CMPU 240 · Theory of Computation

The Power of PDAs

26 March 2026

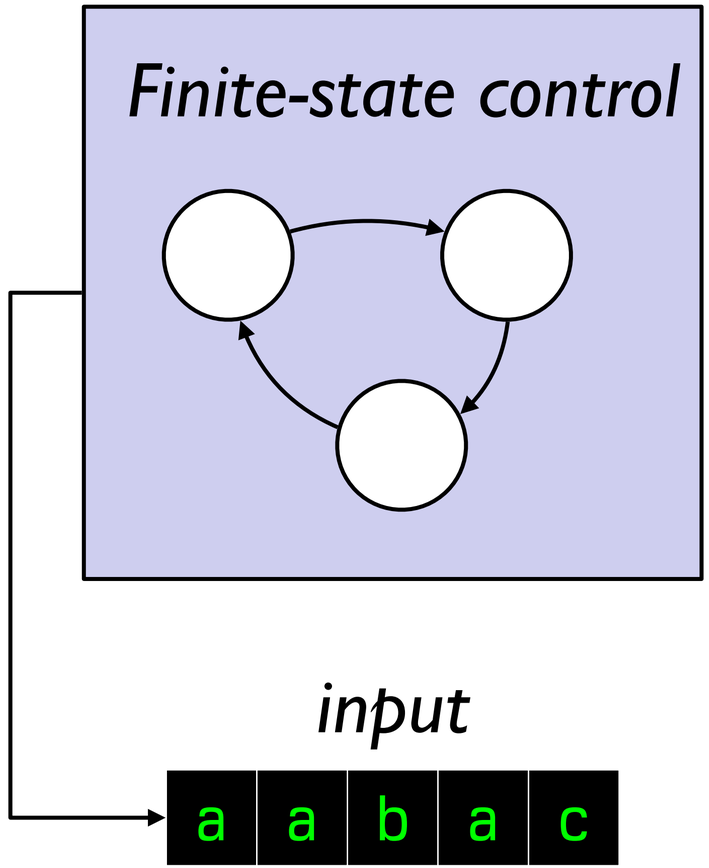
Where are we?

