PushDown Automata

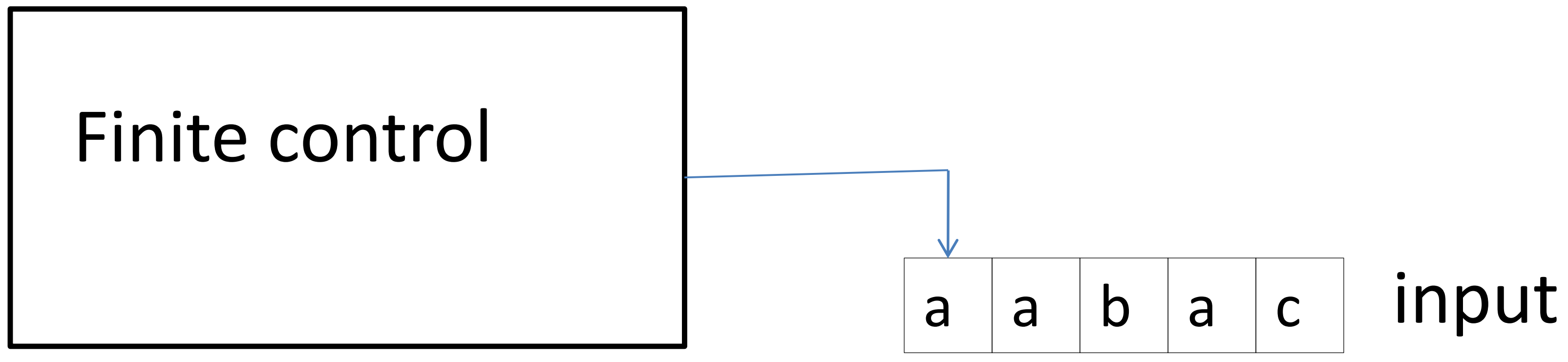
Introduction and Motivation

In this lecture we introduce ***Pushdown Automata***, a computational model equivalent to context free languages.

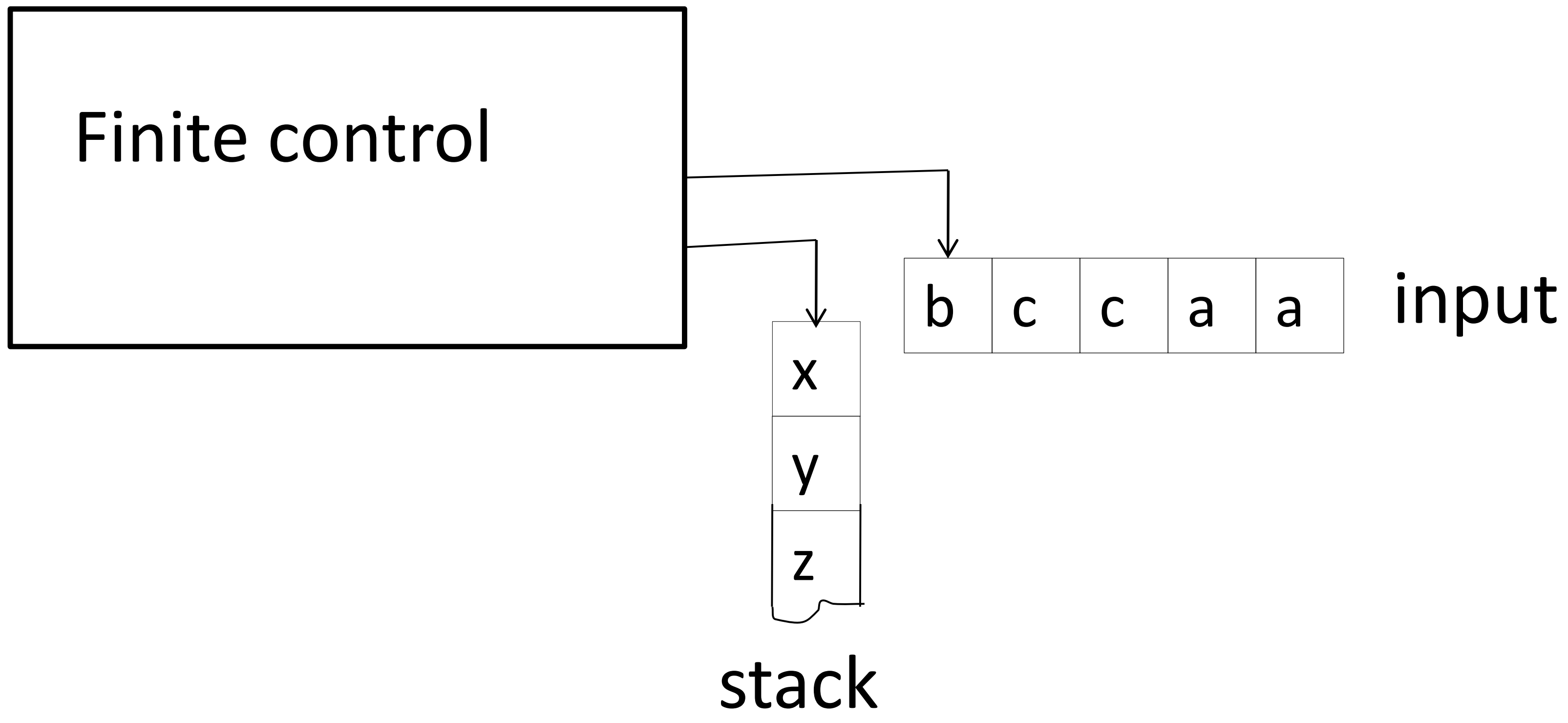
A pushdown automata is an NFA **augmented with an infinitely large stack**.

The additional memory enables recognition of some non regular languages.

Schematic of a Finite Automaton



Schematic of a Pushdown Automaton



Informal Description

A Pushdown Automata (PDA) can write an unbounded number of **stack symbols** on the stack and read these symbols later.

Writing a symbol onto the stack is called ***pushing*** and it pushes all symbols on the stack one stack cell down.

Informal Description

Removing a symbol off the stack is called ***popping*** and every symbol on the stack moves one stack cell up.

Note: A PDA can access only the stack’s **topmost symbol** (LIFO).

A PDA Recognizing $L = \{0^n 1^n\}$

This PDA reads symbols from the input.

As each 0 is read, it is pushed onto the stack.

As each 1 is read, a 0 is popped from the stack.

If the stack becomes empty exactly when the last 1 is read – **accept**.

Otherwise – **reject**.

Checking Stack Emptiness

The definition of a PDA does not give a special way to check emptiness.

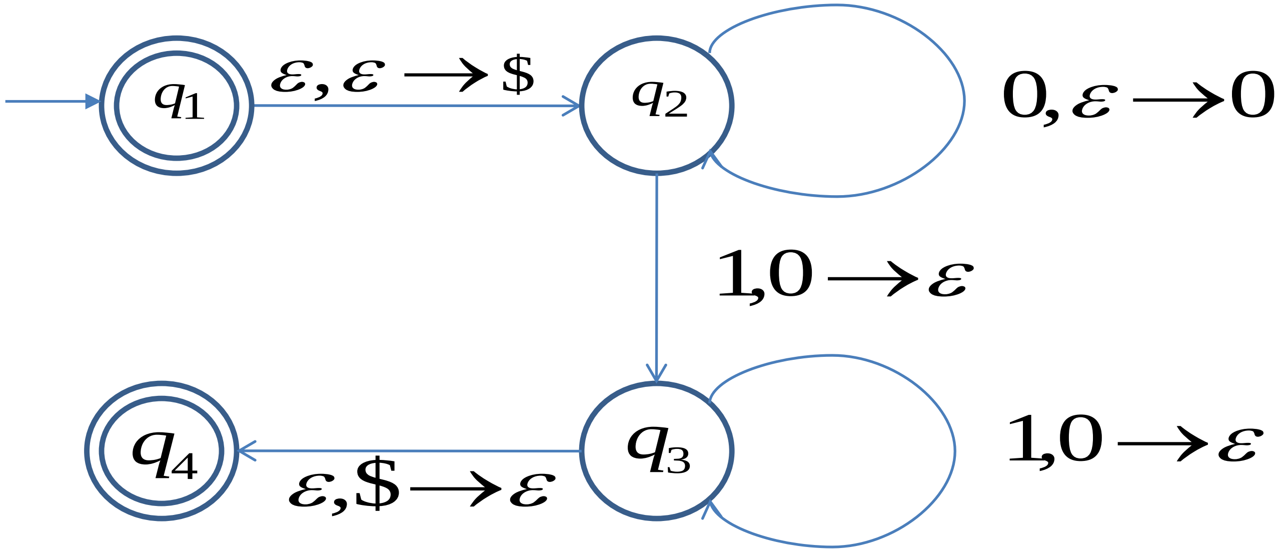
One way to do it is to augment the stack alphabet with a special “emptiness” marker, the symbol \$. (**Note:** There is nothing special about \$ any other symbol not in the original stack alphabet can do.)

