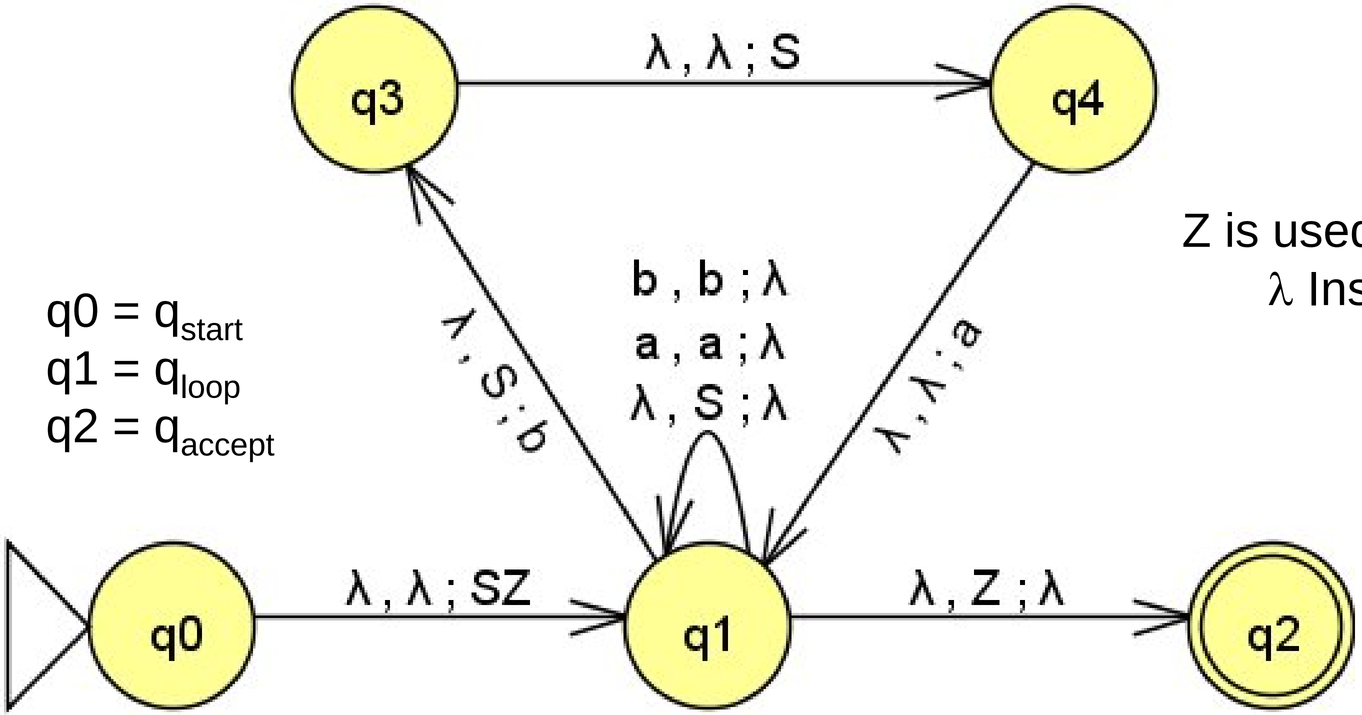


CFG → PDA

- The PDA simulates a leftmost derivation of the string.

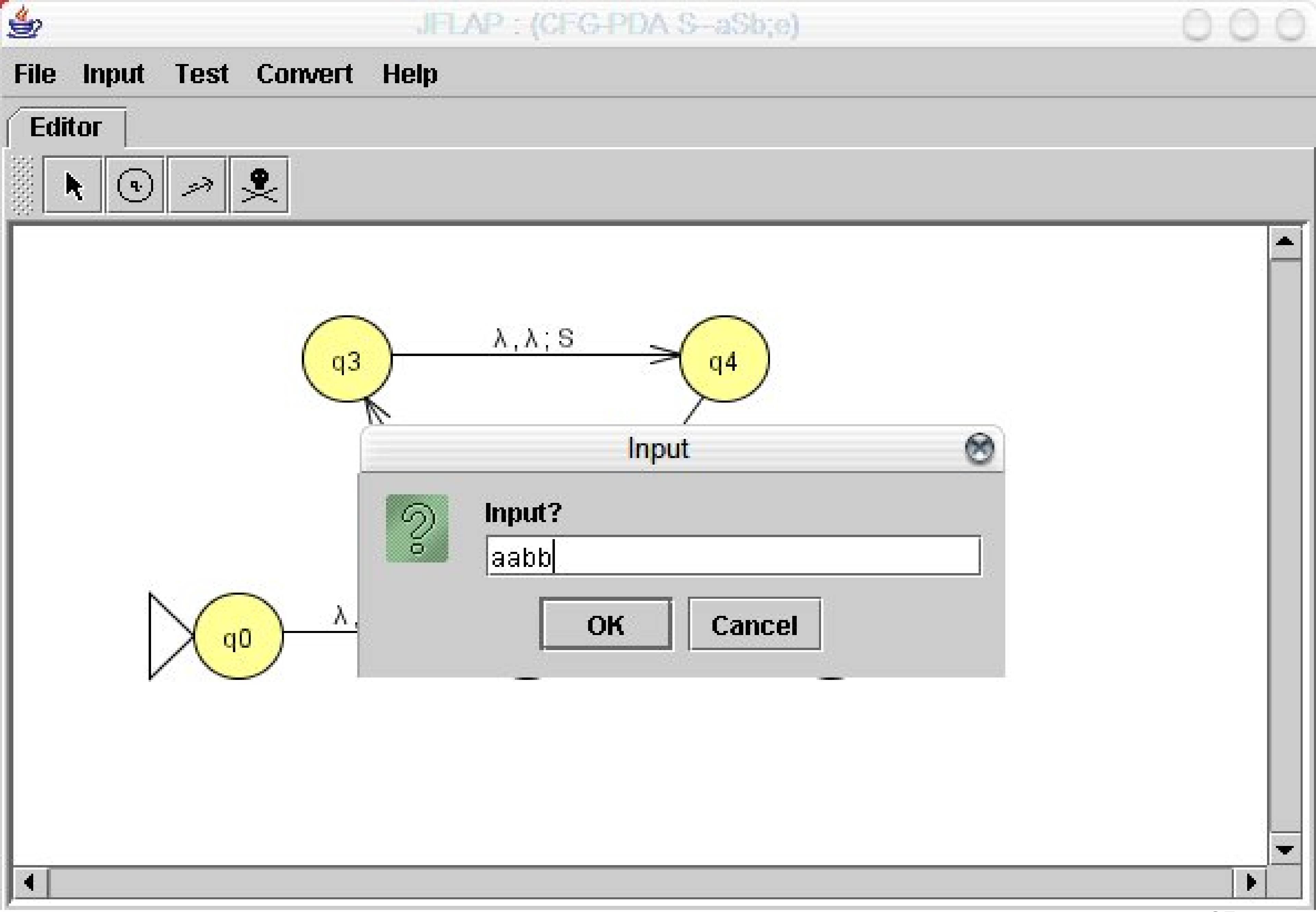
1. Place the marker symbol \$ and the start variable on the stack.
2. Repeat the following steps forever
   (a) If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule.
   (b) If the top of stack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

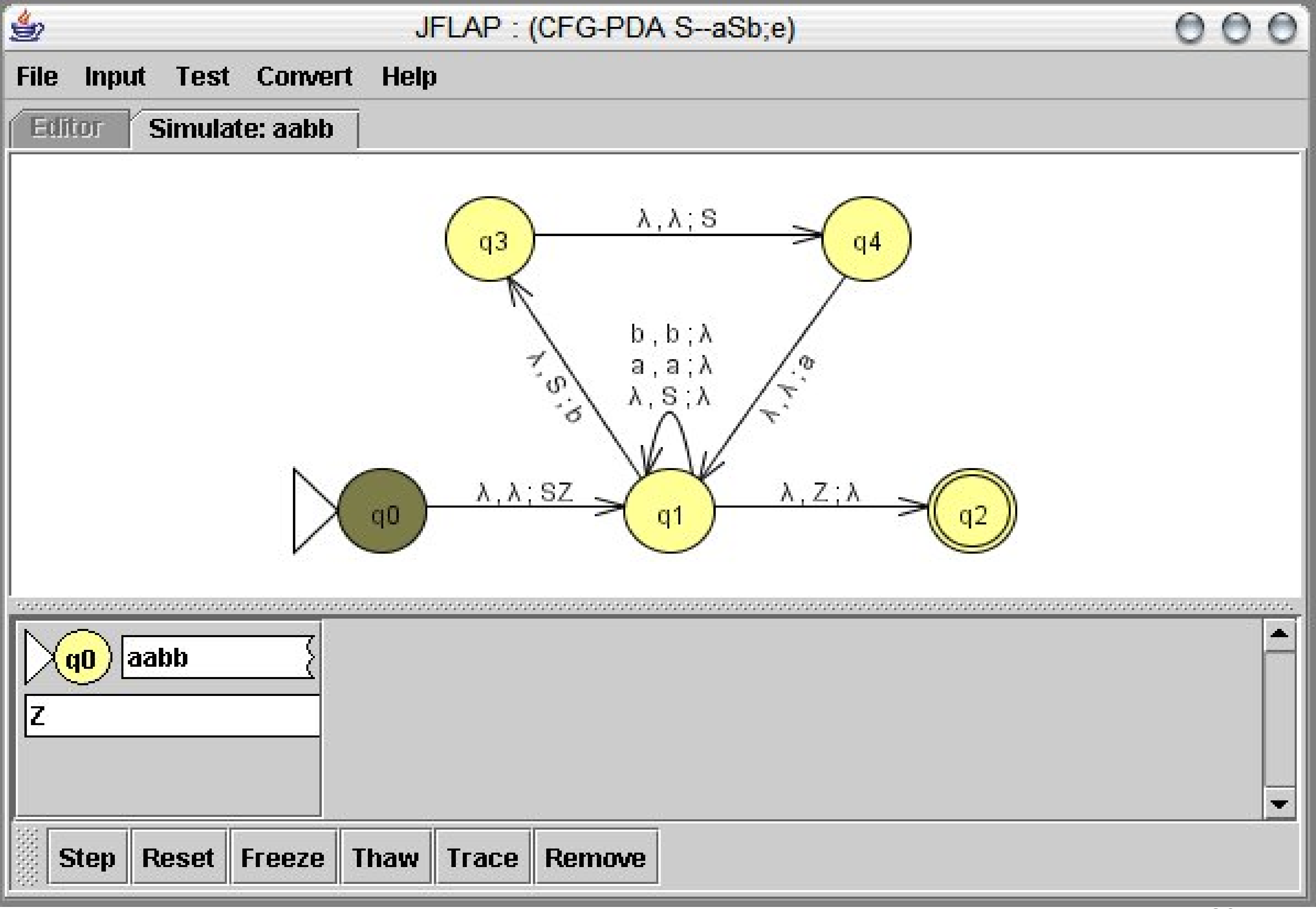
Example: S → aSb | ε

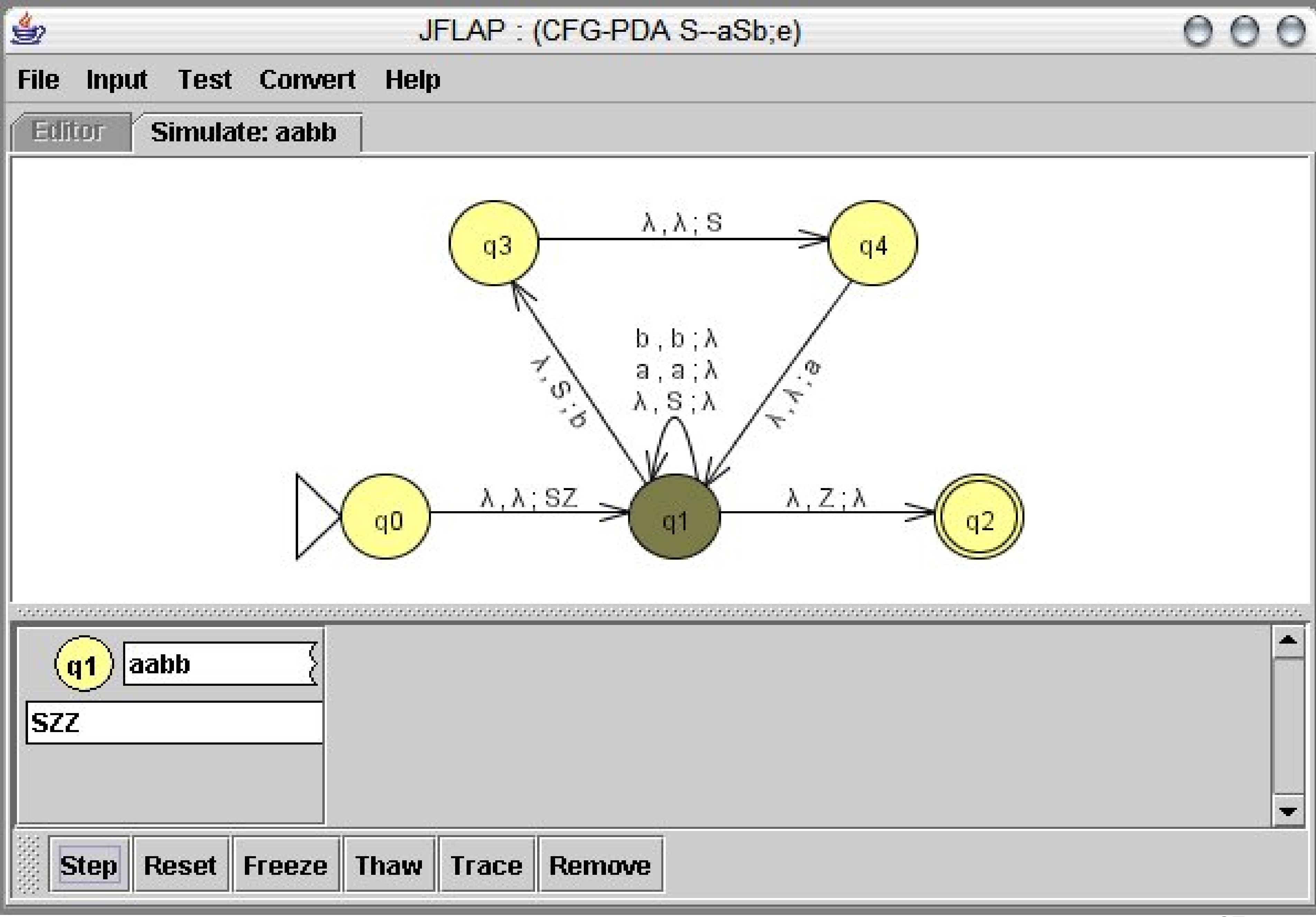


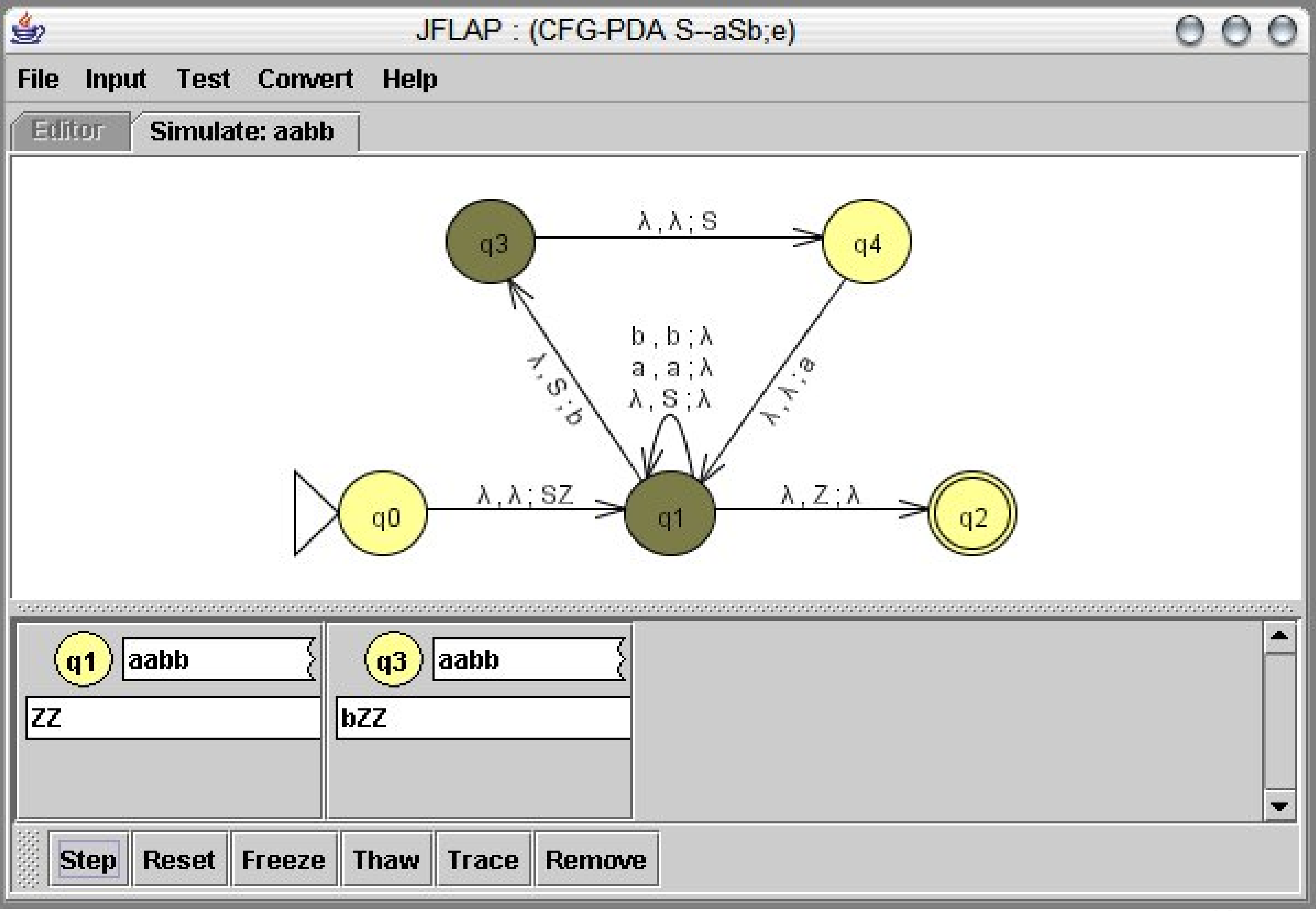
$q0 = q_{start}$
$q1 = q_{loop}$
$q2 = q_{accept}$