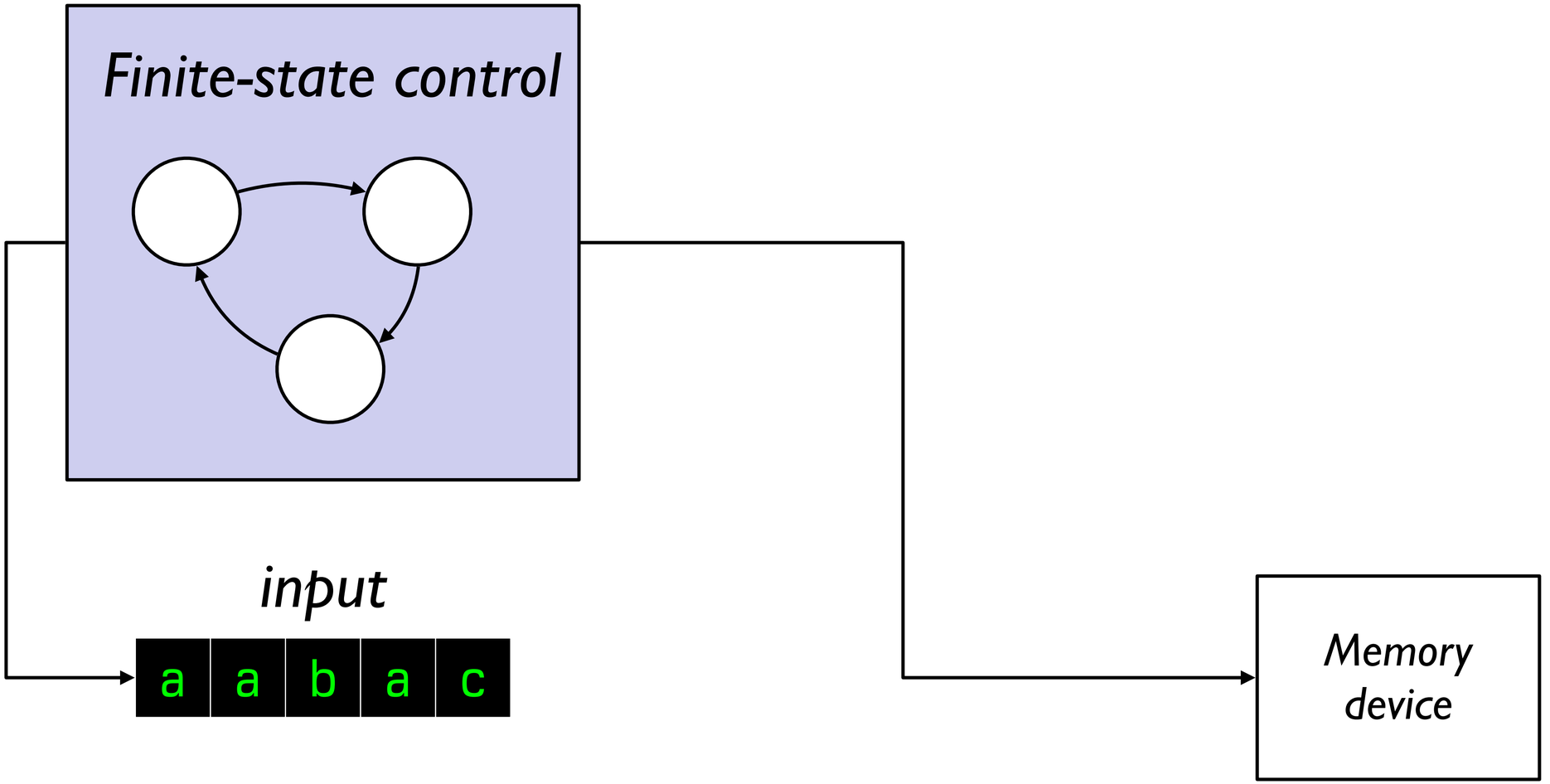
Previously:

- Introduced context-free grammars
- Introduced pushdown automata

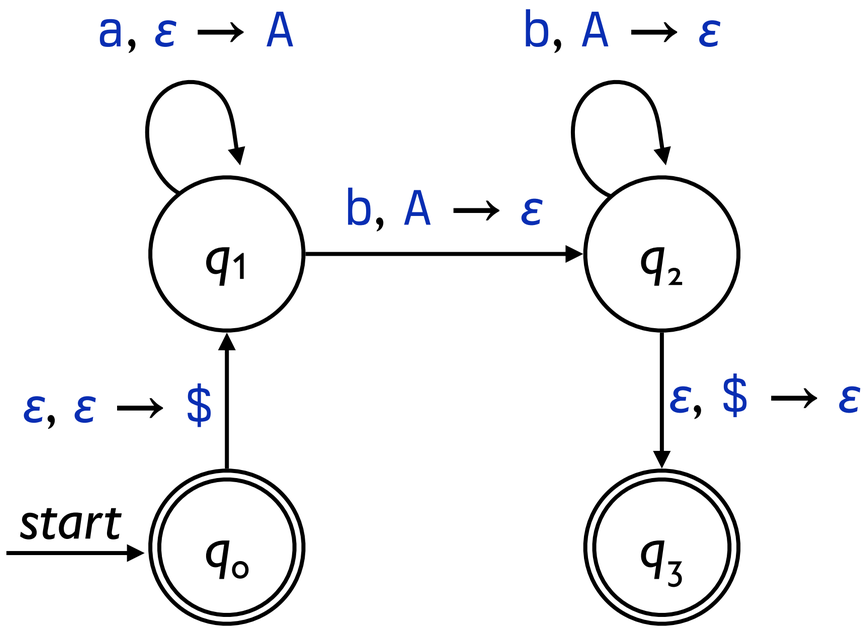
Today:

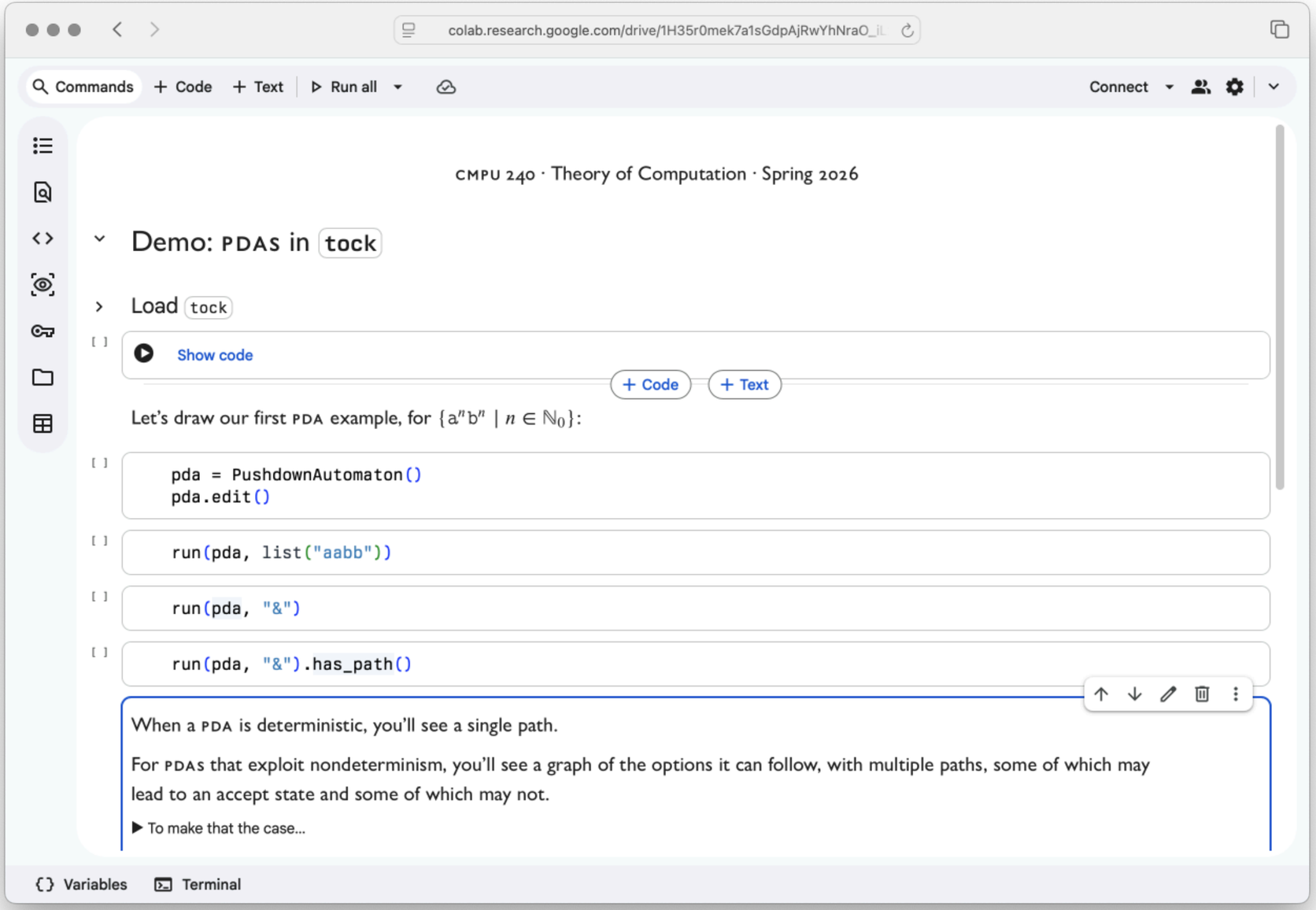
- Equivalence of CFGs and PDAs
- (Time allowing:) A bit more about PDAs and determinism