Checking Stack Emptiness

The computation is started by an ε transition in which \$ is pushed on the stack.

If the end marker \$ is found on the stack at the end of the computation, it is popped by a single additional ε transition after which the automaton "knows" that the stack is empty.

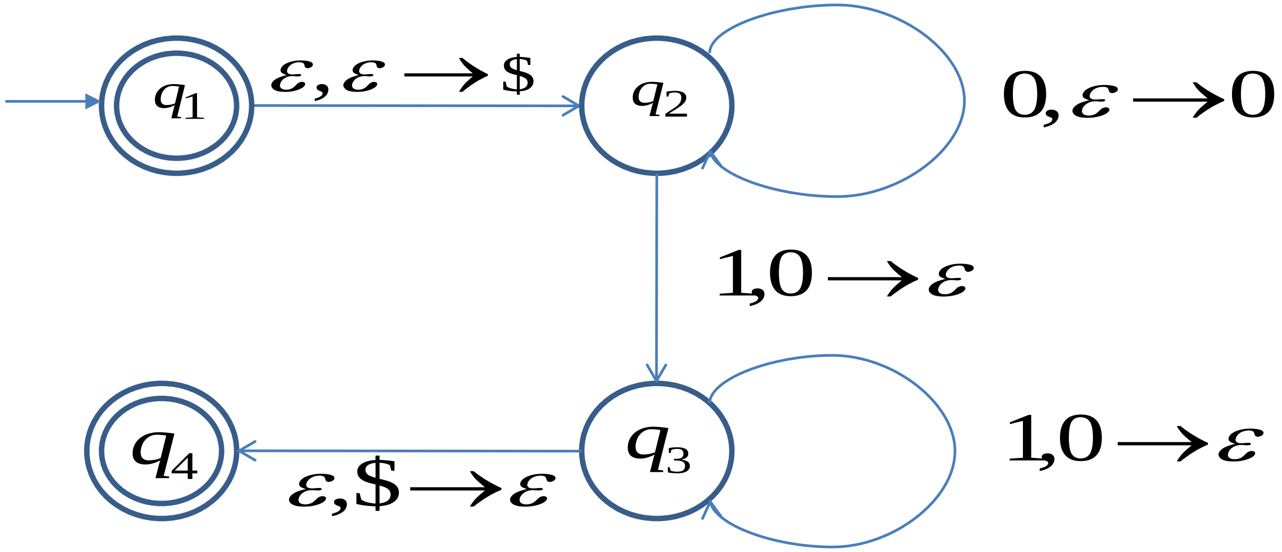
A PDA Recognizing $L = \{0^n 1^n\}$



q_1
$\varepsilon, \varepsilon \rightarrow \$$
q_2
$0, \varepsilon \rightarrow 0$
$1, 0 \rightarrow \varepsilon$
q_3
$1, 0 \rightarrow \varepsilon$
q_4
$\varepsilon, \$ \rightarrow \varepsilon$

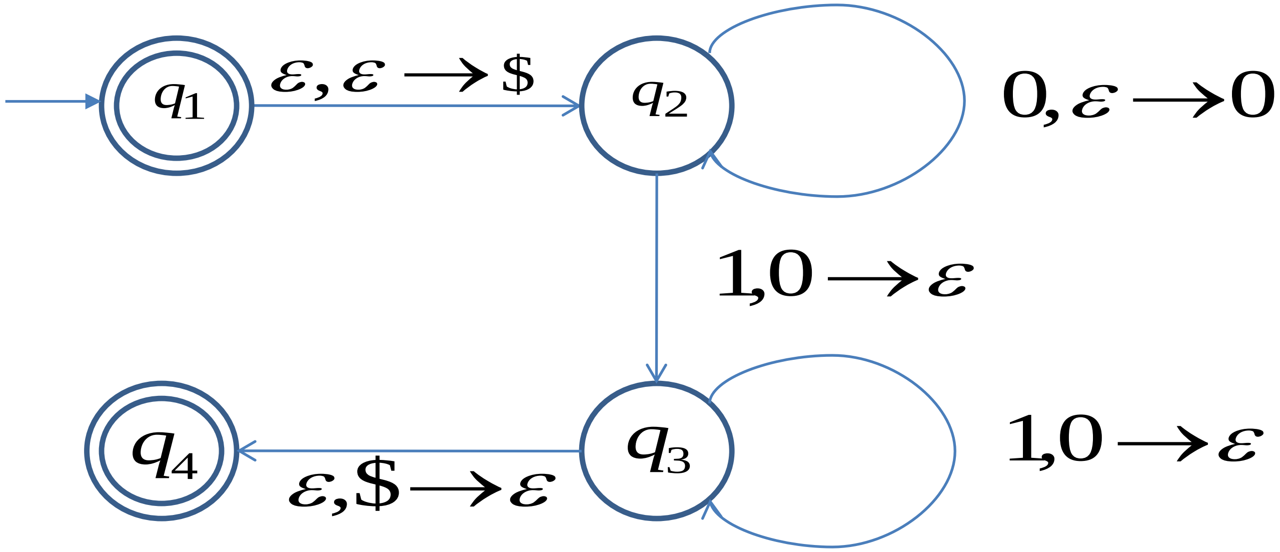
The label of each transition represents the input (left of arrow) and pushed stack symbol (right of the arrow).

A PDA Recognizing $L = \{0^n 1^n\}$



The $ symbol, pushed onto the stack at the beginning of the computation, is used as an "empty" marker.

A PDA Recognizing $L = \{0^n 1^n\}$


q_1
$\varepsilon, \varepsilon \rightarrow \$$
q_2
$0, \varepsilon \rightarrow 0$
$1, 0 \rightarrow \varepsilon$
q_3
$1, 0 \rightarrow \varepsilon$
q_4
$\varepsilon, \$ \rightarrow \varepsilon$

The PDA accepts either if the input is empty, or if scanning the input is completed and the PDA is at q_4.

Nondeterministic PDAs

A Nondeterministic PDA allows nondeterministic transitions.

Nondeterministic PDA-s are **strictly stronger** then deterministic PDA-s

In this respect, the situation is not similar to the situation of DFA-s and NFA-s.

Nondeterministic PDA-s are **equivalent to CFL-s**.

PDA – A Formal Definition

A ***pushdown automaton*** is a 6-tupple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

1. Q is a finite set called the ***states***.
2. Σ is the ***input alphabet***.
3. Γ is the ***stack alphabet***.
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$ is the ***transition function***.
5. $q_0 \in Q$ is the ***start state***, and
6. $F \subseteq Q$ is the set of ***accepting states***.

PDA - The Transition Function

Consider the expression $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$

Recall that $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, and that $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$.

Assume that the PDA is in state $q \in Q$, the next input symbol is $\sigma \in \Sigma$, and the top stack symbol is $\gamma \in \Gamma$.

PDA - The Transition Function

The next transition may either depend on the input symbol σ and the stack symbol γ , or only on the input symbol σ, or only on the stack symbol γ, or on none of them.

This choice is formally expressed by the argument of the transition function as detailed in the next slides.

Transition Function Sub-steps

Each step of the automaton is **atomic**, meaning it is executed in a single indivisible time unit.

For descriptive purposes only, each step is divided into two separate sub-steps:

Sub-step1: A symbol may be read from the input, a symbol may be read and popped off the stack.

Sub-step2: A state transition is carried out and a stack symbol may be pushed on the stack.

Transition Function – 1st Sub-step

If the transition depends both on σ and γ we write $\delta(q,\sigma,\gamma)$. In this case σ is consumed and γ is removed from the stack.