Z is used instead of $
λ Instead of ε

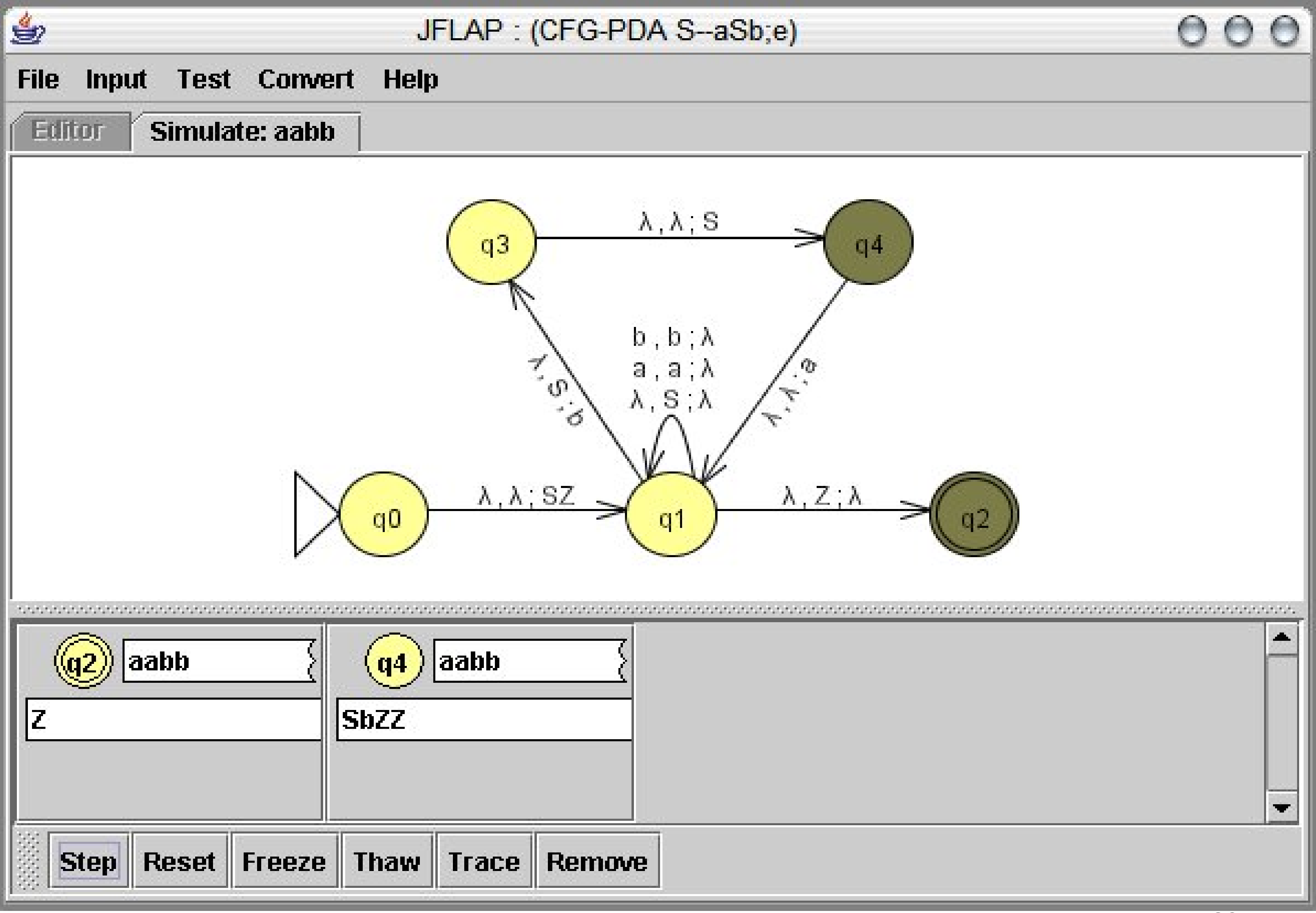

JFLAP : (CFG-PDA S→aSb;e)
File
Input
Test
Convert
Help
Editor
λ , λ ; S
q3
q4
q0
Input
Input?
aabb
OK
Cancel

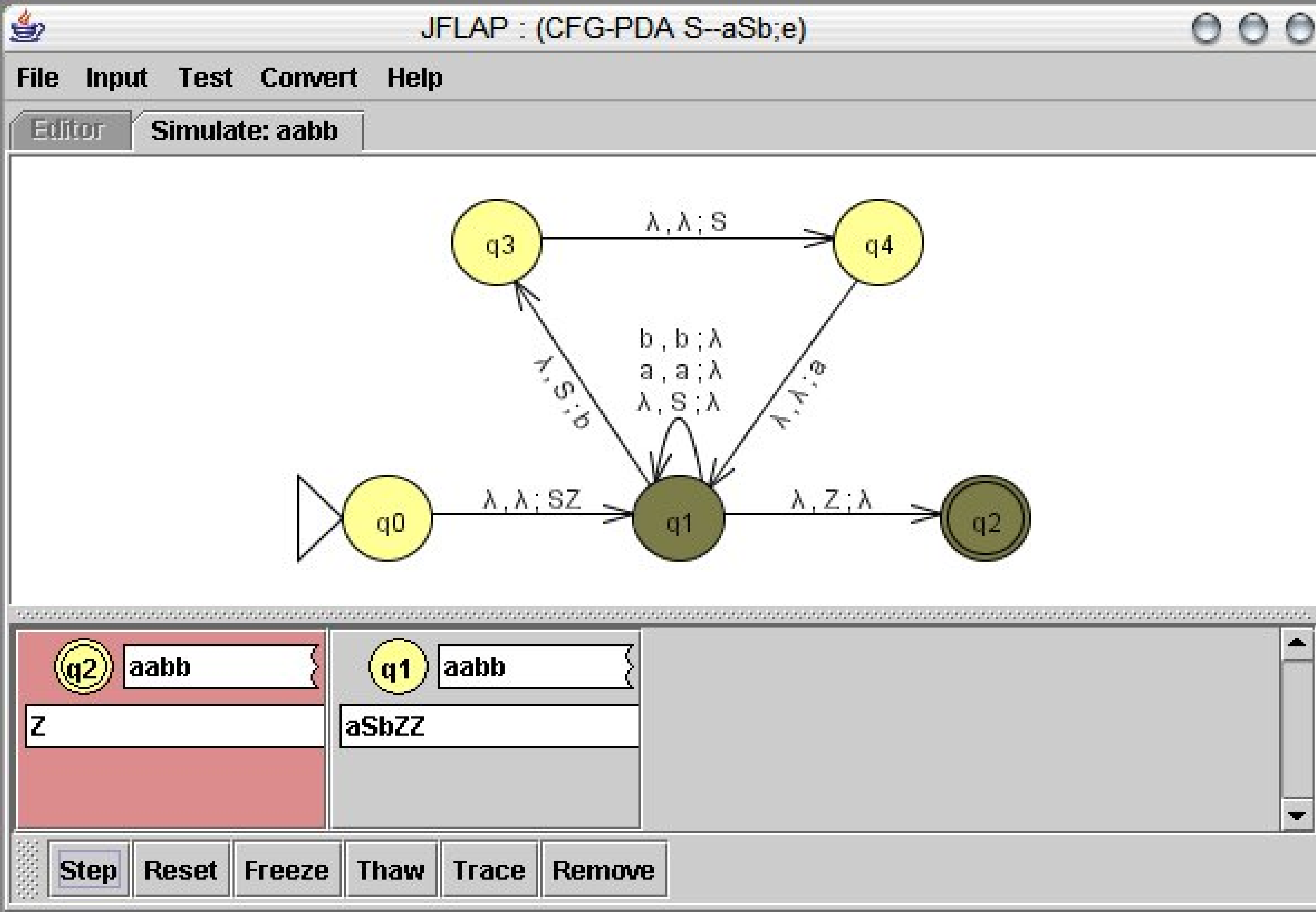

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
λ , λ ; S
q3
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
λ , λ ; SZ
λ , Z ; λ
q0
q1
q2
q0
aabb
Z
Step Reset Freeze Thaw Trace Remove

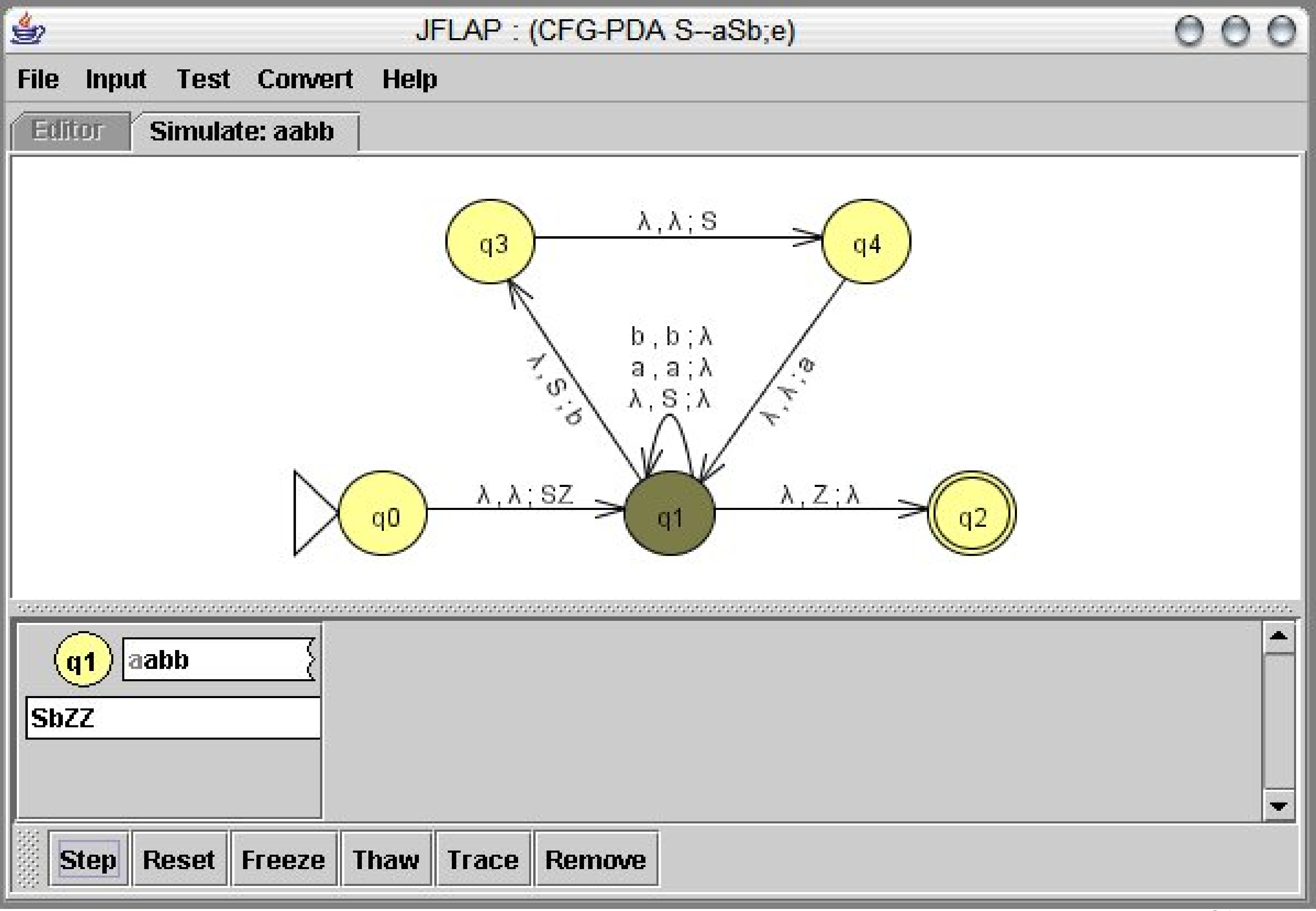

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
q3
λ , λ ; S
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
aabb
SZZ
Step
Reset
Freeze
Thaw
Trace
Remove

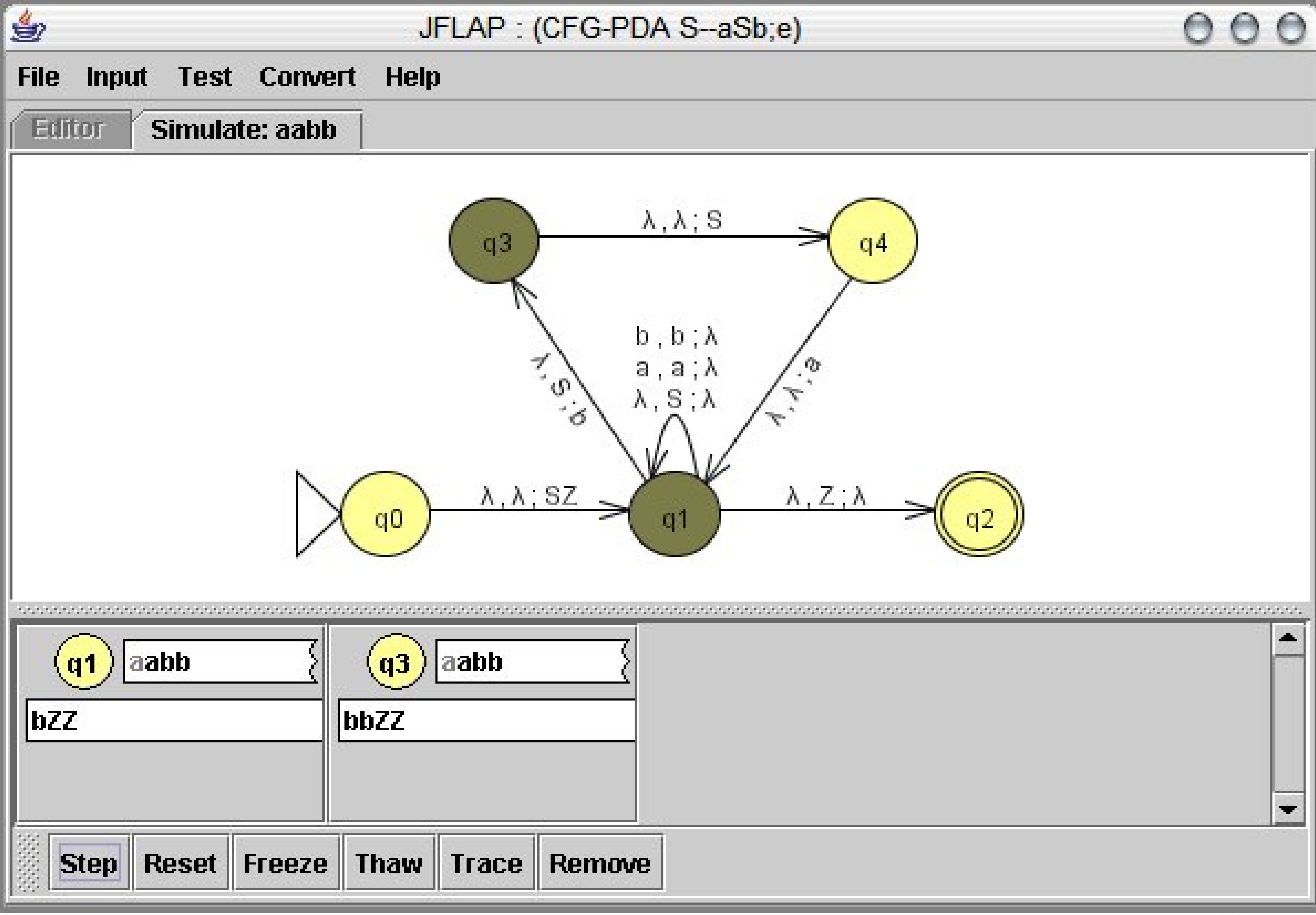


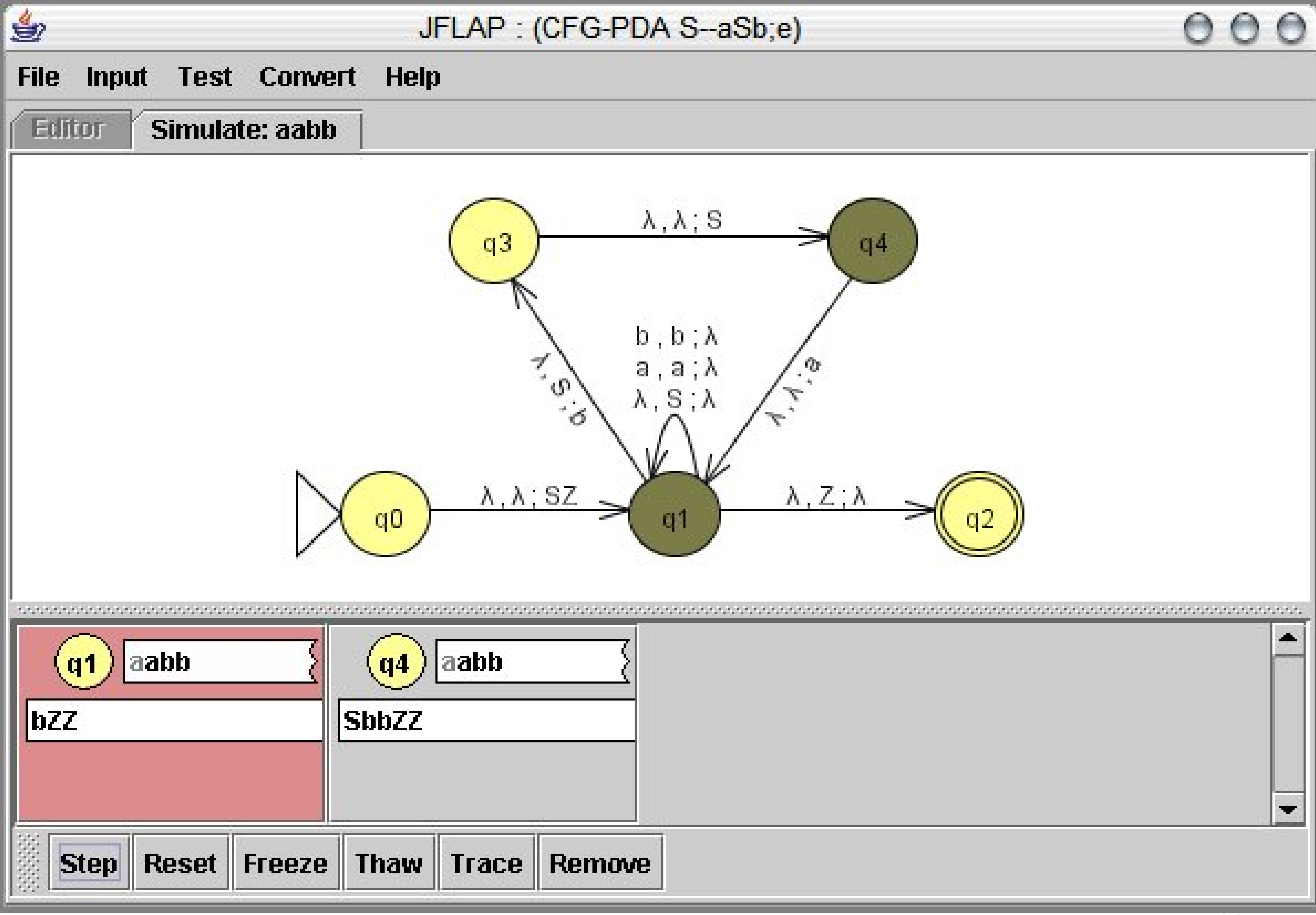
JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
λ , λ ; S
q3
q4
λ , S ; b
b , b ; λ
a , a ; λ
λ , S ; λ
λ , λ ; a
λ , λ ; SZ
q0
q1
λ , Z ; λ
q2
q1
aabb
ZZ
q3
aabb
bZZ
Step
Reset
Freeze
Thaw
Trace
Remove