Finite-state control
input
a
a
b
a
c

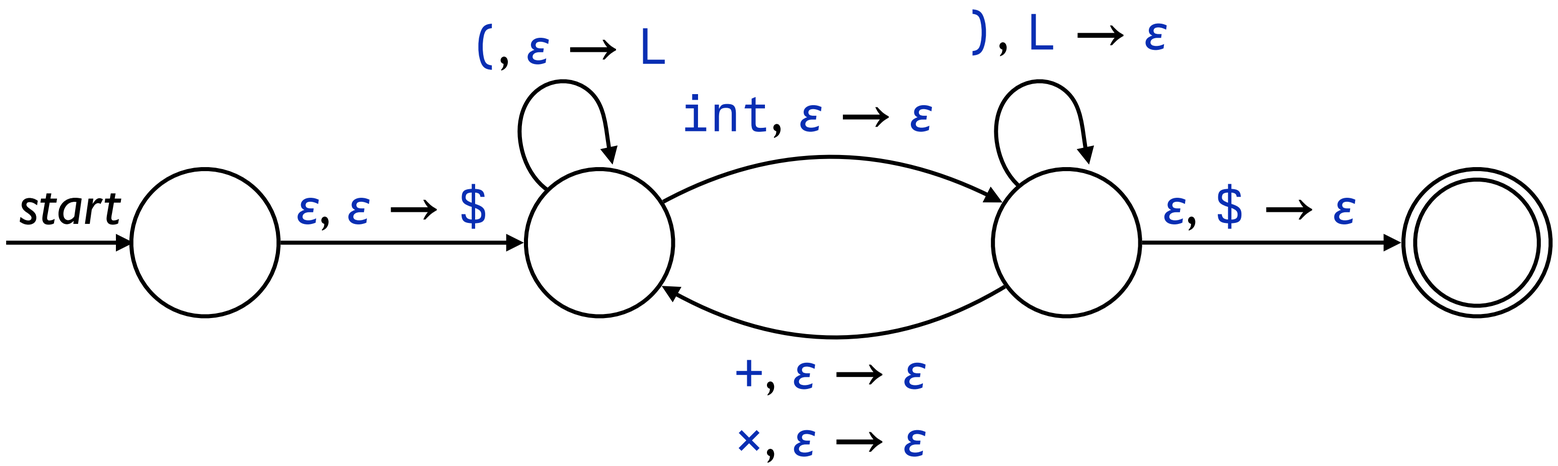

Finite-state control
input
a a b a c
Memory device

Example: $a^n b^n$




colab.research.google.com/drive/1H35r0mek7a1sGdpAjRwYhNraO_iL
Commands
+ Code
+ Text
Run all
Connect
CMPU 240 · Theory of Computation · Spring 2026
Demo: PDAs in tock
Load tock
Show code
+ Code
+ Text
Let's draw our first PDA example, for $\{a^n b^n \mid n \in \mathbb{N}_0\}$:
pda = PushdownAutomaton()
pda.edit()
run(pda, list("aabb"))
run(pda, "&")
run(pda, "&").has_path()
When a PDA is deterministic, you'll see a single path.
For PDAs that exploit nondeterminism, you'll see a graph of the options it can follow, with multiple paths, some of which may lead to an accept state and some of which may not.
▶ To make that the case...
Variables
Terminal