If the transition depends only on σ we write $\delta(q,\sigma,\varepsilon)$, σ is consumed and the stack does not change.

Transition Function – 1st Sub-step

If the transition depends only on γ , we write $\delta(q,\varepsilon,\gamma)$. In this case σ is not consumed and γ is removed from the stack.

Finally, If the transition depends neither on σ , nor on γ , we write $\delta(q,\varepsilon,\varepsilon)$. In this case σ is not consumed and the stack is not changed.

PDA - The Transition Function

The range of the transition function is P(Q$\times\Gamma$):

the power set of the Cartesian product of the set of PDA states and the stack alphabet.

Using pairs means that δ determines:

1. The new state to which the PDA moves.
2. The new stack symbol **pushed** on the stack.

PDA - The Transition Function

Using the power set means that the PDA is nondeterministic: At any given situation, it may make a ***nondeterministic transition***.

Finally, the use of $\Gamma_{\varepsilon} = \Gamma \cup \{\varepsilon\}$ means that at each transition the PDA may either push a stack symbol onto the stack or not (if the value is ε).

CFL-s and PDA-s are Equivalent

Theorem:

A language is CFL if and only if there exists a PDA accepting it.

Lemma->

For any CFL L, there exists a PDA P such that $L = L(P)$.

Proof Idea

Since L is a CFL there exists a CFG G such that $L = L(G)$. We will present a PDA P, that recognizes L.

The PDA P starts with a word $w \in \Sigma^*$ on its input.

In order to decide whether $w \in L(G)$, P simulates the derivation of w.

Proof Idea (cont.)

Recall that a derivation is a sequence of strings, where each string contains variables and terminals. The first string is always the **start symbol** of G and each string is obtained from the previous one by a single activation of some substitution rule.

Proof Idea (cont.)

A string may allow activation of several rules and the PDA P non deterministically guesses the next rule to be activated.

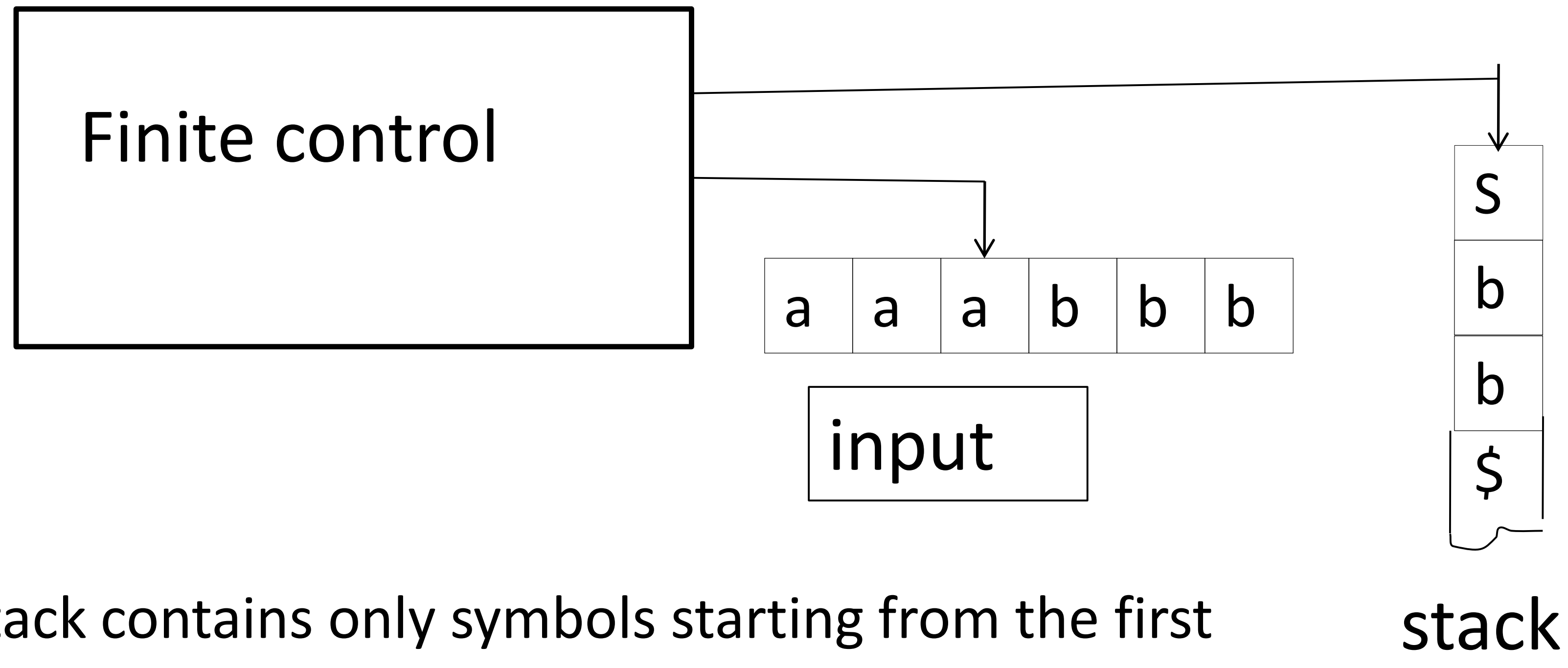
The initial idea for the simulation is to store each intermediate string on the stack. Upon each production, the string on the stack before production is transformed to the string after production.

Proof Idea (cont.)

Unfortunately, this idea does not quite work since at any given moment, P can only access the top symbol on the stack.

To overcome this problem, the stack holds only a suffix of each intermediate string where the top symbol is the variable to be substituted during the next production.

The Intermediate String $aaSbb$



Stack contains only symbols starting from the first variable in the intermediate string.

Informal Description of P

Push the marker \$ and the start symbol S on the stack.

Repeat

If the top symbol is a variable V – Replace V by the right hand side of some non deterministically chosen rule whose left hand side is V .

…..

Informal Description of P

Push the marker \$ and the start symbol S on the stack.

Repeat

If the top symbol is a variable V – Replace V by the right hand side of some non deterministically chosen rule whose left hand side is V .

If the top symbol is a terminal compare it with the next symbol on the input. **If** equal – advance the input and pop the variable **else** – reject.

Informal Description of P

Push the marker \$ and the start symbol S on the stack.

Repeat

If the top symbol is a variable V – Replace V by the right hand side of some non deterministically chosen rule whose left hand side is V .

If the top symbol is a terminal compare it with the next symbol on the input. **If** equal – advance the input and pop the variable **else** – reject.

If the top symbol is \$ **and** the input is finished – accept **else** – reject

The Proof (formal)

We start by defining **Extended Transitions**:

Assume that PDA P is in state q , it reads $a \in \Sigma$ from the input and pops $s \in \Gamma$ from the stack and then moves to state r while pushing $u = u_1, u_{2,} \ldots, u_l$ onto the stack.

This is denoted by $(r,u) \in \delta(q,a,s)$.

Next, extended transitions are implemented.

Implementing Extended Trans.

Add states $q_1, q_2, \ldots, q_{l-1}$.

Set the transition function δ as follows:

Add (q_1, u_l) to $\delta(q, a, s)$.