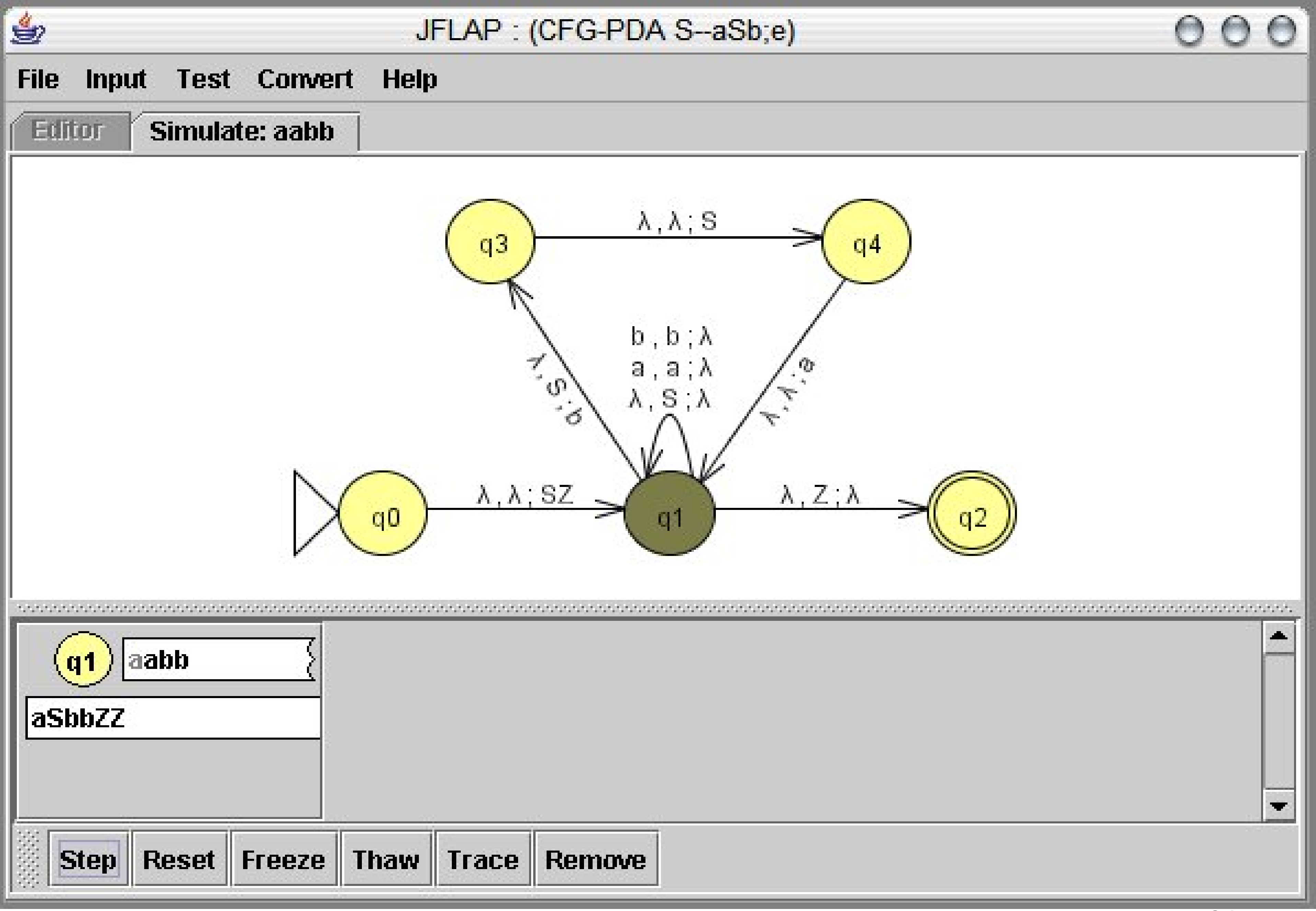

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
q3
λ , λ ; S
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q2
aabb
Z
q4
aabb
SbZZ
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
q3
λ , λ ; S
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q2
aabb
Z
q1
aabb
aSbZZ
Step
Reset
Freeze
Thaw
Trace
Remove

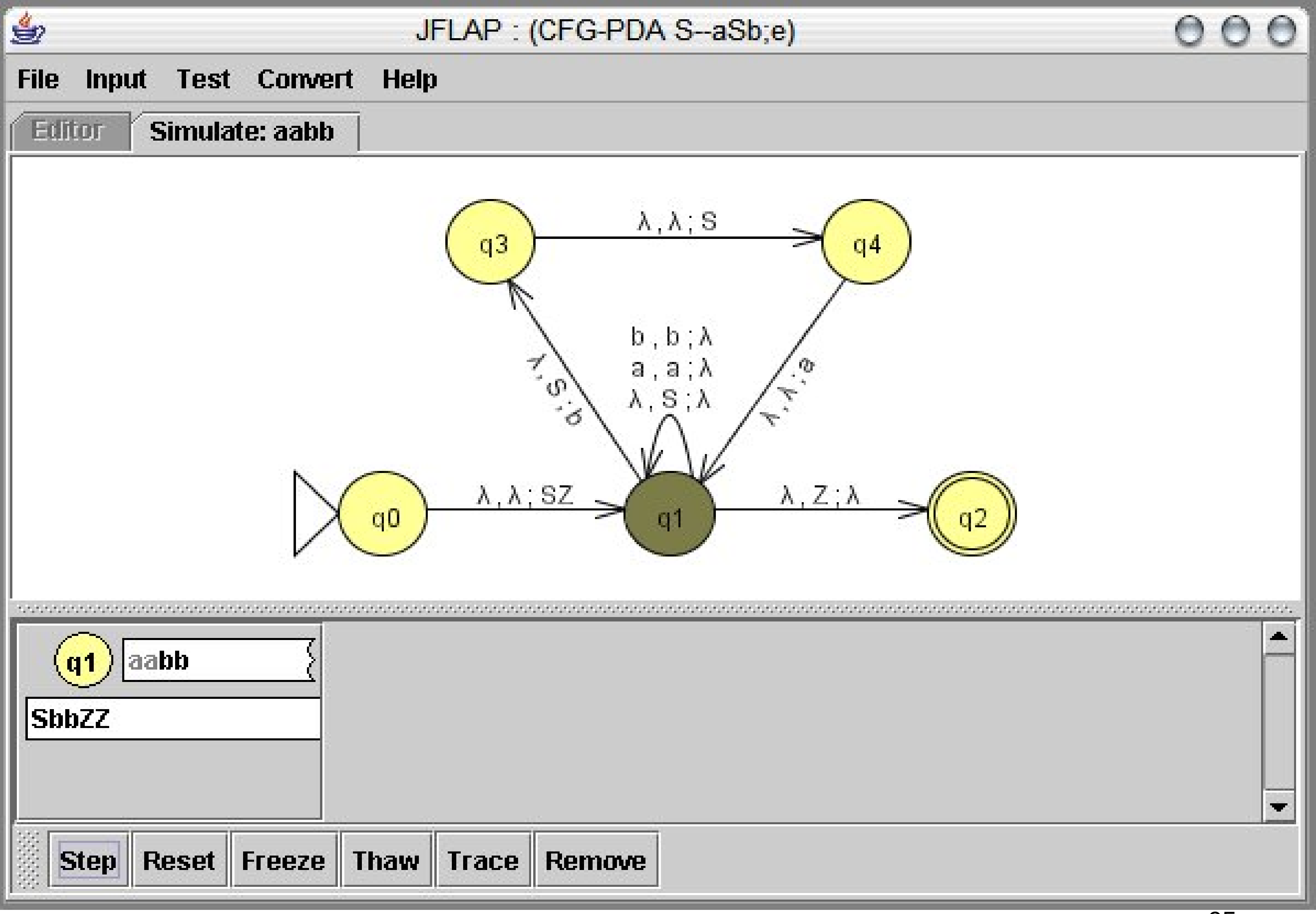

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
λ , λ ; S
q3
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
λ , λ ; SZ
λ , Z ; λ
q0
q1
q2
q1
aabb
SbZZ
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
q3
λ , λ ; S
q4
λ , S ; b
b , b ; λ
a , a ; λ
λ , S ; λ
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
aabb
bZZ
q3
aabb
bbZZ
Step
Reset
Freeze
Thaw
Trace
Remove

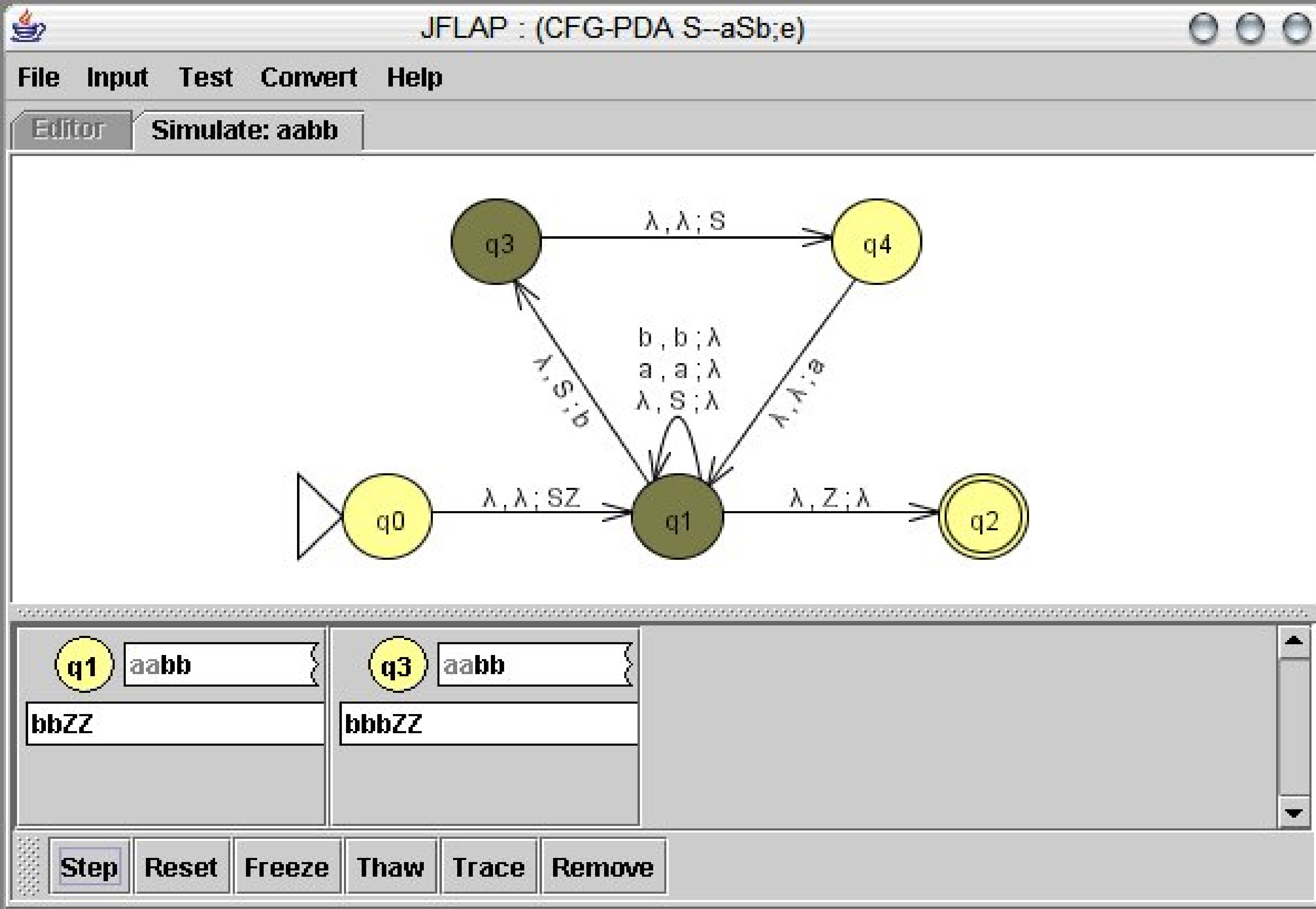

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
λ , λ ; S
q3
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
aabb
bZZ
q4
aabb
SbbZZ
Step
Reset
Freeze
Thaw
Trace
Remove