Example: Arithmetic



Shorthand for pushing multiple symbols to the stack

$(r, \mathtt{xyz}) \in \delta(q, \mathtt{a}, \mathtt{s})$

q is the current state

a is the next input symbol

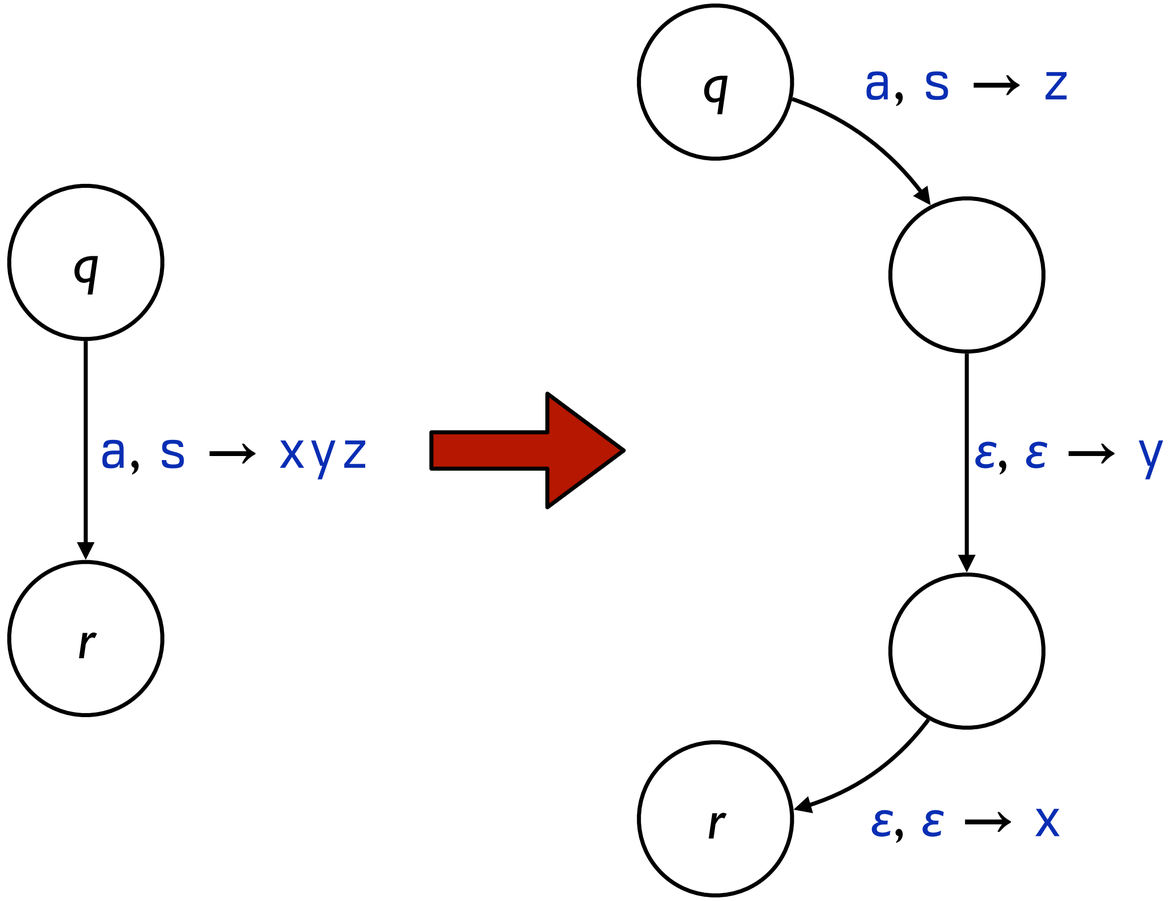
s is on the top of the stack

Do the following:

Read a

Pop s

Push xyz



Describing the operation of a PDA

How can we describe the operation of a pushdown automaton for a given input?

(Other than a lot of slides…)

To know what a pushdown automaton can do at any point in its operation, we need to know

its state,

the input it can read, and

the entire contents of its stack.

We call this combination an ***instantaneous description*** (ID), written in the form (q, w, a), where