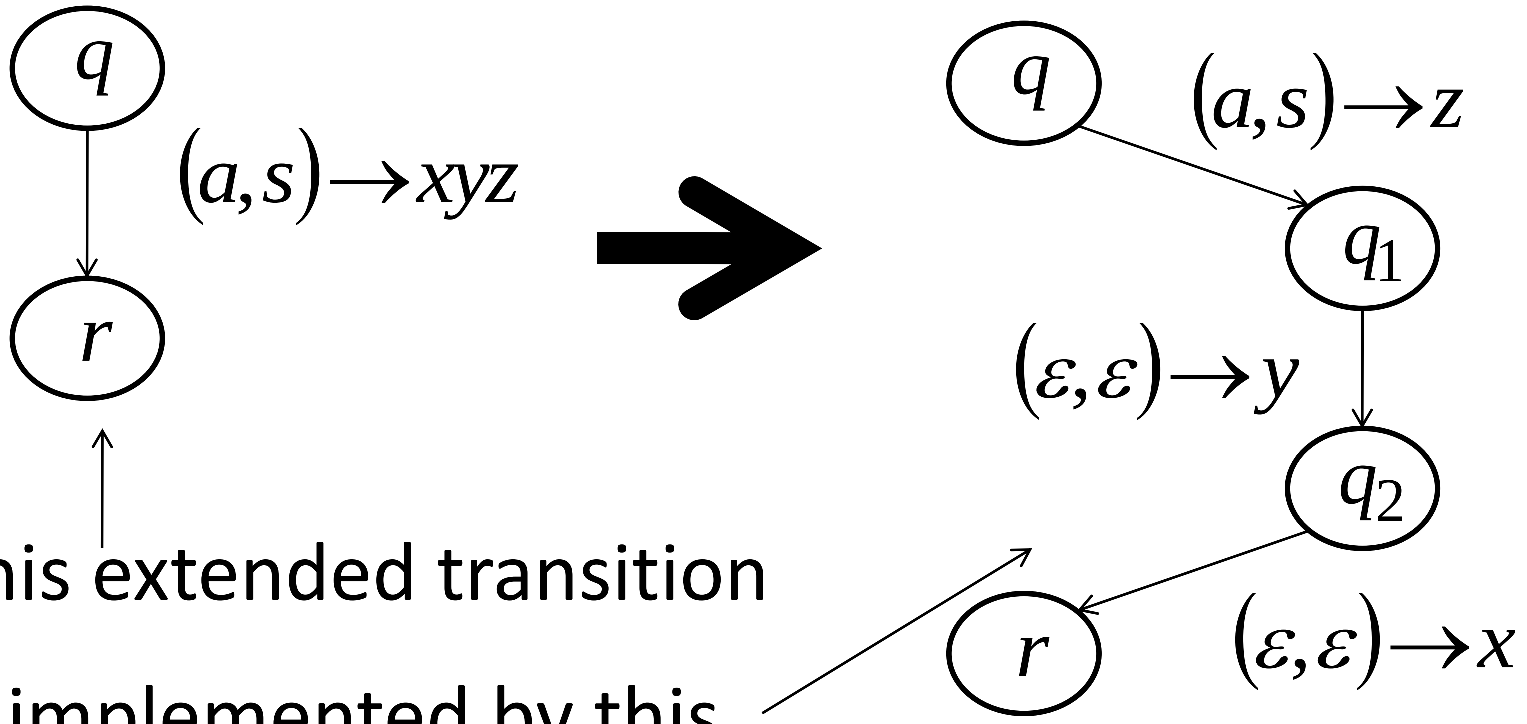
Set $\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, u_{l-1})\}$,

$\delta(q_2, \varepsilon, \varepsilon) = \{(q_3, u_{l-2})\}$,

……

$\delta(q_{l-1}, \varepsilon, \varepsilon) = \{(r, u_1)\}$ (see next slide)

Implementing Extended Trans.



q
$(a,s)\rightarrow xyz$
r
q
$(a,s)\rightarrow z$
q_1
$(\varepsilon,\varepsilon)\rightarrow y$
q_2
r
$(\varepsilon,\varepsilon)\rightarrow x$

This extended transition

Is implemented by this

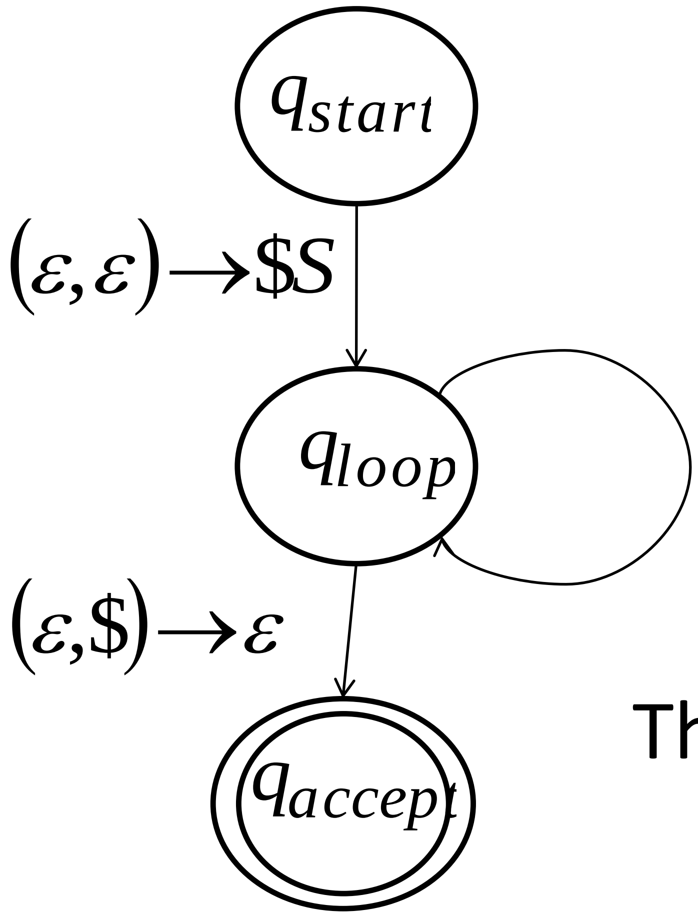
transition sequence

The Proof

Let G be an arbitrary CFG. Now we are ready to construct the PDA, P such that that $L(P)=L(G)$. The states of P are a $Q=\{q_{start}, q_{loop}, q_{accept}\} \cup E$ where E contains all states needed to implement the extended transitions presented in the previous slide.

The PDA P is presented on the next slide:

The Result PDA


q_{start}
$(\varepsilon,\varepsilon)\rightarrow \$S$
q_{loop}
$(\varepsilon,\$)\rightarrow\varepsilon$
q_{accept}

$(\varepsilon, A)\rightarrow \overleftarrow{w}$ for rule $A\rightarrow w$

$(a,a)\rightarrow\varepsilon$ for terminal a

This completes the Proof

Example

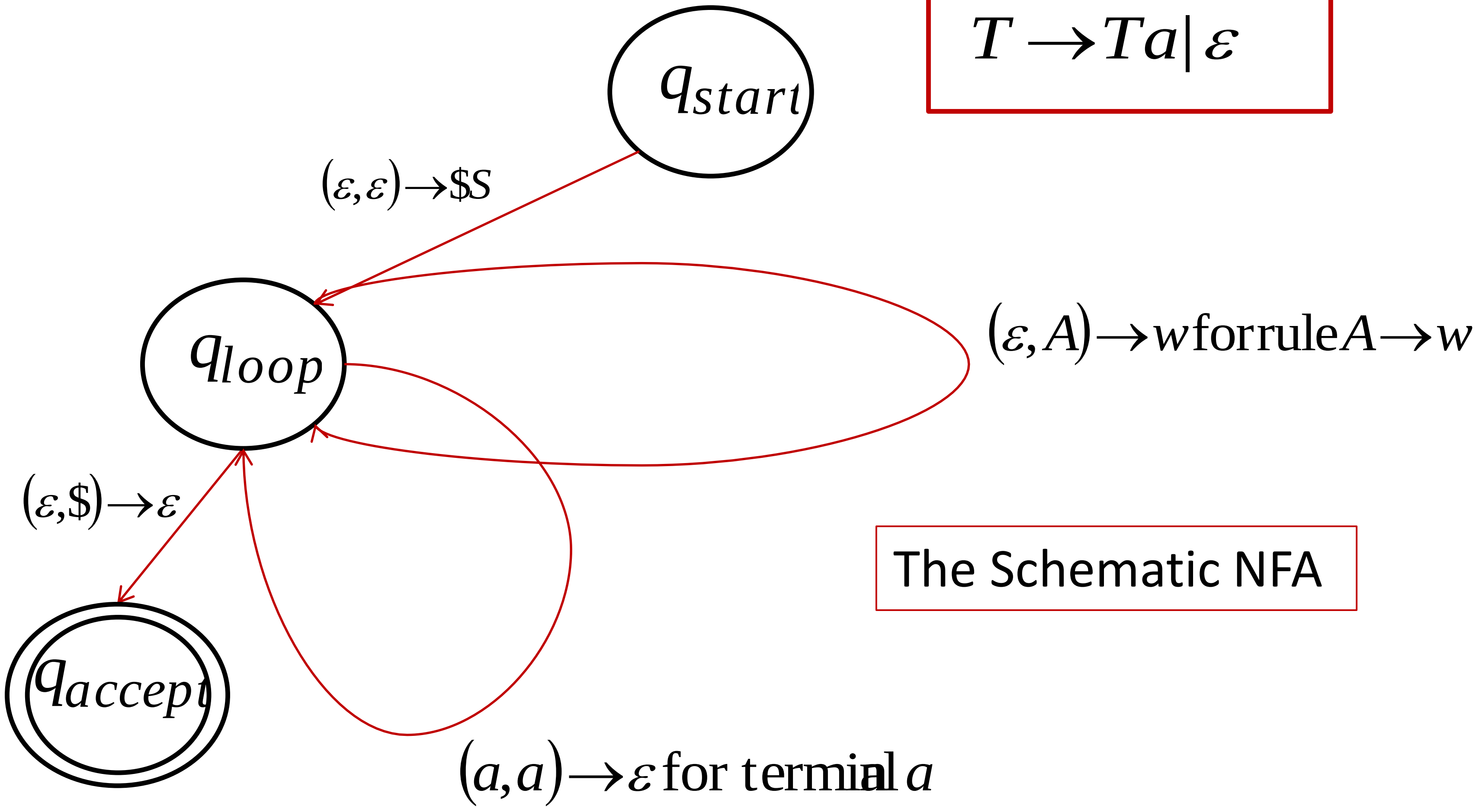
Consider the following CFG:

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

Example

Consider the following CFG:

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



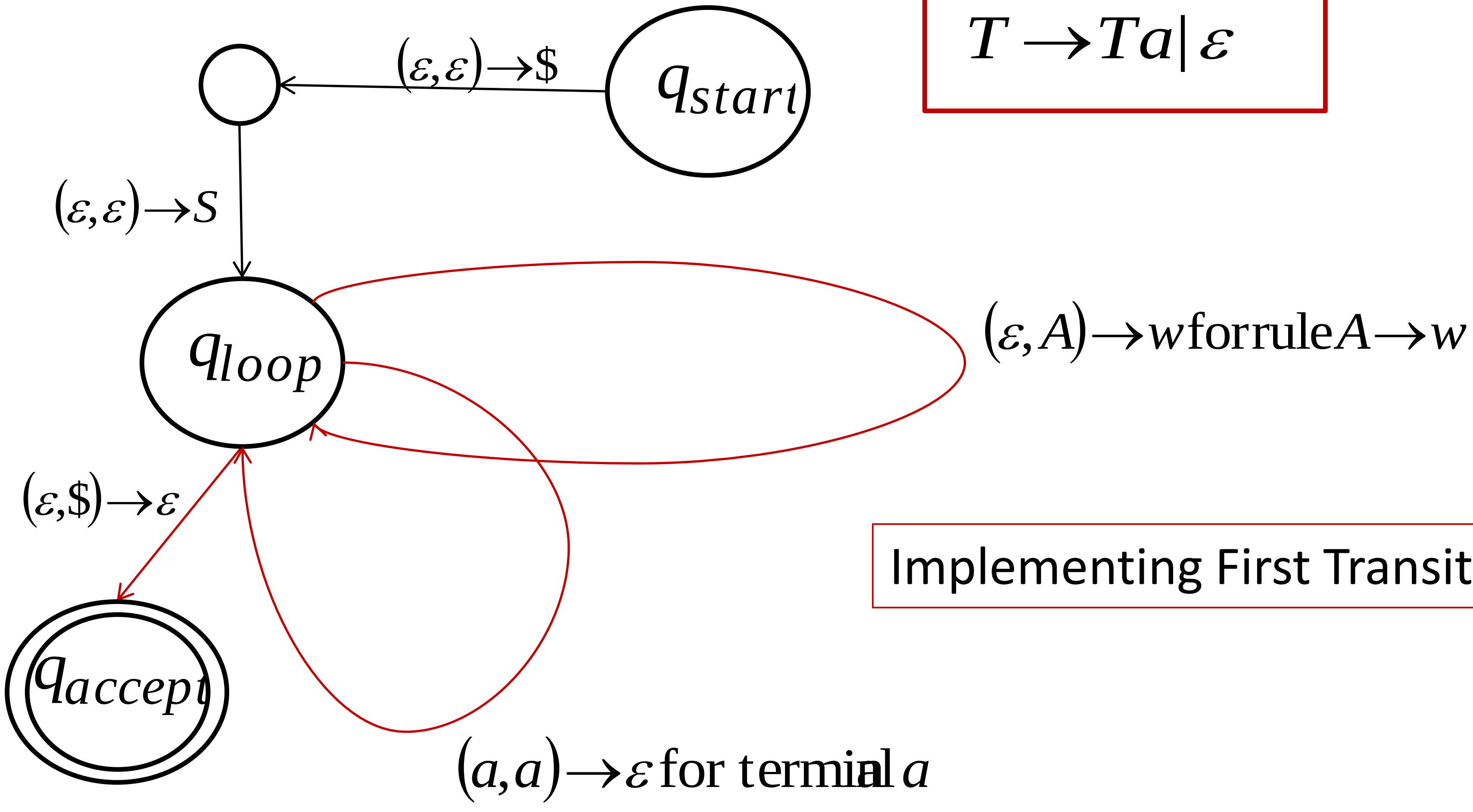
q_{start}
$(\varepsilon,\varepsilon) \to \$S$
q_{loop}
$(\varepsilon, A) \to w$ for rule $A \to w$
$(\varepsilon,\$) \to \varepsilon$
q_{accept}
$(a,a) \to \varepsilon$ for terminal a

The Schematic NFA

Example

Consider the following CFG:

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



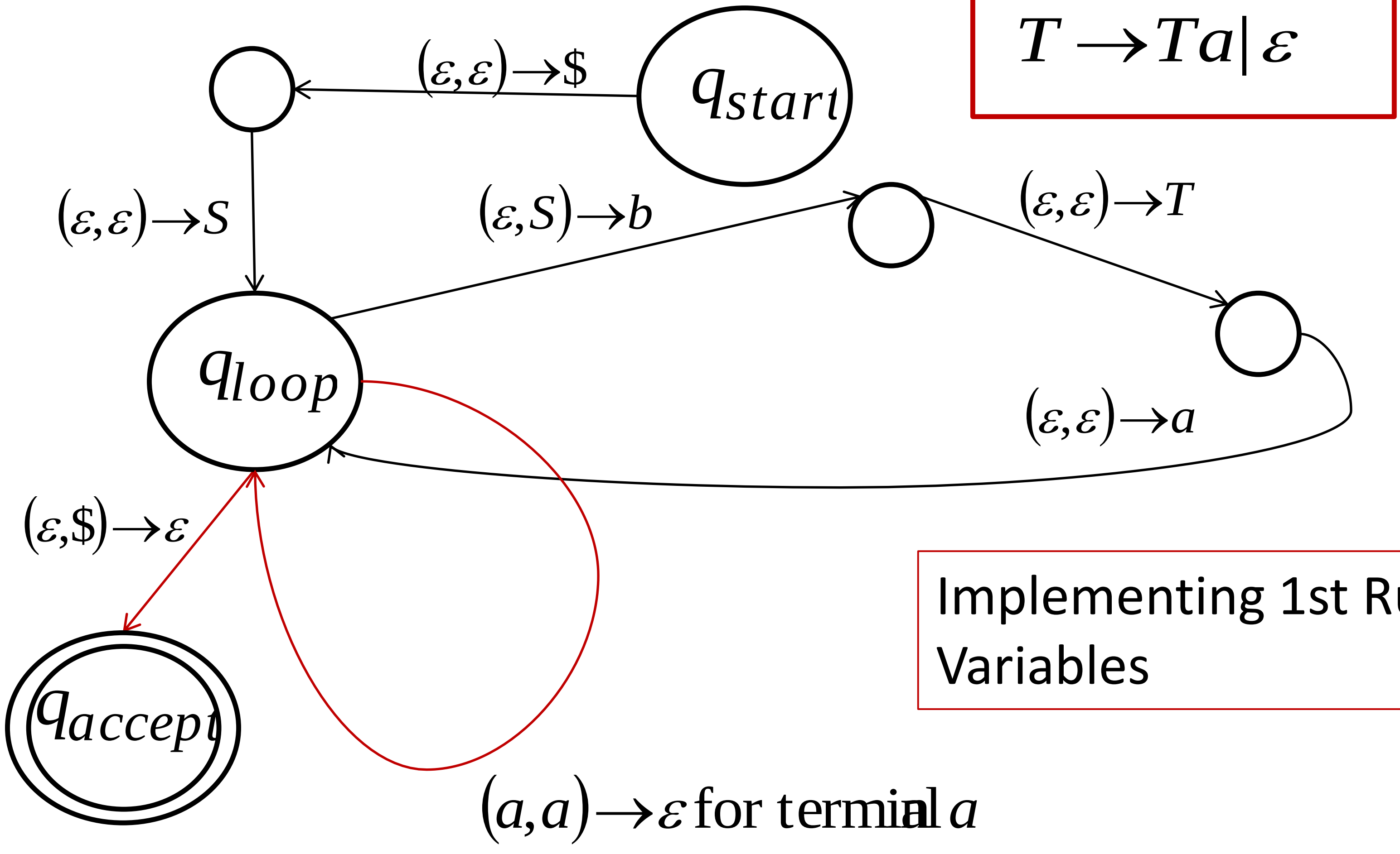
$(\varepsilon,\varepsilon)\to\$$
q_{start}
$(\varepsilon,\varepsilon)\to S$
q_{loop}
$(\varepsilon,A)\to w$ for rule $A\to w$
$(\varepsilon,\$)\to\varepsilon$
q_{accept}
$(a,a)\to\varepsilon$ for terminal a