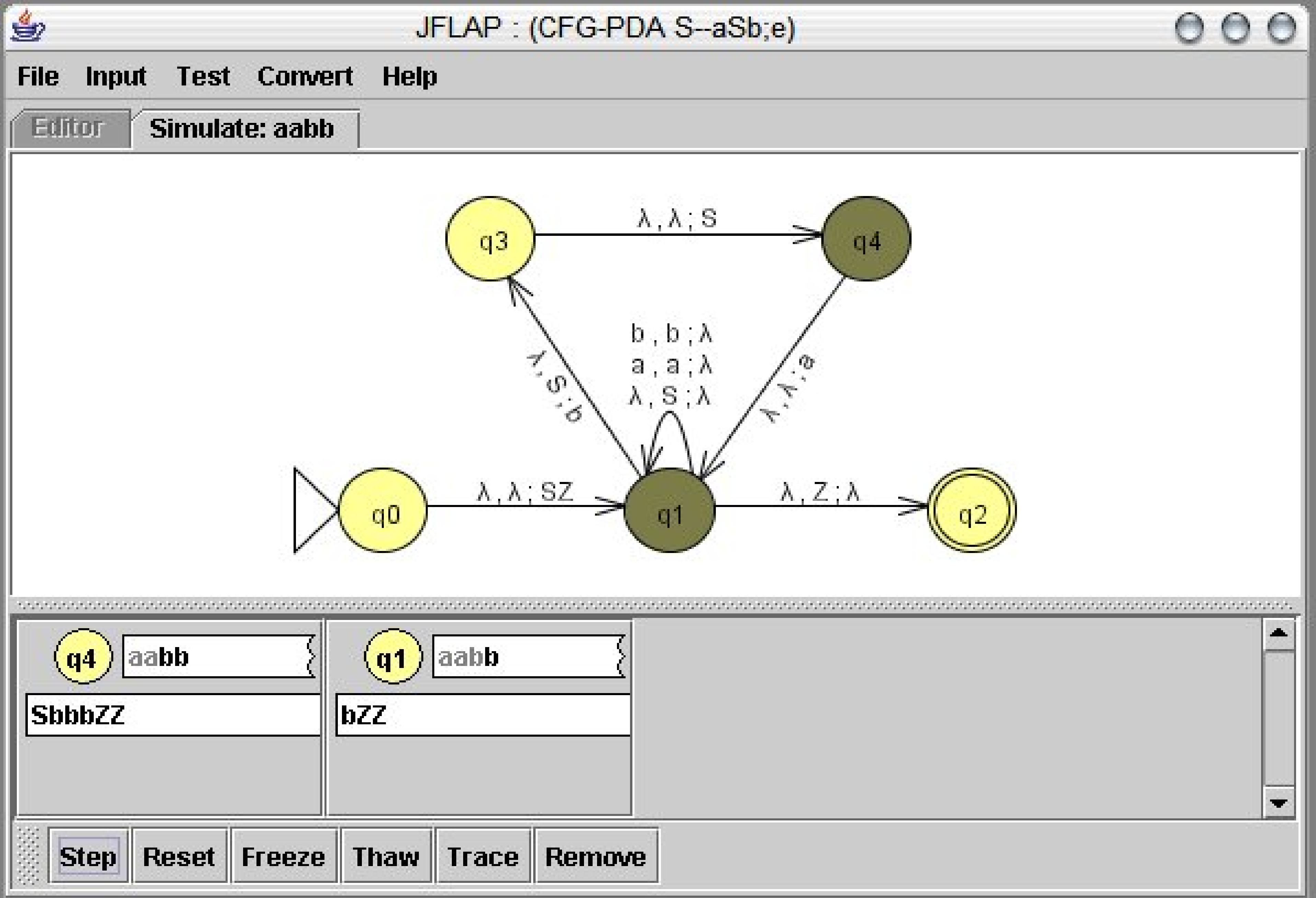


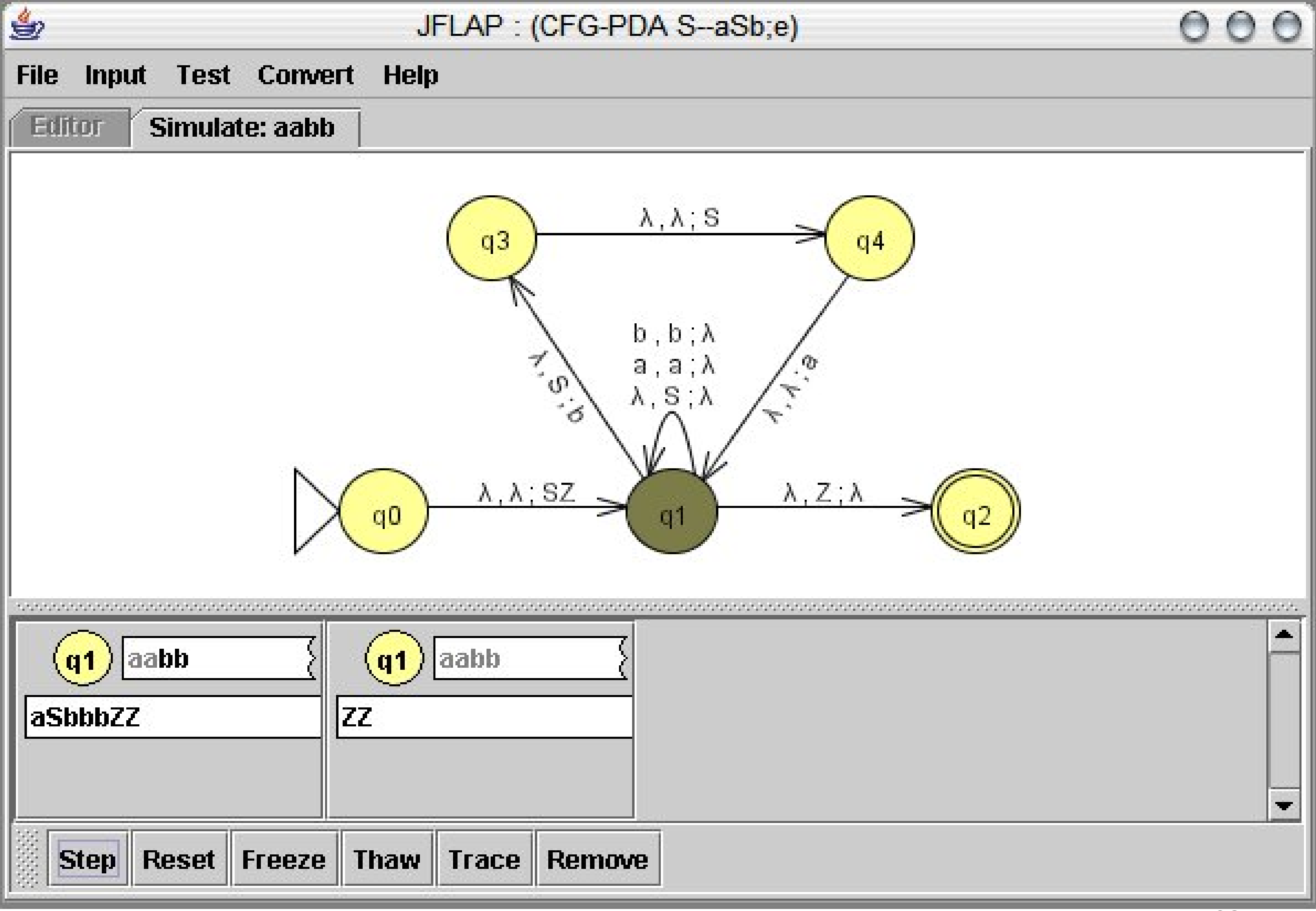
JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
q3
λ , λ ; S
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
aabb
aSbbZZ
Step
Reset
Freeze
Thaw
Trace
Remove



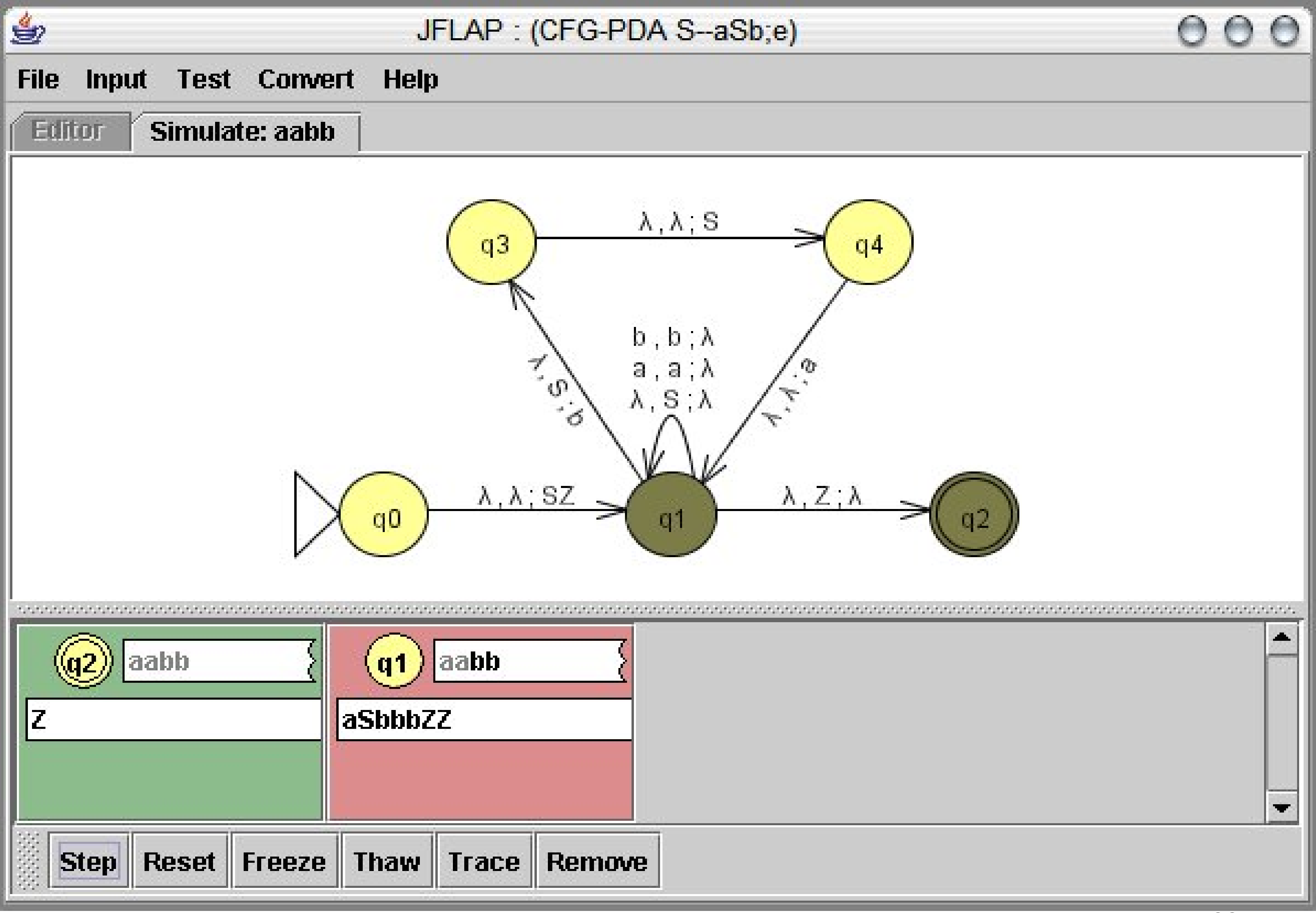
JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
q3
λ , λ ; S
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
aabb
SbbZZ
Step
Reset
Freeze
Thaw
Trace
Remove

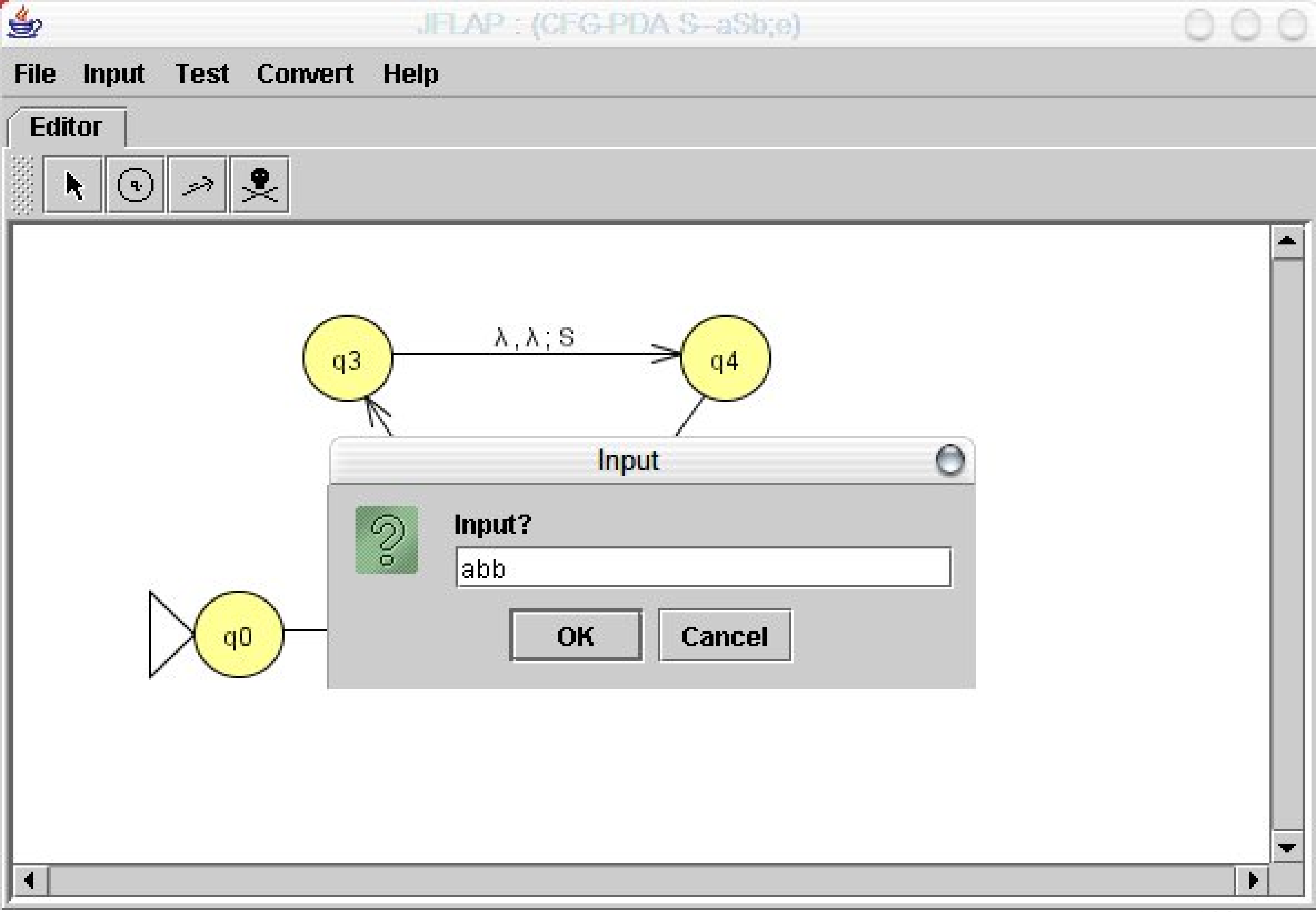

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
λ , λ ; S
q3
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
λ , λ ; SZ
λ , Z ; λ
q0
q1
q2
q1
aabb
bbZZ
q3
aabb
bbbZZ
Step
Reset
Freeze
Thaw
Trace
Remove

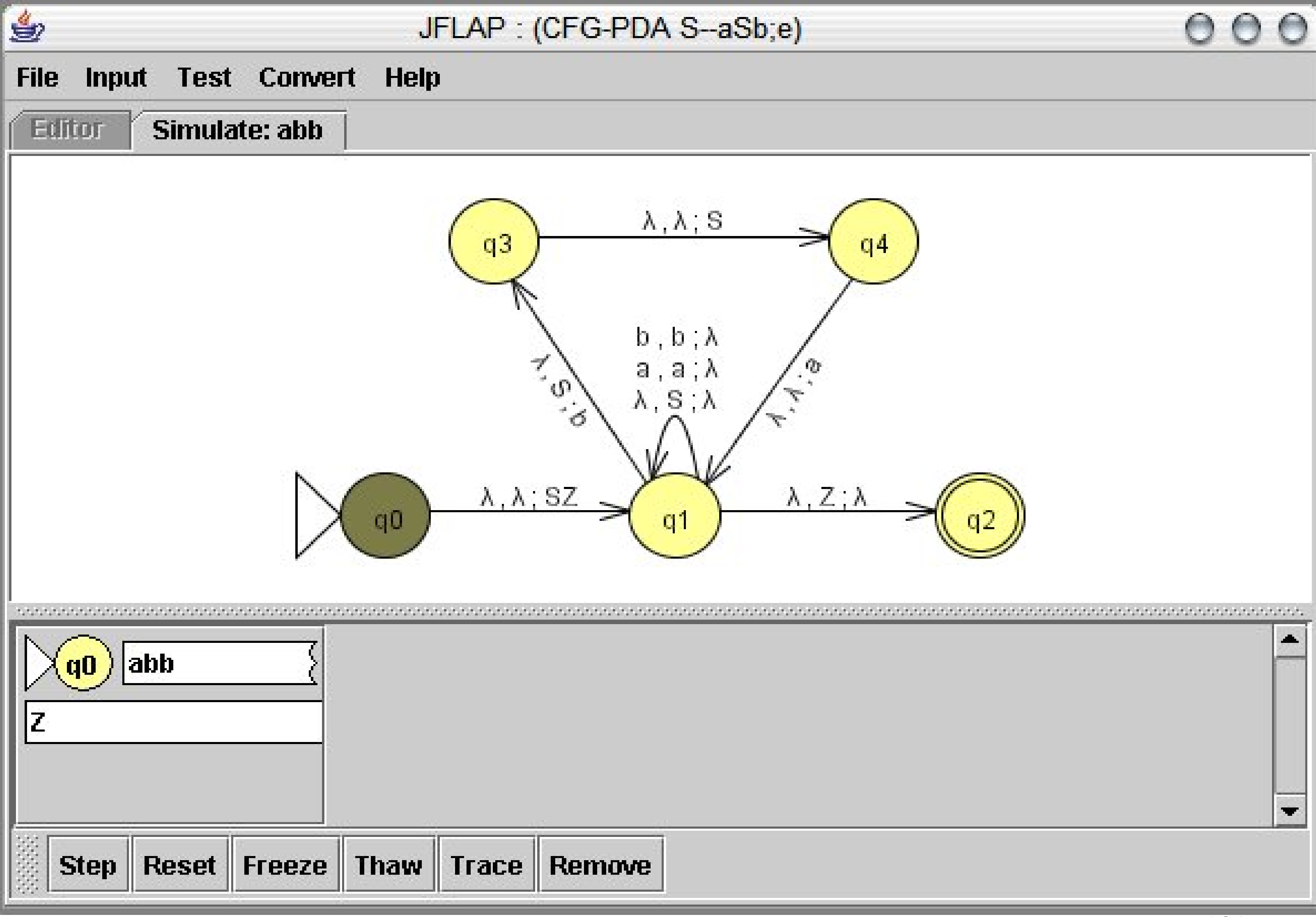

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
q3
q4
q0
q1
q2
λ , λ ; S
λ , S ; b
λ , λ ; a
b , b ; λ
a , a ; λ
λ , S ; λ
λ , λ ; SZ
λ , Z ; λ
q4
aabb
SbbbZZ
q1
aabb
bZZ
Step Reset Freeze Thaw Trace Remove