q = current state

w = input characters that haven't been read yet

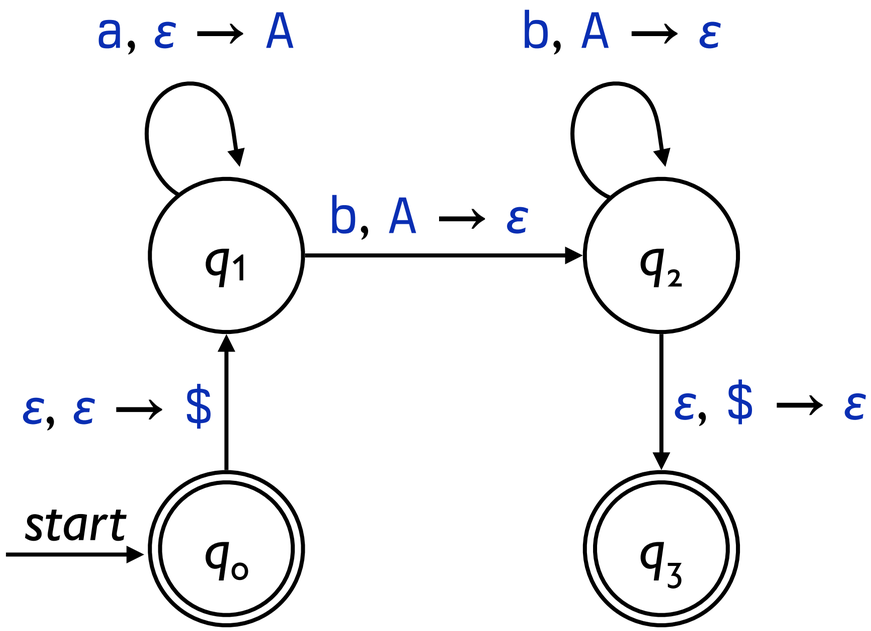
a = stack contents, with the top on the left

Describing the moves of the PDA

If $\delta(q, a, X)$ contains (p, a), then we say

$$(q, aw, X\beta) \vdash (p, w, a\beta)$$

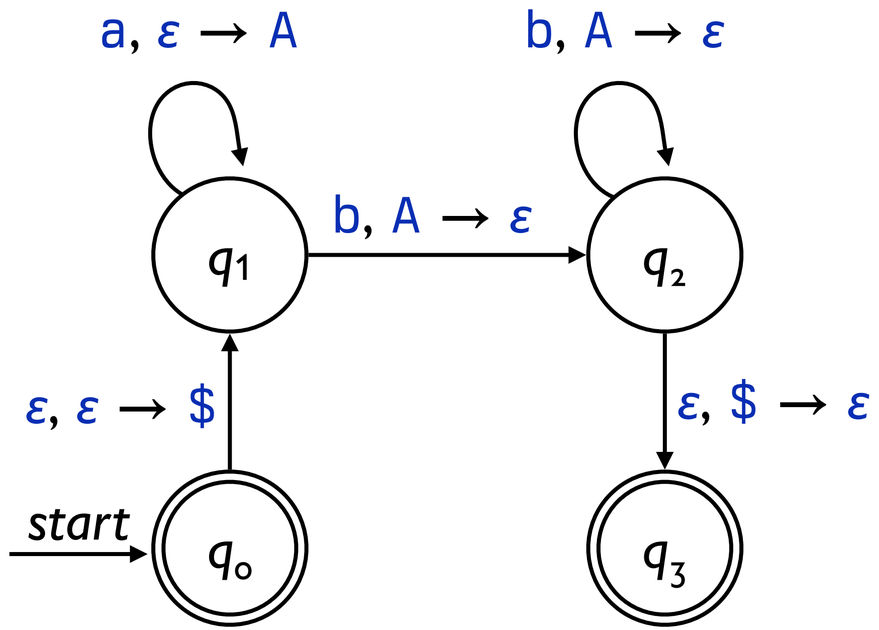
$\vdash$ *is the "goes to" relation*



What are the moves of this PDA for input aabb?