Implementing First Transition

Example

Consider the following CFG:

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



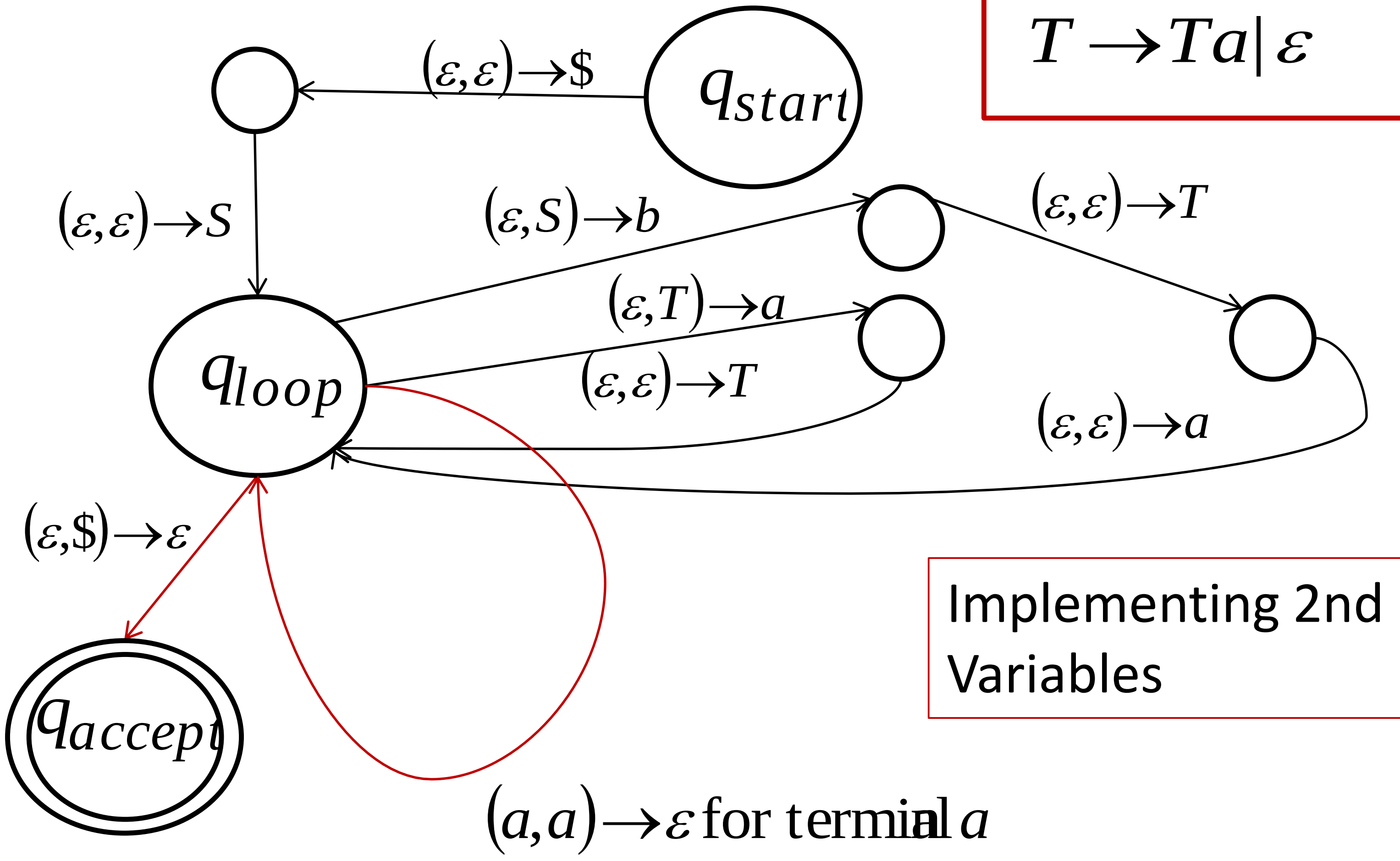
$(\varepsilon,\varepsilon)\to\$$
q_{start}
$(\varepsilon,\varepsilon)\to S$
$(\varepsilon,S)\to b$
$(\varepsilon,\varepsilon)\to T$
q_{loop}
$(\varepsilon,\varepsilon)\to a$
$(\varepsilon,\$)\to\varepsilon$
q_{accept}
$(a,a)\to\varepsilon$ for terminal a

Implementing 1st Rule with Variables

Example

Consider the following CFG:

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



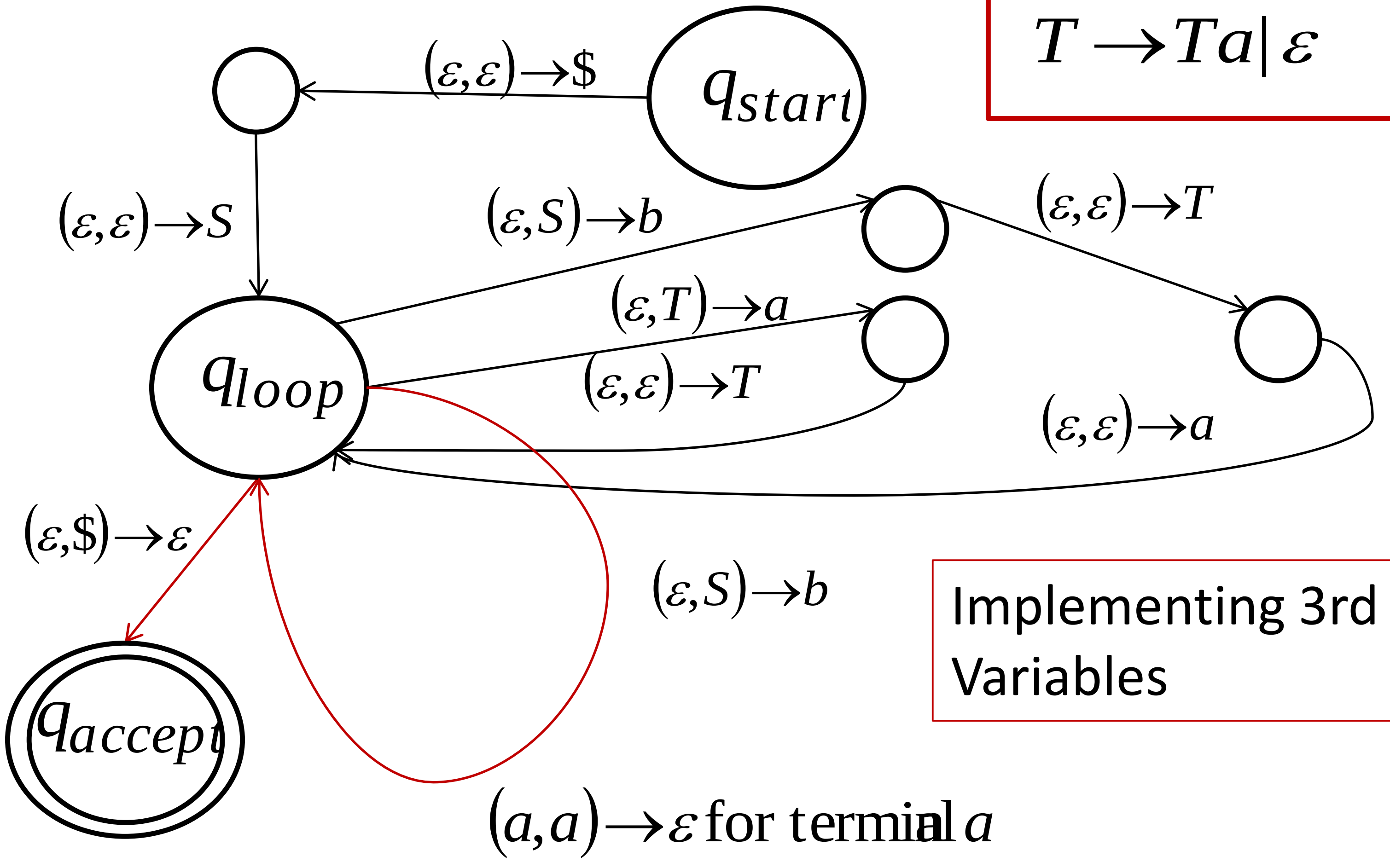
$(\varepsilon,\varepsilon)\rightarrow\$$
q_{start}
$(\varepsilon,\varepsilon)\rightarrow S$
$(\varepsilon,S)\rightarrow b$
$(\varepsilon,\varepsilon)\rightarrow T$
$(\varepsilon,T)\rightarrow a$
$(\varepsilon,\varepsilon)\rightarrow T$
$(\varepsilon,\varepsilon)\rightarrow a$
q_{loop}
$(\varepsilon,\$)\rightarrow\varepsilon$
q_{accept}
$(a,a)\rightarrow\varepsilon$ for terminal a