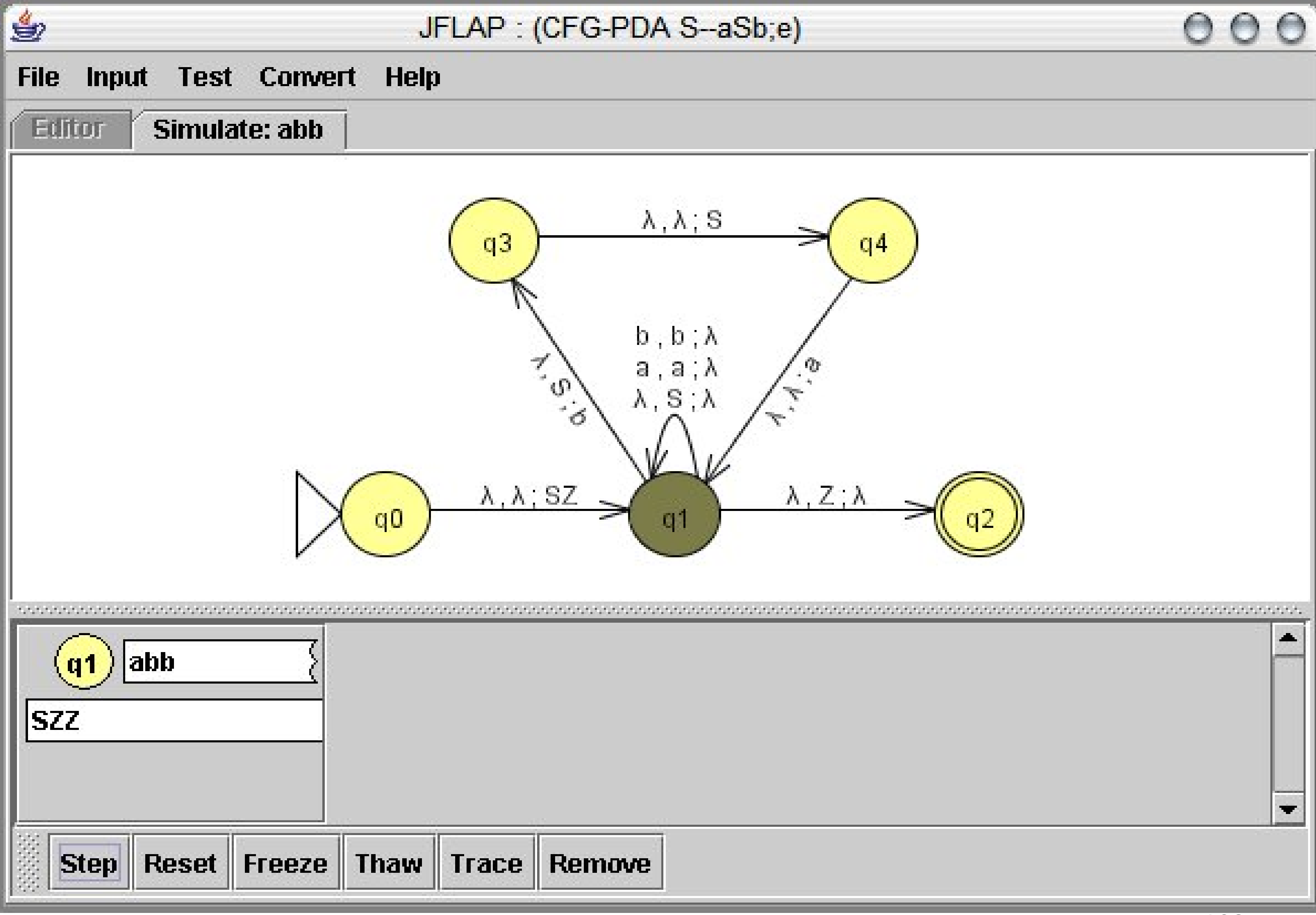
JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: aabb
q3
q4
λ , λ ; S
λ , S ; b
b , b ; λ
a , a ; λ
λ , S ; λ
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
aabb
aSbbbZZ
q1
aabb
ZZ
Step Reset Freeze Thaw Trace Remove

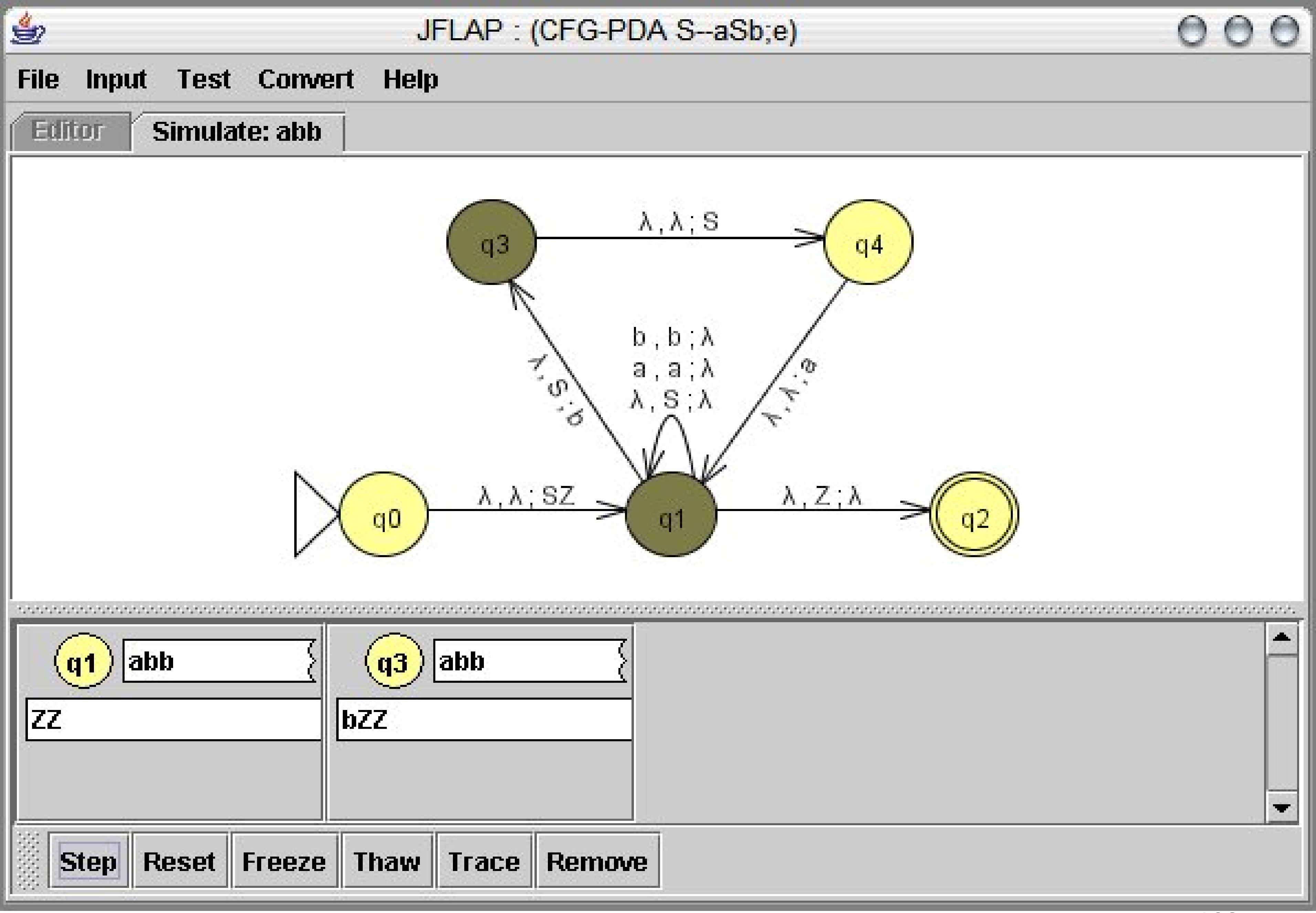

JFLAP : (CFG-PDA S--aSb;e)
File
Input
Test
Convert
Help
Editor
Simulate: aabb
q3
λ , λ ; S
q4
λ , S ; b
b , b ; λ
a , a ; λ
λ , S ; λ
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q2
aabb
Z
q1
aabb
aSbbbZZ
Step
Reset
Freeze
Thaw
Trace
Remove

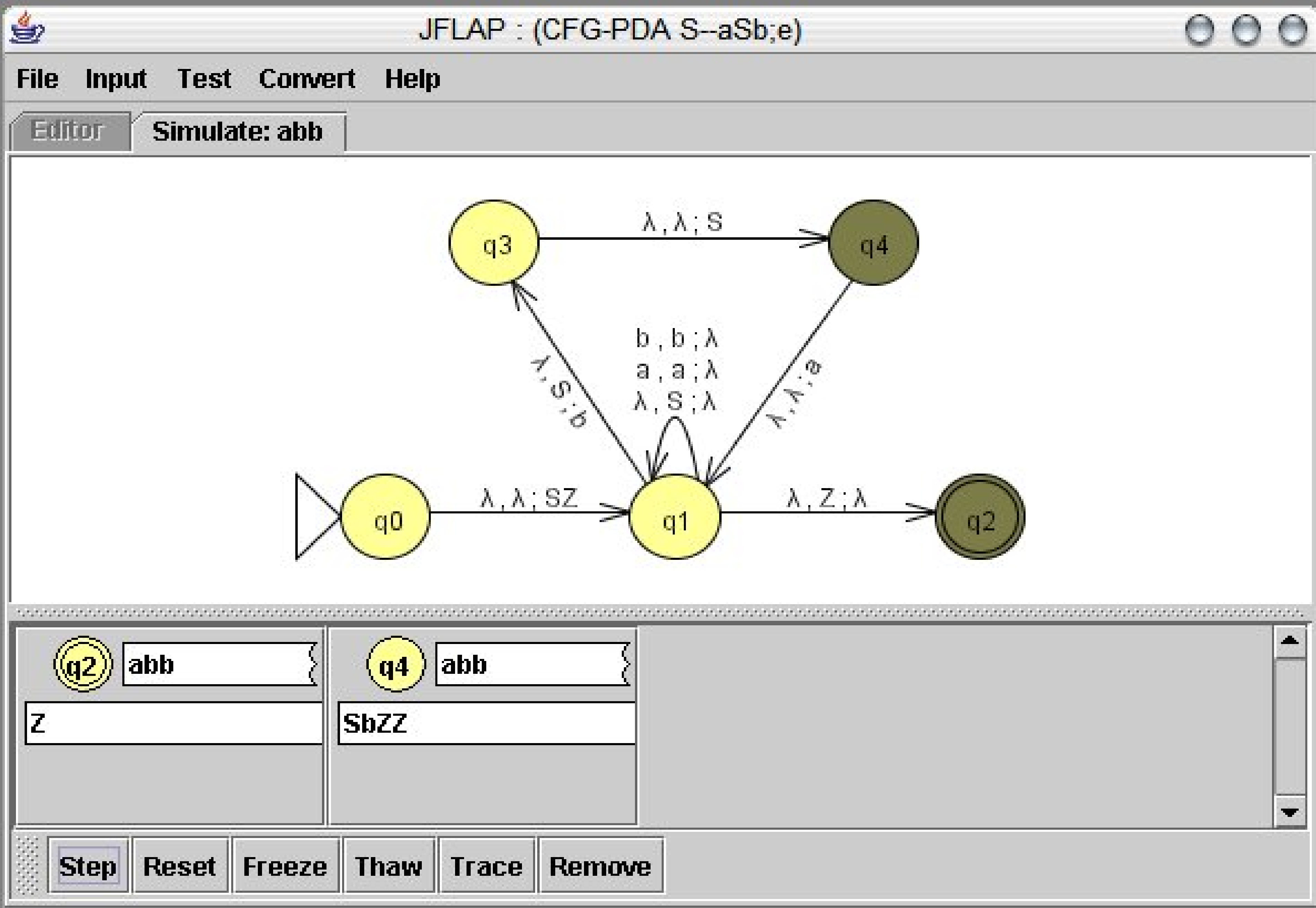

JFLAP : (CFG-PDA S→aSb;e)
File
Input
Test
Convert
Help
Editor
λ , λ ; S
q3
q4
q0
Input
Input?
abb
OK
Cancel

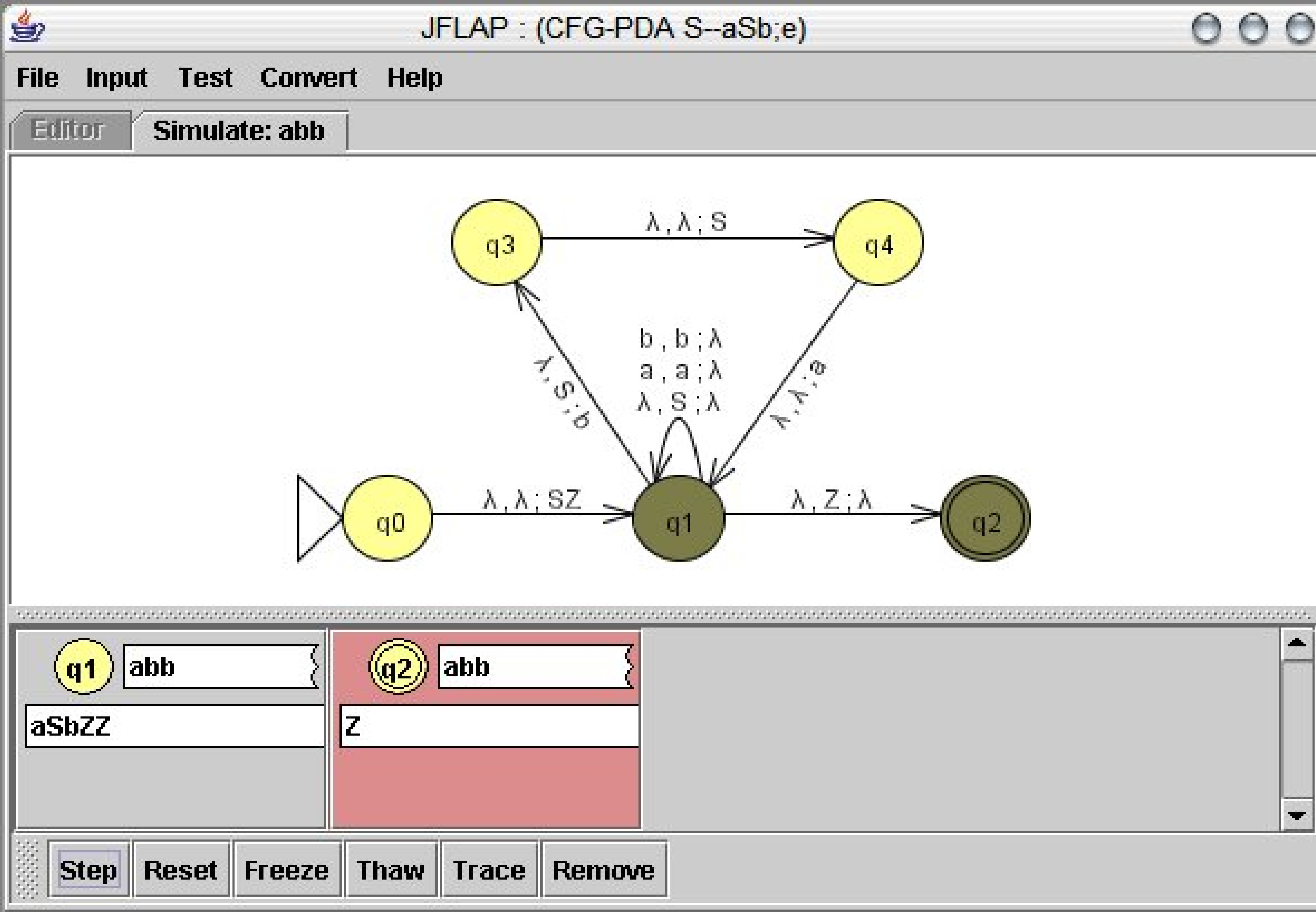


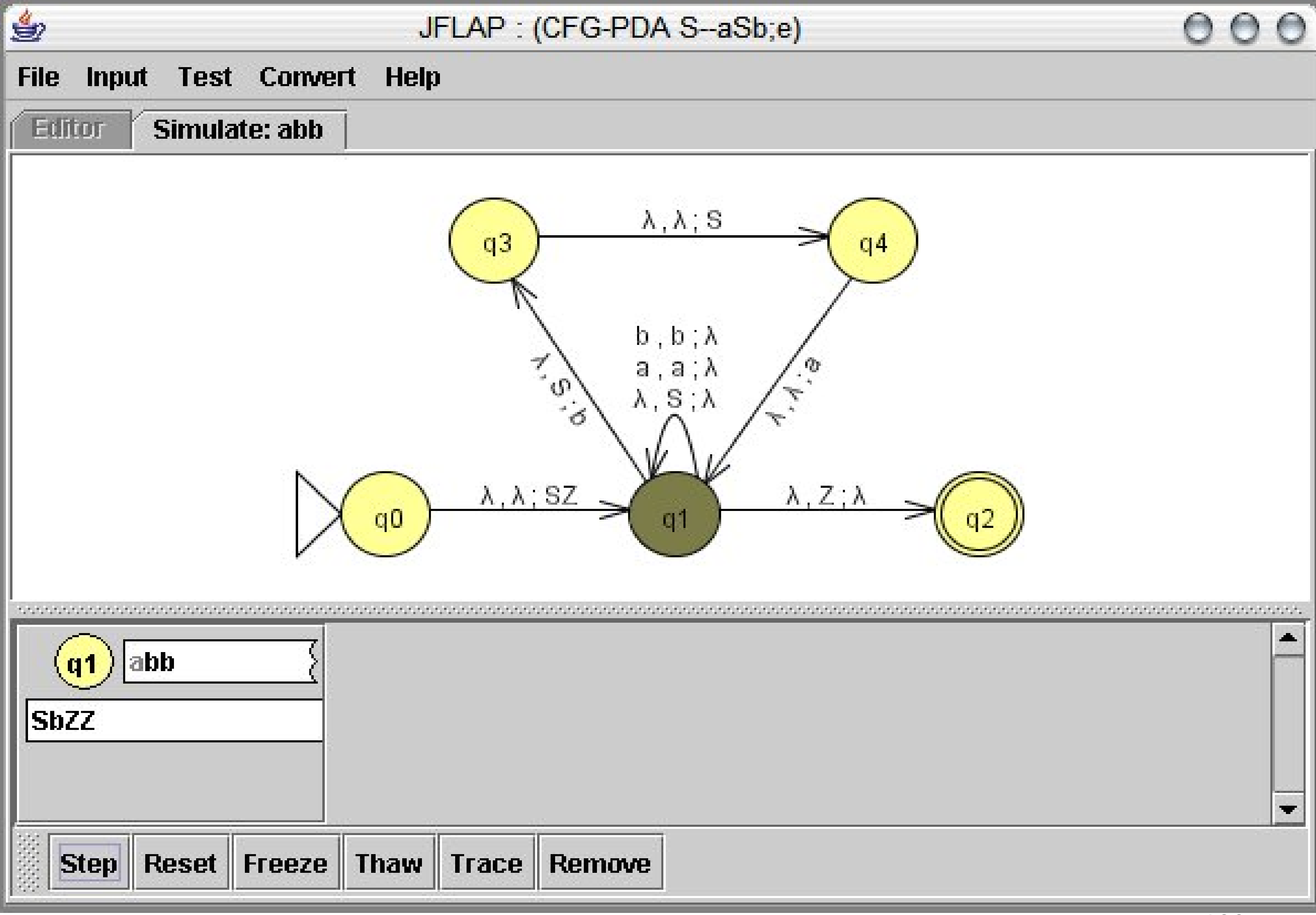
JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: abb
q3
λ , λ ; S
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q0
abb
Z
Step
Reset
Freeze
Thaw
Trace
Remove