Example

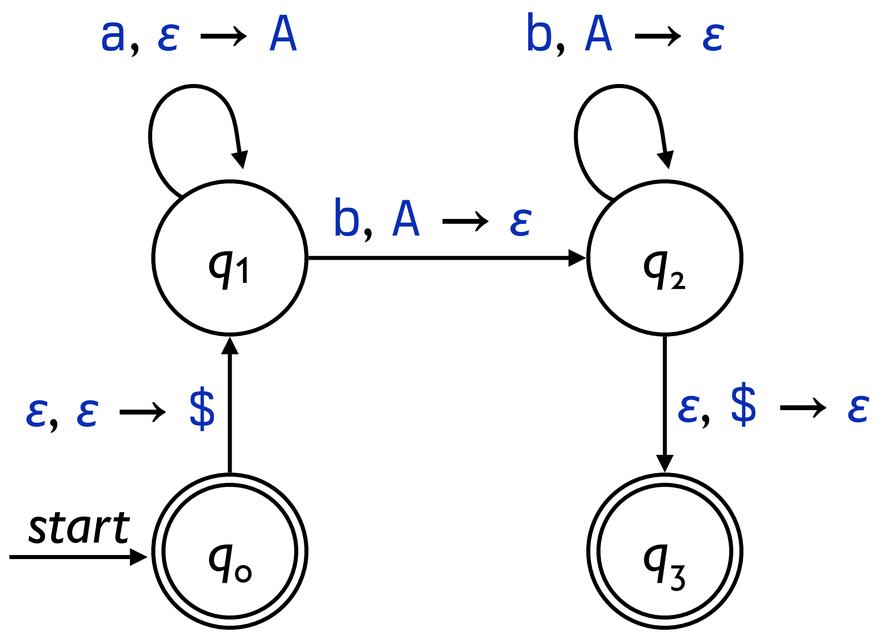
$(q_o,$ aabb, $\varepsilon)$



Example

$(q_0,\ \texttt{aabb},\ \varepsilon)$

$\vdash (q_1,\ \texttt{aabb},\ \$)$

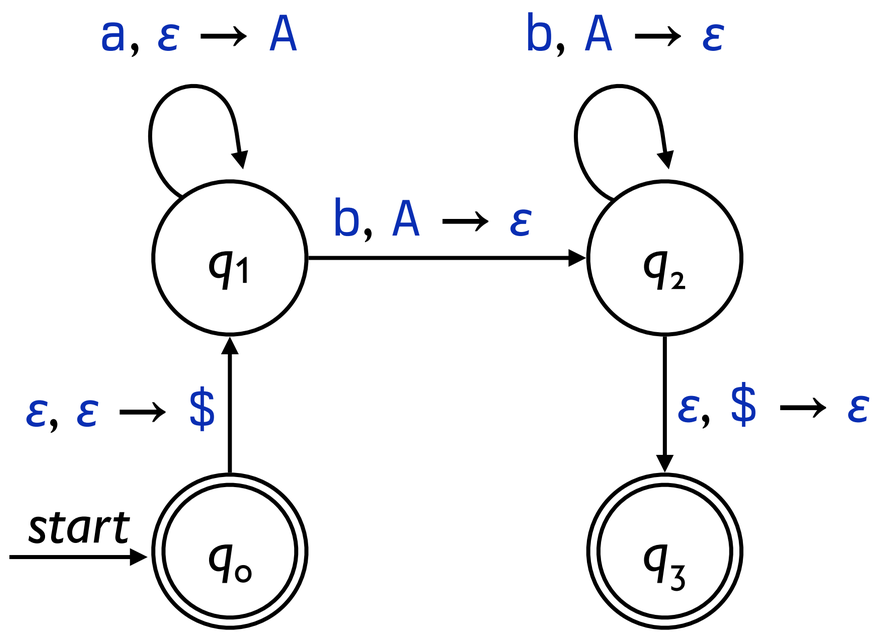


Example

$(q_0, \texttt{aabb}, \varepsilon)$

$\vdash (q_1, \texttt{aabb}, \$)$

$\vdash (q_1, \texttt{abb}, \texttt{A\$})$



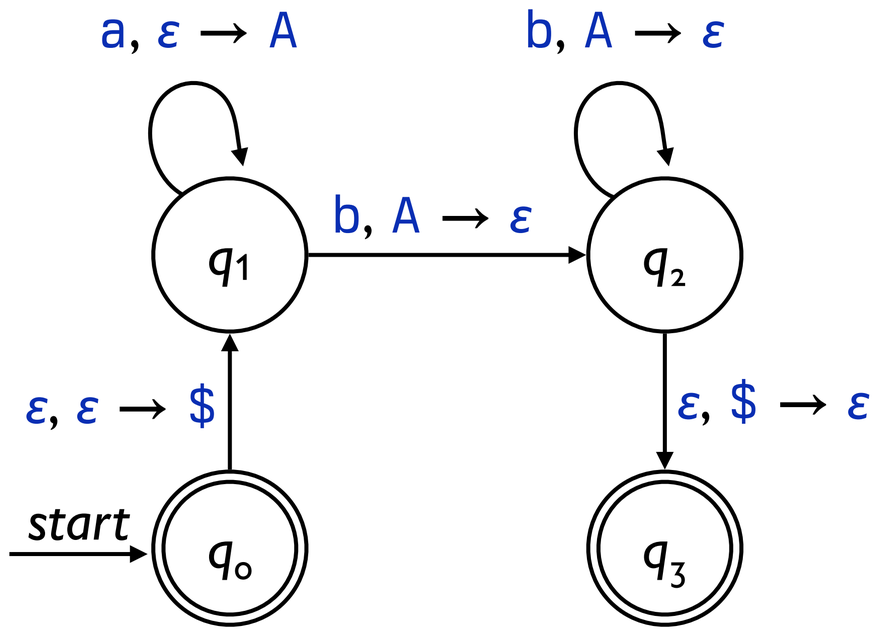
Example

$(q_0, \texttt{aabb}, \varepsilon)$