Implementing 2nd Rule with Variables

Example

Consider the following CFG:

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

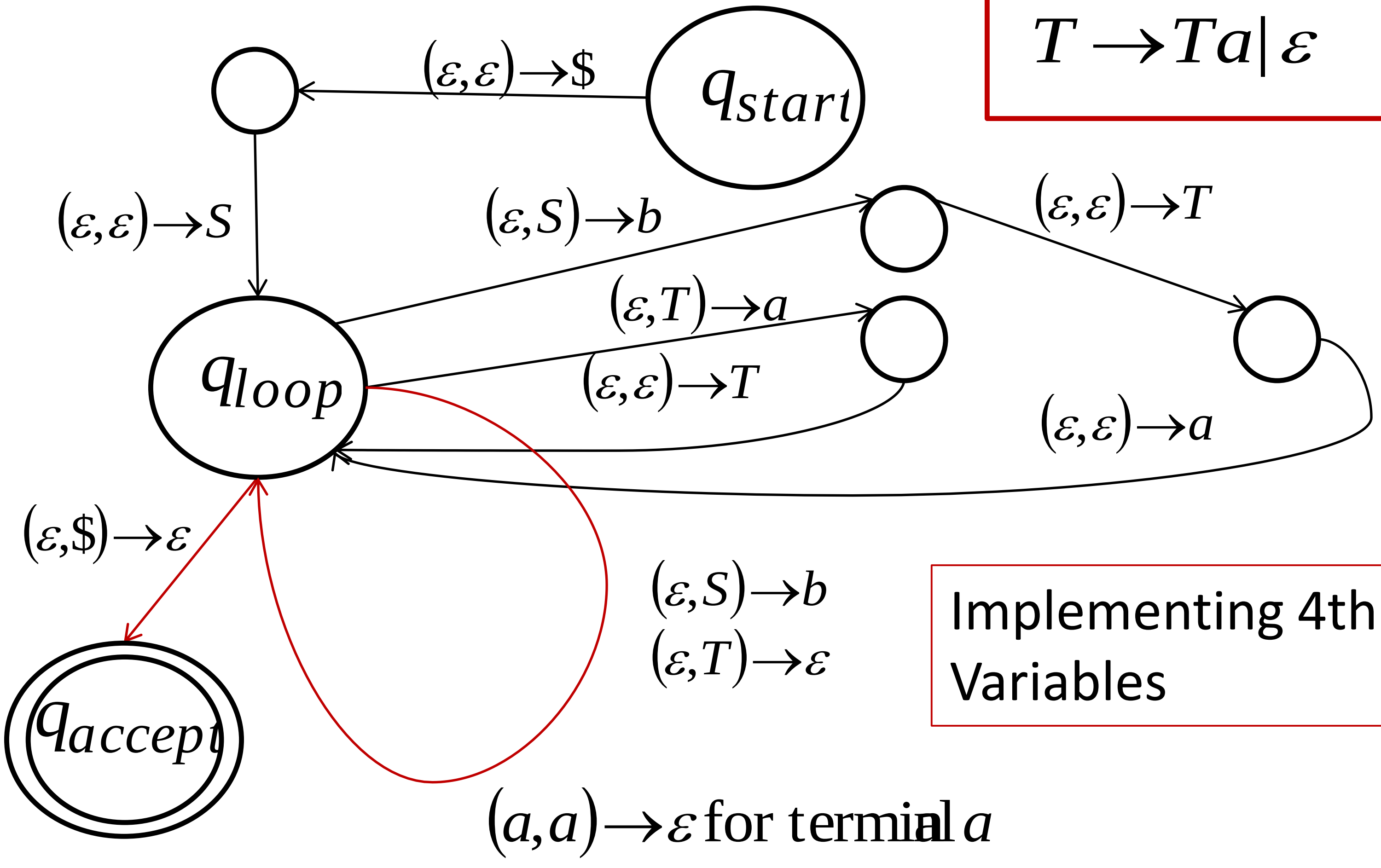


$(\varepsilon,\varepsilon)\rightarrow\$$
q_{start}
$(\varepsilon,\varepsilon)\rightarrow S$
$(\varepsilon,S)\rightarrow b$
$(\varepsilon,\varepsilon)\rightarrow T$
$(\varepsilon,T)\rightarrow a$
q_{loop}
$(\varepsilon,\varepsilon)\rightarrow T$
$(\varepsilon,\varepsilon)\rightarrow a$
$(\varepsilon,\$)\rightarrow\varepsilon$
$(\varepsilon,S)\rightarrow b$
q_{accept}
$(a,a)\rightarrow\varepsilon$ for terminal a