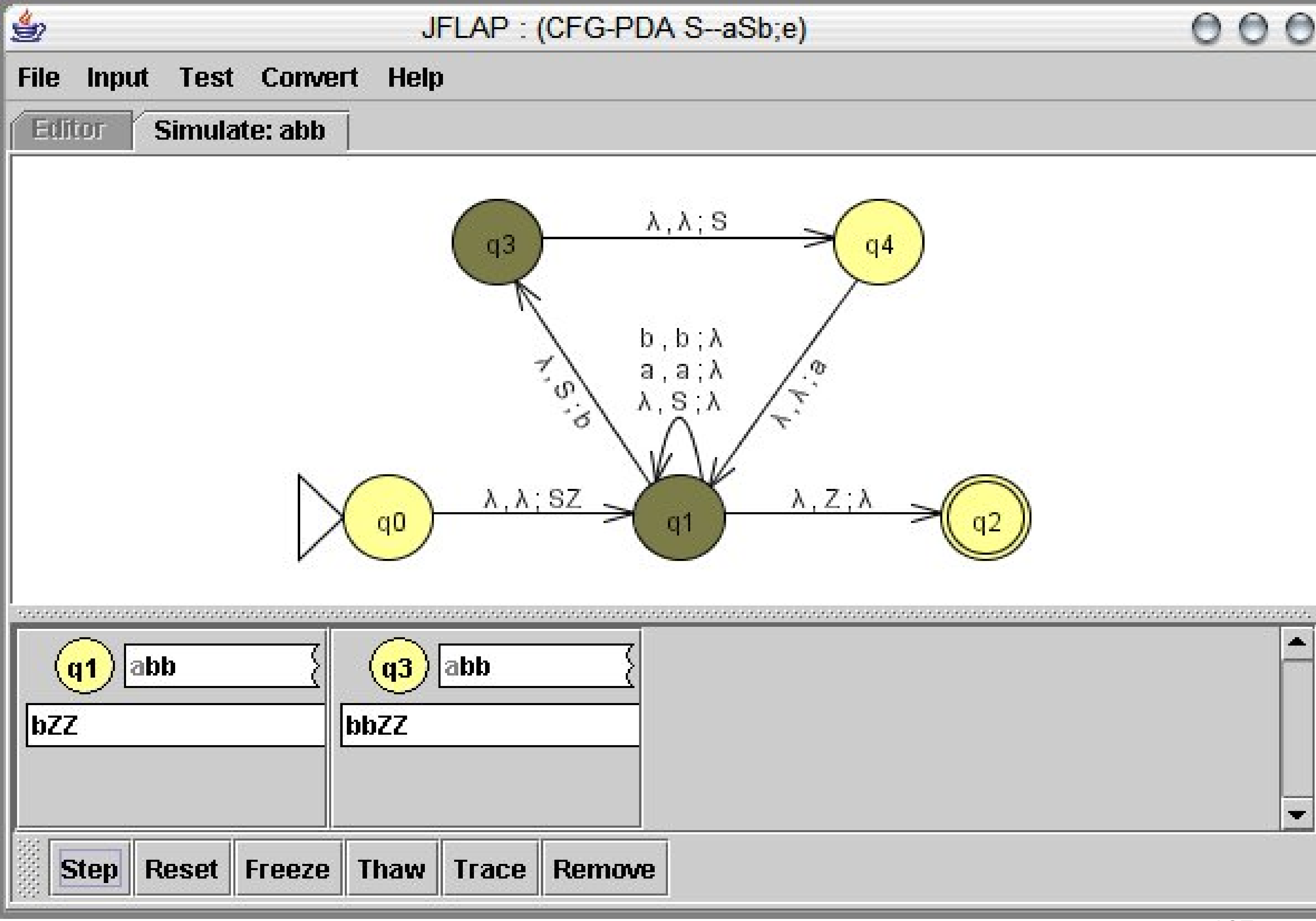

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: abb
q3
λ , λ ; S
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
abb
SZZ
Step
Reset
Freeze
Thaw
Trace
Remove

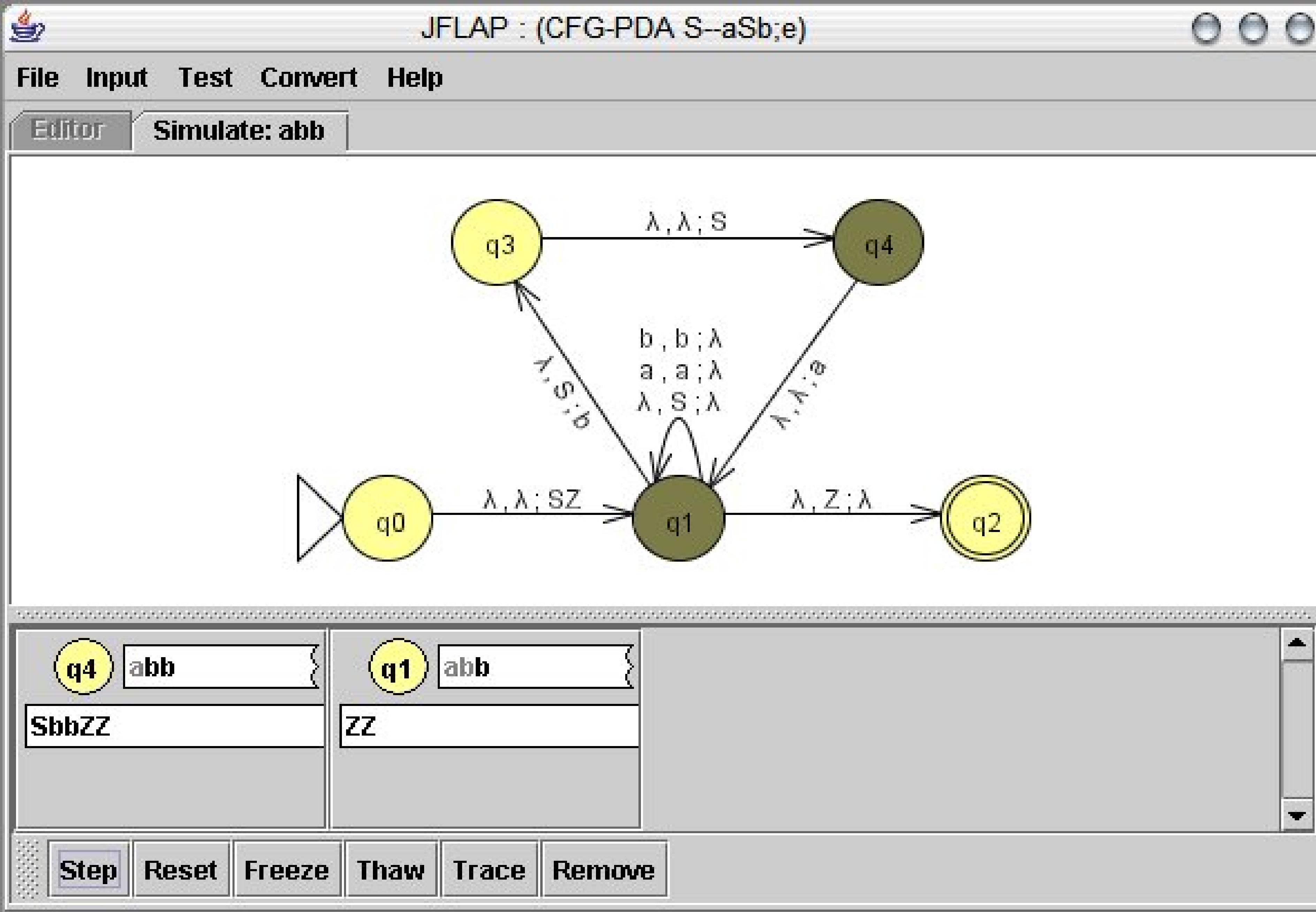

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: abb
λ , λ ; S
q3
q4
λ , S ; b
b , b ; λ
a , a ; λ
λ , S ; λ
λ , λ ; a
λ , λ ; SZ
λ , Z ; λ
q0
q1
q2
q1
abb
ZZ
q3
abb
bZZ
Step
Reset
Freeze
Thaw
Trace
Remove

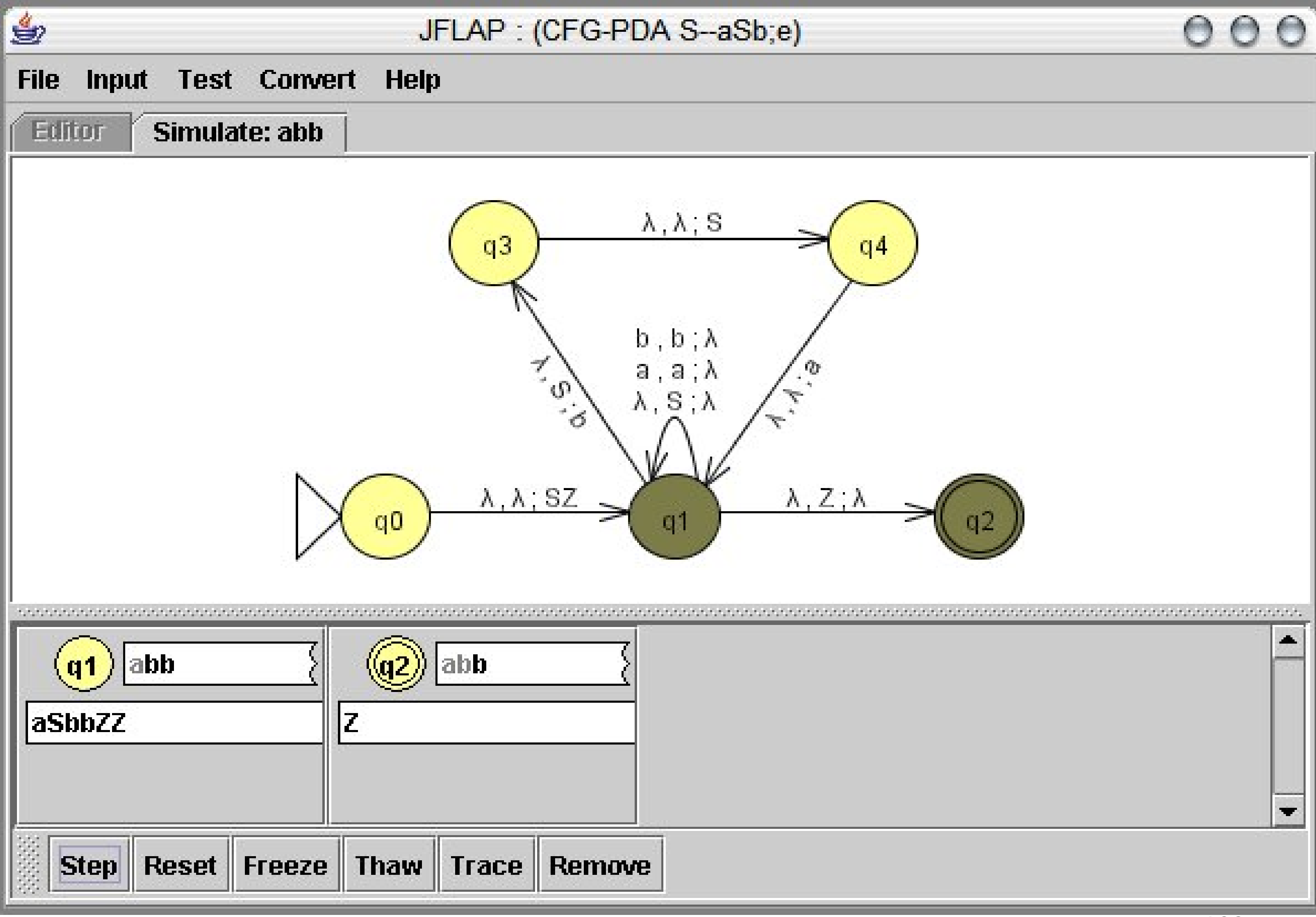

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: abb
q3
q4
q0
q1
q2
λ , λ ; S
λ , S ; b
b , b ; λ
a , a ; λ
λ , S ; λ
λ , λ ; a
λ , λ ; SZ
λ , Z ; λ
q2
abb
Z
q4
abb
SbZZ
Step
Reset
Freeze
Thaw
Trace
Remove

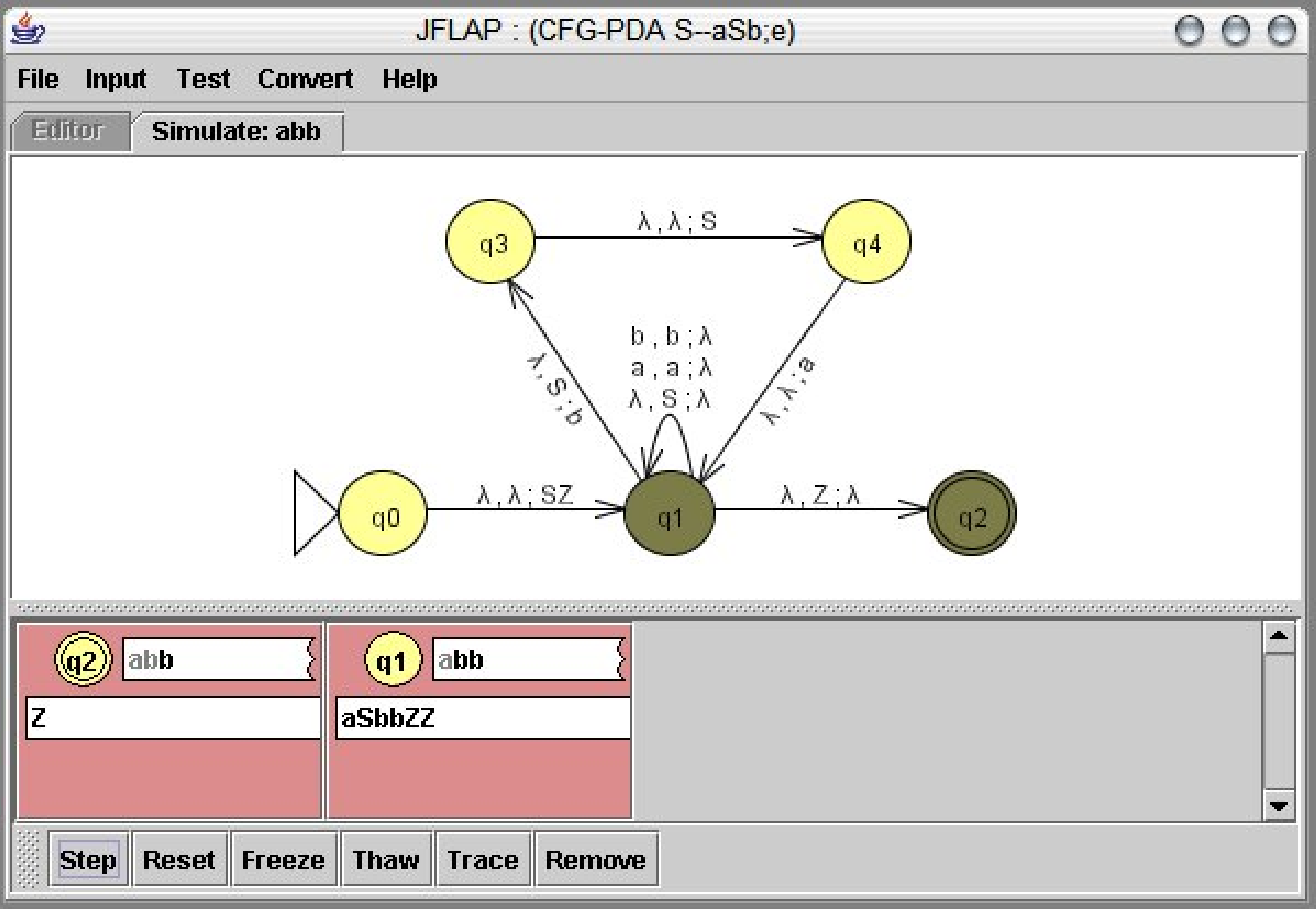

JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: abb
q3
q4
λ , λ ; S
λ , S ; b
b , b ; λ
a , a ; λ
λ , S ; λ
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
abb
aSbZZ
q2
abb
Z
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: abb
q3
λ , λ ; S
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
abb
SbZZ
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: abb
q3
λ , λ ; S
q4
λ , S ; b
b , b ; λ
a , a ; λ
λ , S ; λ
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
abb
bZZ
q3
abb
bbZZ
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: abb
λ , λ ; S
q3
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q4 abb
SbbZZ
q1 abb
ZZ
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: abb
q3
λ , λ ; S
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q1
abb
aSbbZZ
q2
abb
Z
Step
Reset
Freeze
Thaw
Trace
Remove


JFLAP : (CFG-PDA S--aSb;e)
File Input Test Convert Help
Editor
Simulate: abb
λ , λ ; S
q3
q4
b , b ; λ
a , a ; λ
λ , S ; λ
λ , S ; b
λ , λ ; a
q0
λ , λ ; SZ
q1
λ , Z ; λ
q2
q2
abb
Z
q1
abb
aSbbZZ
Step
Reset
Freeze
Thaw
Trace
Remove

Idea of PDA → CFG

- First, we simplify our task by modifying P slightly to give it the following three features:
  1. It has a single accept state, q_{accept}.
  2. It empties its stack before accepting.
  3. Each transition either pushes a symbol onto the stack (a *push* move) or pops one off the stack (a *pop* move), but does not do both at the same time.

PDA → CFG

- Suppose $P = \{Q, \Sigma, \Gamma, \Delta, q_0, \{q_{accept}\}\}$ to construct G.
- The variables of G are $\{A_{pq} \mid p, q \in Q\}$. A_{pq} generates all the strings that can take P from p with an empty stack to q with an empty stack.
- Two possibilities occur during P's computation on x. Either the symbol popped at the end is the symbol pushed at the beginning or not. First, simulated by Type 1 rules on next slide and the second by Type 2 rules.

PDA → CFG

- The start variable is $A_{q0qaccept}$. Now we describe G's rules.
  - [Type 1] For each p, q, r, s $\in$ Q, t $\in \Gamma$, and a, b $\in \Sigma_\varepsilon$, if ((p, a, ε), (r, t)) is in Δ and ((s, b, t), (q, ε)) is in Δ, put the rule $A_{pq} \rightarrow aA_{rs}b$ in G.
  - In other words, find pairs of transitions in the PDA such that the first transition in the pair pushes a symbol *t* and the second transition pops the same symbol *t*. Each such pair of transitions gives a Type 1 rule. The states p, q, r, s, and the symbols a, b are determined by looking at the transitions in the pair.

PDA → CFG

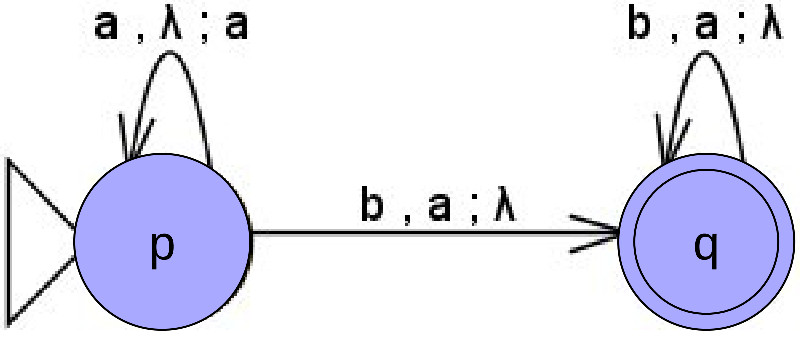
- [Type 2] For each p, q, $r \in Q$ put the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in G.
- [Type 3] Finally, for each $p \in Q$ put the rule $A_{pp} \rightarrow \varepsilon$ in G.

Example

- Let M be the PDA for $\{a^n b^n \mid n > 0\}$
  - Note that *n* cannot be 0, which makes the example a little simpler.

  M = {{p, q}, {a, b}, {a}, Δ, p, {q}}, where

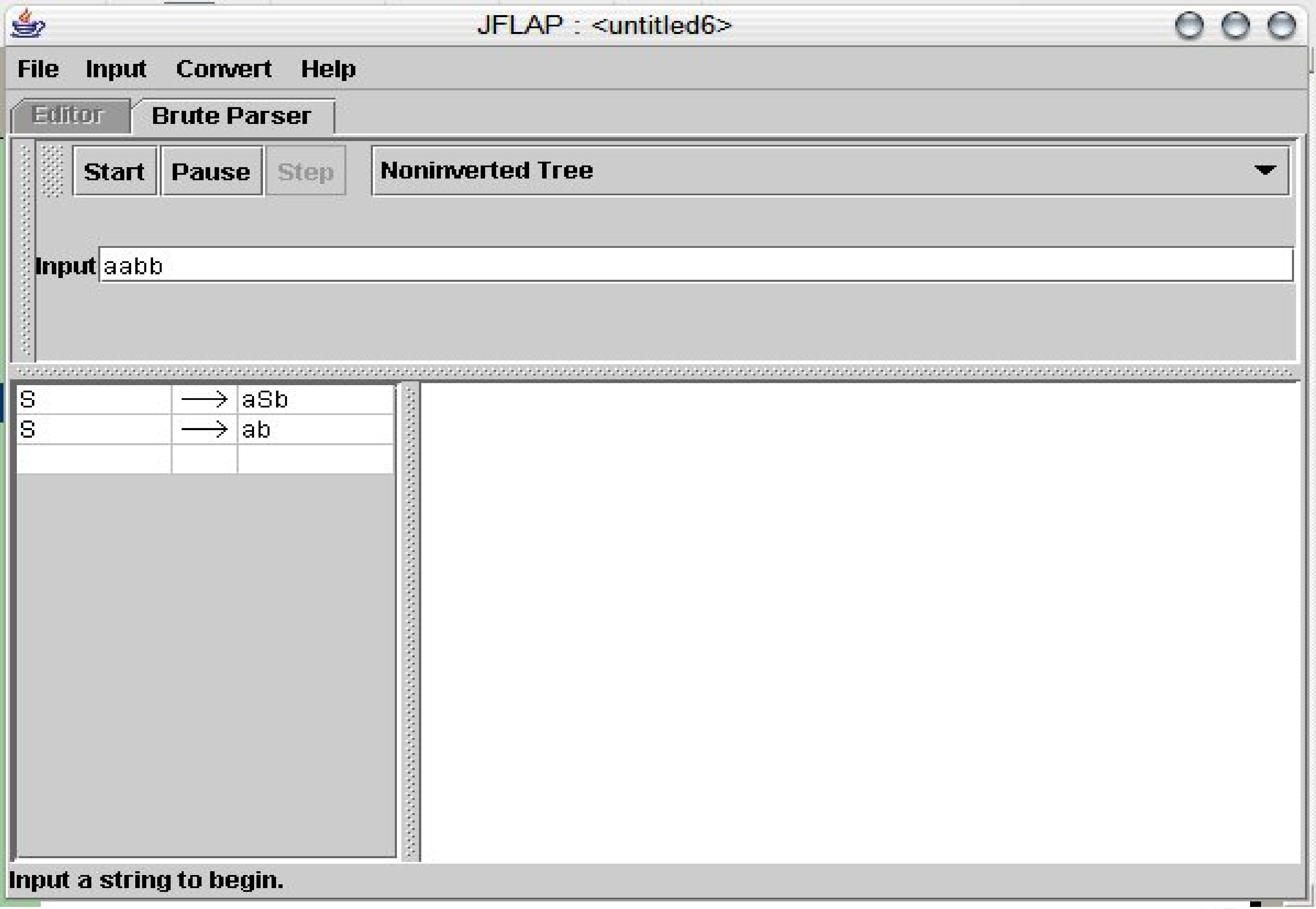
  Δ = {((p, a, ε),(p, a)),((p, b, a), (q, ε)),((q, b, a),(q, ε))}

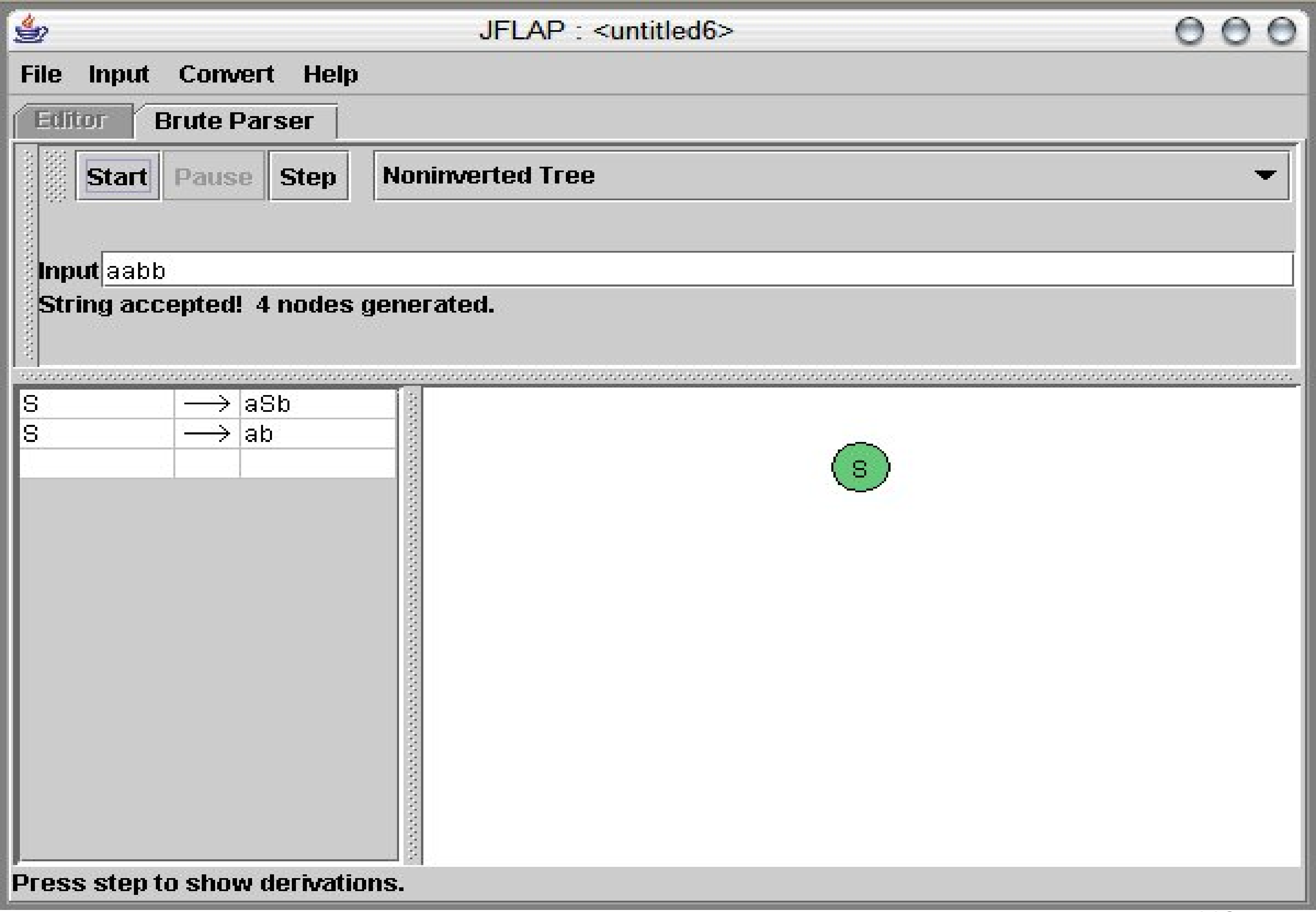


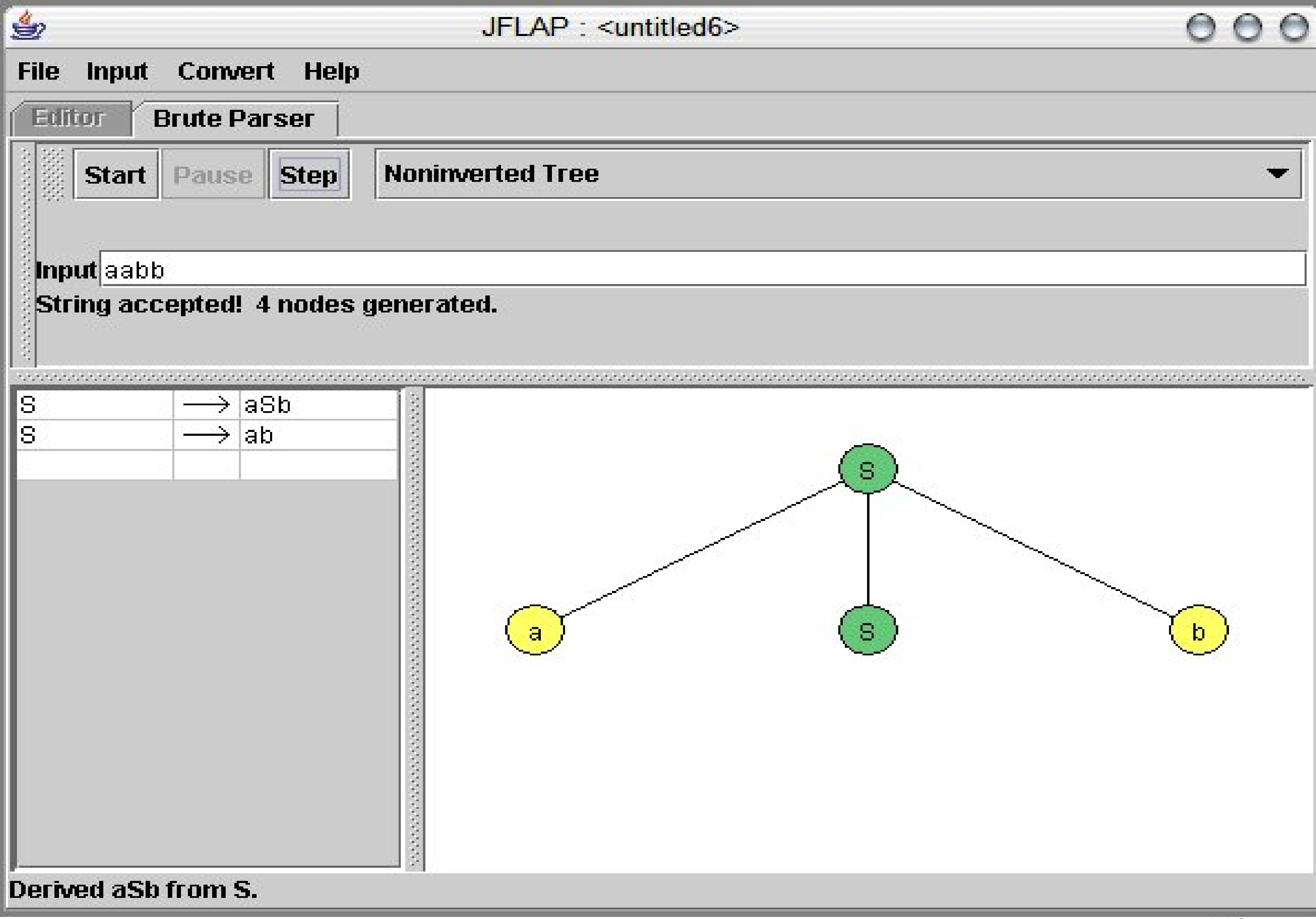
Example: cont'd.

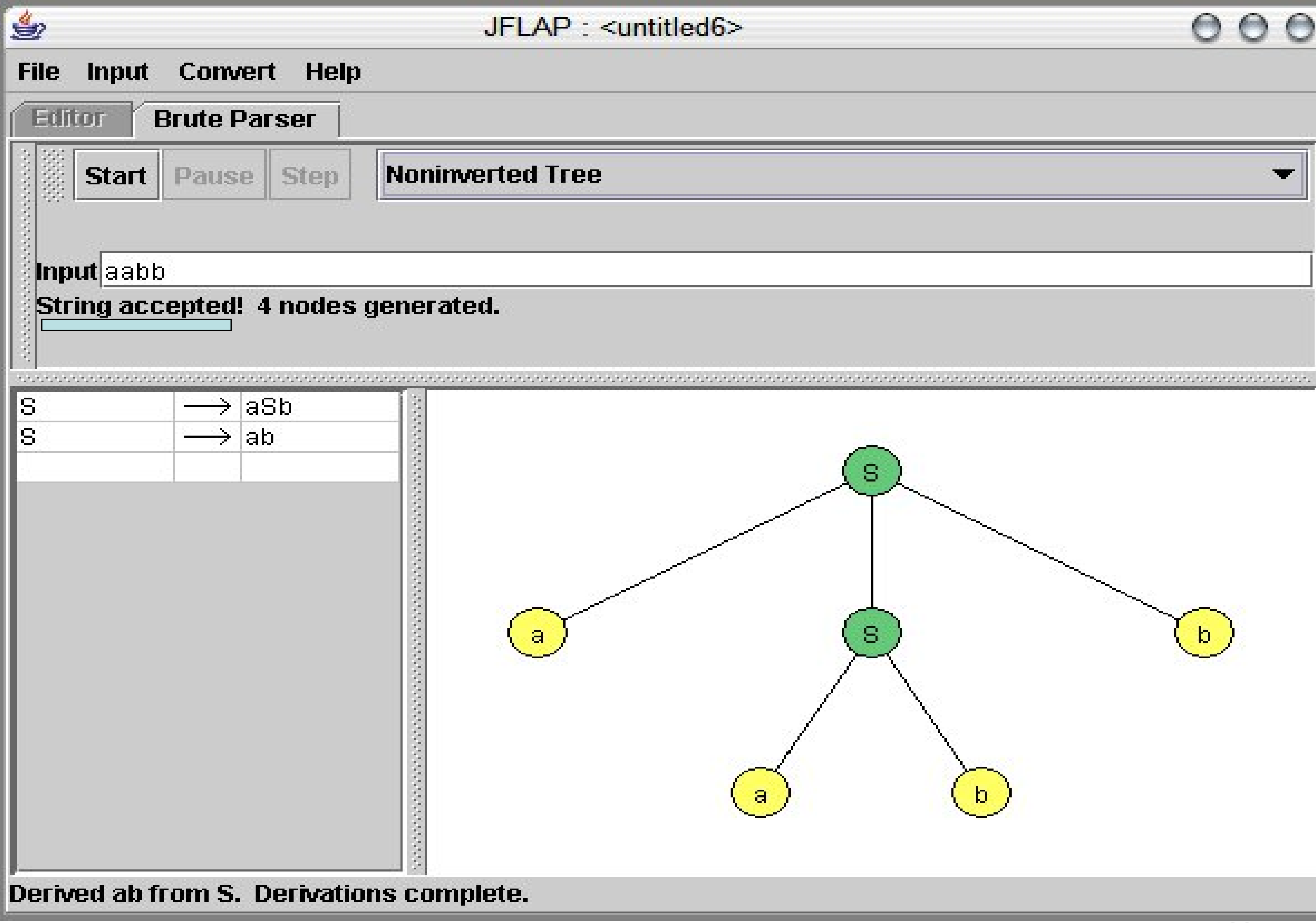
- CFG, $G = (V, \{a, b\}, A_{pq}, R)$ corresponding to M has $V = \{A_{pp}, A_{pq}, A_{qp}, A_{qq}\}$. R contains the following rules:
- Type I:
  - $A_{pq} \rightarrow aA_{pp}b$
  - $A_{pq} \rightarrow aA_{pq}b$
- Type II:
  - $A_{pp} \rightarrow A_{pp} A_{pp} \mid A_{pq} A_{qp}$
  - $A_{pq} \rightarrow A_{pp} A_{pq} \mid A_{pq} A_{qq}$
  - $A_{qp} \rightarrow A_{qp} A_{pp} \mid A_{qq} A_{qp}$
  - $A_{qq} \rightarrow A_{qp} A_{pq} \mid A_{qq} A_{qq}$
- Type III:
  - $A_{pp} \rightarrow \varepsilon$
  - $A_{qq} \rightarrow \varepsilon$