Implementing 3rd Rule with Variables

Example

Consider the following CFG: $S \to aTb \mid b$ $T \to Ta \mid \varepsilon$



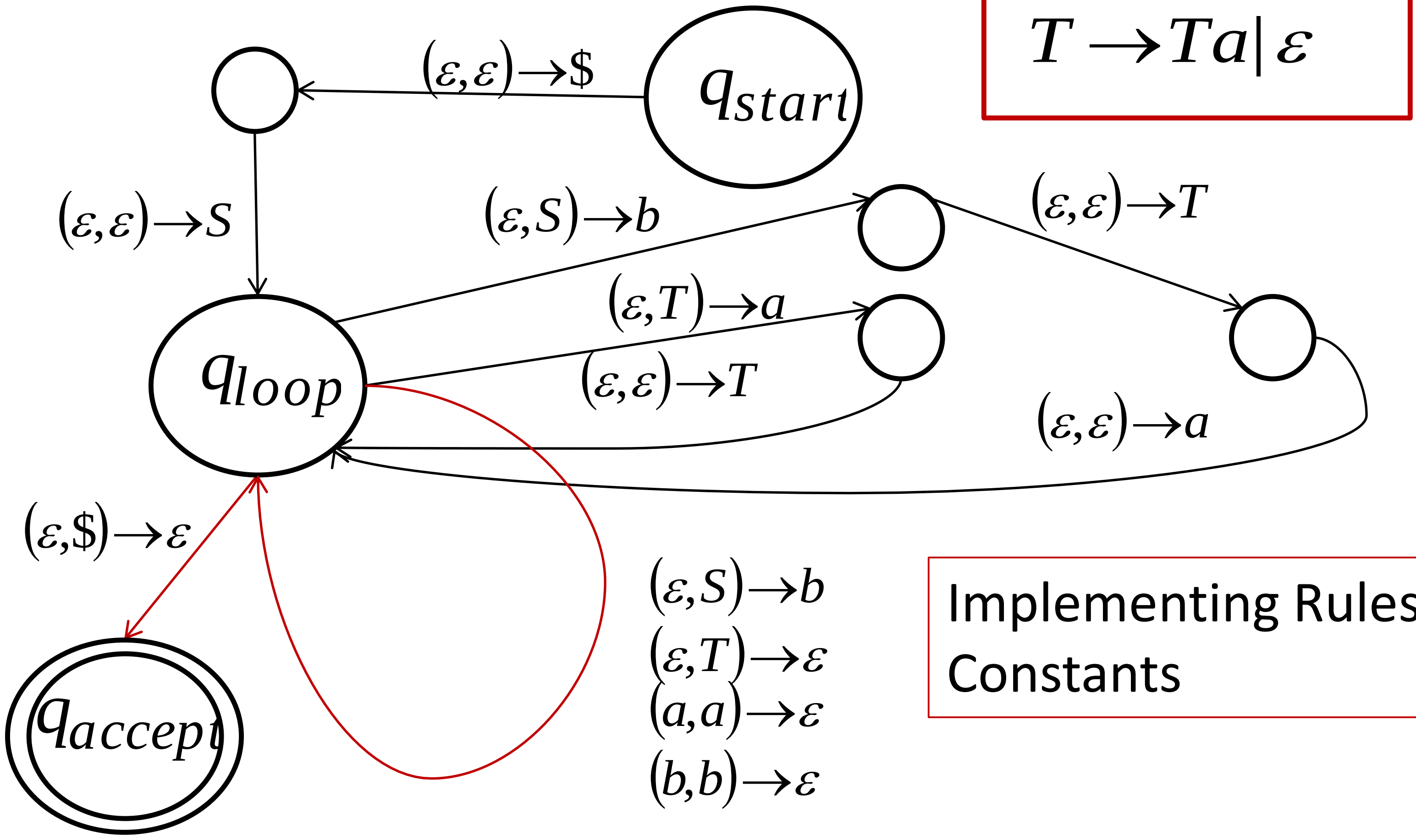
q_{start}
$(\varepsilon,\varepsilon)\to\$$
$(\varepsilon,\varepsilon)\to S$
q_{loop}
$(\varepsilon,S)\to b$
$(\varepsilon,\varepsilon)\to T$
$(\varepsilon,T)\to a$
$(\varepsilon,\varepsilon)\to T$
$(\varepsilon,\varepsilon)\to a$
$(\varepsilon,\$)\to\varepsilon$
q_{accept}
$(\varepsilon,S)\to b$
$(\varepsilon,T)\to\varepsilon$
$(a,a)\to\varepsilon$ for terminal a

Implementing 4th Rule with Variables

Example

Consider the following CFG:

$$\boxed{\begin{array}{l} S \to aTb \mid b \\ T \to Ta \mid \varepsilon \end{array}}$$



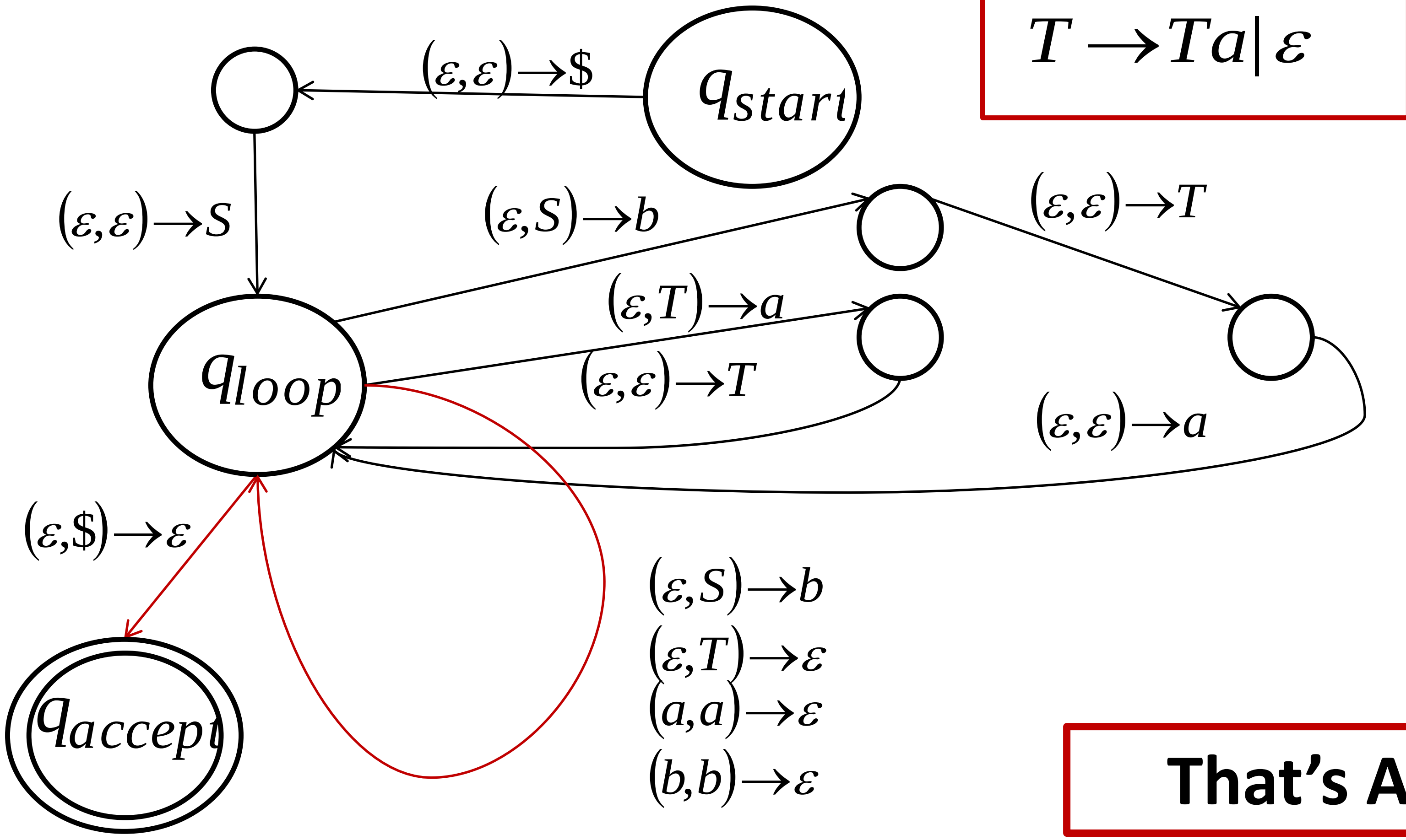
q_{start}
$(\varepsilon,\varepsilon)\to\$$
$(\varepsilon,\varepsilon)\to S$
q_{loop}
$(\varepsilon,S)\to b$
$(\varepsilon,\varepsilon)\to T$
$(\varepsilon,T)\to a$
$(\varepsilon,\varepsilon)\to T$
$(\varepsilon,\varepsilon)\to a$
$(\varepsilon,\$)\to\varepsilon$
q_{accept}
$(\varepsilon,S)\to b$
$(\varepsilon,T)\to\varepsilon$
$(a,a)\to\varepsilon$
$(b,b)\to\varepsilon$

Implementing Rules with Constants

Example

Consider the following CFG:

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


q_{start}
$(\varepsilon,\varepsilon)\rightarrow\$$
$(\varepsilon,\varepsilon)\rightarrow S$
q_{loop}
$(\varepsilon,S)\rightarrow b$
$(\varepsilon,\varepsilon)\rightarrow T$
$(\varepsilon,T)\rightarrow a$
$(\varepsilon,\varepsilon)\rightarrow T$
$(\varepsilon,\varepsilon)\rightarrow a$
$(\varepsilon,\$)\rightarrow\varepsilon$
$(\varepsilon,S)\rightarrow b$
$(\varepsilon,T)\rightarrow\varepsilon$
$(a,a)\rightarrow\varepsilon$
$(b,b)\rightarrow\varepsilon$
q_{accept}

That's All !!!