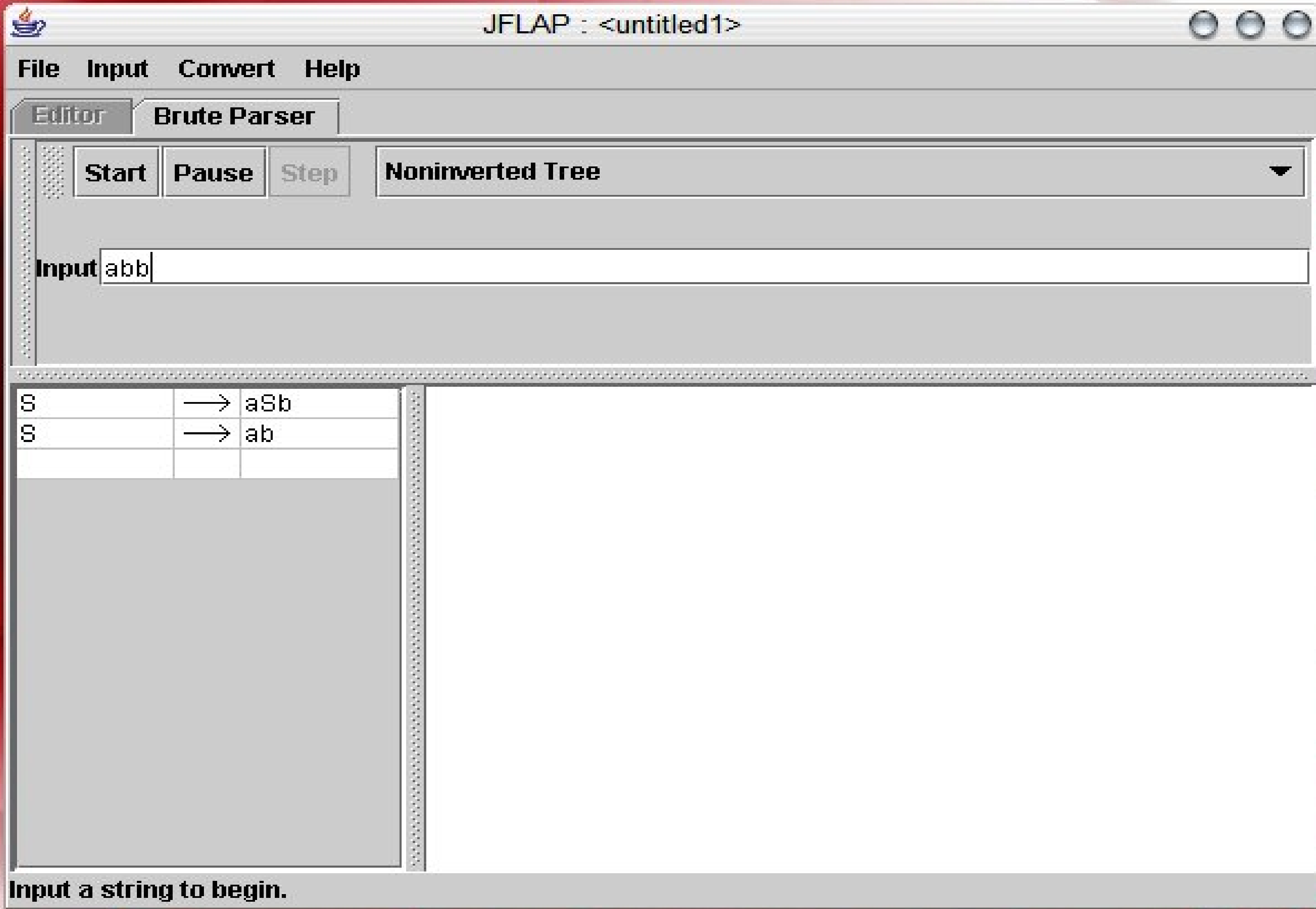
We can discard all rules containing the variables A_{qq} and A_{qp}. And we can also simplify the rules containing A_{pp} and get the grammar with just two rules $A_{pq} \rightarrow ab$ and $A_{pq} \rightarrow aA_{pq}b$.

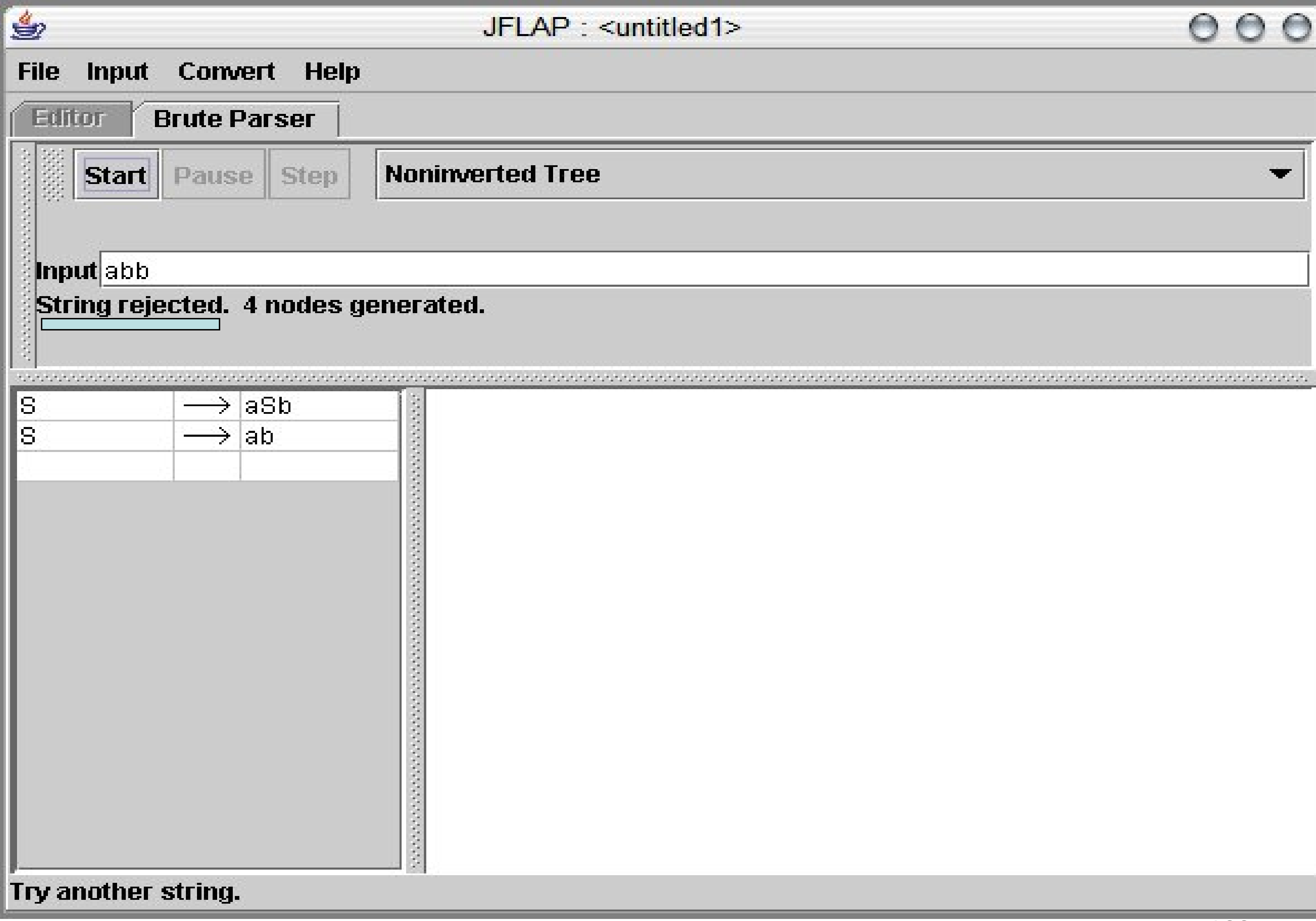

JFLAP : <untitled6>
File
Input
Convert
Help
Editor
Brute Parser
Start
Pause
Step
Noninverted Tree
Input
aabb
S → aSb
S → ab
Input a string to begin.


JFLAP : <untitled6>
File Input Convert Help
Editor
Brute Parser
Start
Pause
Step
Noninverted Tree
Input aabb
String accepted! 4 nodes generated.
S → aSb
S → ab
S
Press step to show derivations.


JFLAP : <untitled6>
File Input Convert Help
Editor
Brute Parser
Start
Pause
Step
Noninverted Tree
Input aabb
String accepted! 4 nodes generated.
S → aSb
S → ab
S
a
S
b
Derived aSb from S.


JFLAP : <untitled6>
File Input Convert Help
Editor
Brute Parser
Start
Pause
Step
Noninverted Tree
Input aabb
String accepted! 4 nodes generated.
S → aSb
S → ab
S
a
S
b
a
b
Derived ab from S. Derivations complete.


JFLAP : <untitled1>
File Input Convert Help
Editor
Brute Parser
Start
Pause
Step
Noninverted Tree
Input abb
S → aSb
S → ab
Input a string to begin.


JFLAP : <untitled1>
File Input Convert Help
Editor
Brute Parser
Start
Pause
Step
Noninverted Tree
Input abb
String rejected. 4 nodes generated.
S → aSb
S → ab
Try another string.

Are all languages context-free?

- Ans: No.
- How do we know this?
  - Ans: Cardinality arguments.
- Let C(CFG) = {G | G is a CFG}, C(CFG) is a countable set. Why?
- Let AL = { L | L is a subset of Σ^*}. AL is uncountable.

Pumping Lemma

- First technique to show that specific given languages are not context-free.
- Cardinality arguments show existence of languages that are not context-free.
- There is a big difference between the two!

Statement of Pumping Lemma

If A is an infinite context-free language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into five pieces $s = uvxyz$ satisfying the conditions.

1. For each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$, and
3. $|vxy| <= p$.

Statement of Pumping Lemma (contd.)

- When *s* is divided into *uvxyz*, condition 2 says -either *v* or *y* is not the empty string.
  - Otherwise the theorem would be trivially true.
- Condition 3 say - the pieces *v*, *x*, and *y* together have length at most *p*.
  - This condition is useful in proving that certain languages are not context free.

Proof of pumping lemma

Idea: If a sufficiently long string s is derived by a CFG, then there is a repeated nonterminal on a path in the parse tree.

One such repeated nonterminal must have a nonempty yield "on the sides" – v, y.

This nonterminal can be used to build infinitely many longer strings (and one shorter string, $i = 0$ case) derived by the CFG.

- Uses: Pigeon-hole principle.

Details of Proof of Pumping Lemma

- Let A be a CFL and let G be a CFG that generates it. We must show how any sufficiently long string *s* in A can be pumped and remain in A.
- Let *s* be a very long string in A.
- Since *s* is in A, it is derivable from G and so has a parse tree. The parse tree for *s* must be very tall because *s* is very long.

Details of Proof of Pumping Lemma (contd)

How long does *s* have to be?

– Let *b* be the maximum number of symbols in the right-hand side of a rule.

– Assume $b \geq 2$.

- A parse tree using this grammar can have no more than *b* children.
- At least b leaves are 1 step from the start variable; at most b^2 leaves are at most 2 steps from the start variable; at most b^h leaves are at most *h* steps from the start variable.

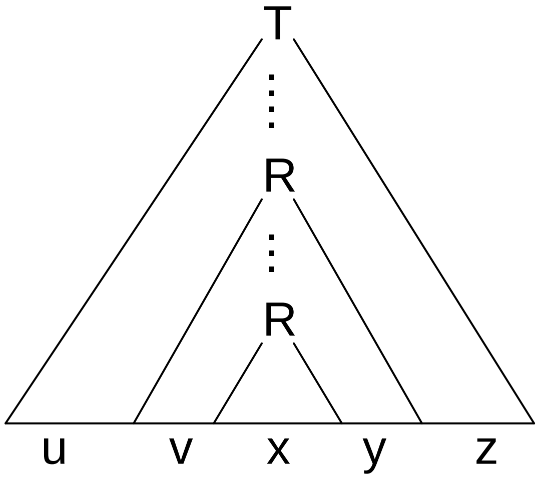
Details of Proof of Pumping Lemma (contd)

- So, if the height of the parse tree is at most h, the length of the string generated is at most b^h
- Let $|V|$ = number of nonterminals in G
- Set $p = b^{|V|+2}$
- Because $b \geq 2$, we know that $p > b^{|V|+1}$, so a parse tree for any string in A of length at least p requires height at least $|V| + 2$.

• Therefore, let s in A be of length at least p.

Details of Proof of Pumping Lemma (contd)

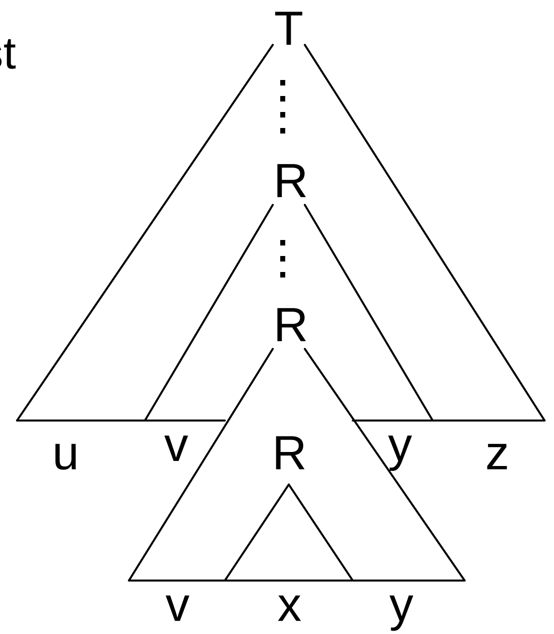
- The parse tree must contain some long path from the start variable at the root of the tree to one of the terminal symbol at a leaf. On this long path some variable symbol R must repeat because of the pigeonhole principle.



We start with a smallest parse tree which yield s

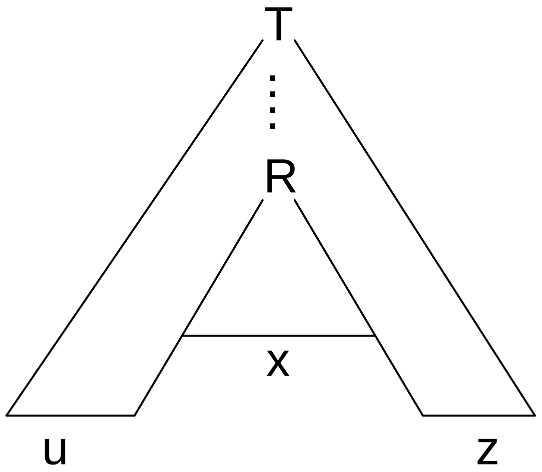
Details of Proof of Pumping Lemma (contd)

- This repetition of R allows us to replace the subtree under the 2nd occurrence of R with the subtree under the 1st occurrence of R and still get a legal parse tree. Therefore we may cut s into 5 pieces uvxyz as the figure indicates and we may repeat the 2nd and 4th pieces and obtain a sting in the language.



Details of Proof of Pumping Lemma (contd)

- In other words, uv^ixy^iz is in A for any $i \geq 0$.
  even if $i = 0$.



Some Applications of Pumping Lemma

- The following languages are not context-free.
    1. $\{a^n b^n c^n \mid n \geq 0\}$.
    2. $\{a^{n^2} \mid n \geq 0\}$.
    3. $\{w \text{ in } \{a,b,c\}^* \mid w$ has equal a's, b's and c's$\}$.

Example: CFL $L=\{a^n b^n c^n \mid n \geq 0\}$

- L is not context free.
- To show this, assume L is a CFL. L is infinite.
- Let $w = a^p b^p c^p$, p is the pumping length

$$\underbrace{a \dots a}_{p}\underbrace{b \dots b}_{p}\underbrace{c \dots c}_{p}$$

$|vy| \geq 0$

$|vxy| \leq p$

$|w| = 3p \geq p$

Example (contd.)

Case 1:

– Both *v* and *y* contain only one type of alphabet symbols, v does not contain both *a*'s and *b*'s or both *b*'s and *c*'s and the same holds for *y*. Two possibilities are shown below.

$$a\underbrace{\ldots\ a}_{\text{v}}\underbrace{b\ldots\ b}_{\text{y/v}}\underbrace{c\ldots}_{\text{y}}\ c$$

– In this case the string uv^2xy^2z cannot contain equal number of *a*'s, *b*'s and *c*'s. Therefore, $uv^2xy^2z \notin$ L.

Example (contd.)

Case 2:

- Either v or y contain more than one type of alphabet symbols. Two possibilities are shown below.

$$a\underbrace{\ldots a \ldots a\, b}_{\text{v}}\;\underbrace{b\, b \ldots}_{\text{y/v}}\;\underbrace{b\, c \ldots c}_{\text{y}}$$

- In this case the string uv^2xy^2z may contain equal number of the three alphabet symbols but won't contain them in the correct order.
- Therefore, $uv^2xy^2z \notin$ L.

CFL is not closed under intersection and complement

- Let Σ = {a,b,c}. L = {w over Σ | w has equal a's and b's}. L' = {w over Σ | w has equal b's and c's}. L, L' are CFLs.
- L intersect L' = {w over Σ | w has equal a's, b's and c's}, which is not a CFL.
- Because of closure under Union and DeMorgan's law, CFLs are not closed under complement either.
- CFLs are closed under intersection with regular languages.

Tips of the trade -- Do not forget!

Closure properties can be used effectively for:

(1) Shortening cumbersome Pumping lemma arguments.

Example: {w in {a, b, c}* | w has equal a's, b's, and c's}.

(2) For showing that certain languages are context-free.

Example: {w in {a, b, c}* | w has equal a's and b's or equal b's and c's }.

Reference

- www.cs.uh.edu/~rmverma by Dr. Rakesh Verma.

Answer of page 80

- $L(M)=\{ww^R\}$

Homework 3

- Exercise 2.1 on page 128
- Exercise 2.2
- Exercise 2.4 b, c, e, f
- Exercise 2.11
- Exercise 2.14
- Problem 2.30 a, b