$\vdash (q_1, \texttt{aabb}, \$)$

$\vdash (q_1, \texttt{abb}, \texttt{A\$})$

$\vdash (q_1, \texttt{bb}, \texttt{AA\$})$



Example

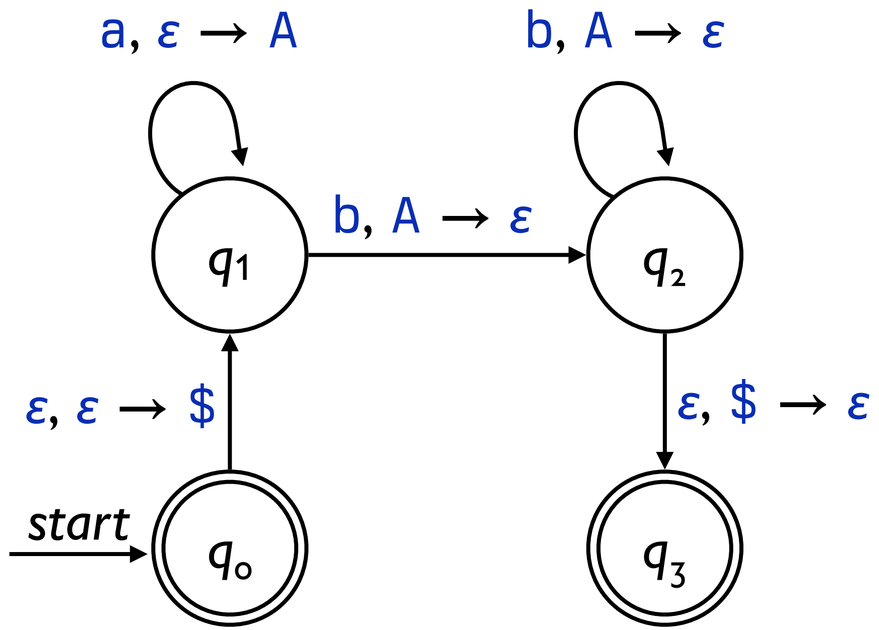
$(q_0, \texttt{aabb}, \varepsilon)$

$\vdash (q_1, \texttt{aabb}, \$)$

$\vdash (q_1, \texttt{abb}, \texttt{A}\$)$

$\vdash (q_1, \texttt{bb}, \texttt{AA}\$)$

$\vdash (q_2, \texttt{b}, \texttt{A}\$)$



Example

$(q_0, \texttt{aabb}, \varepsilon)$

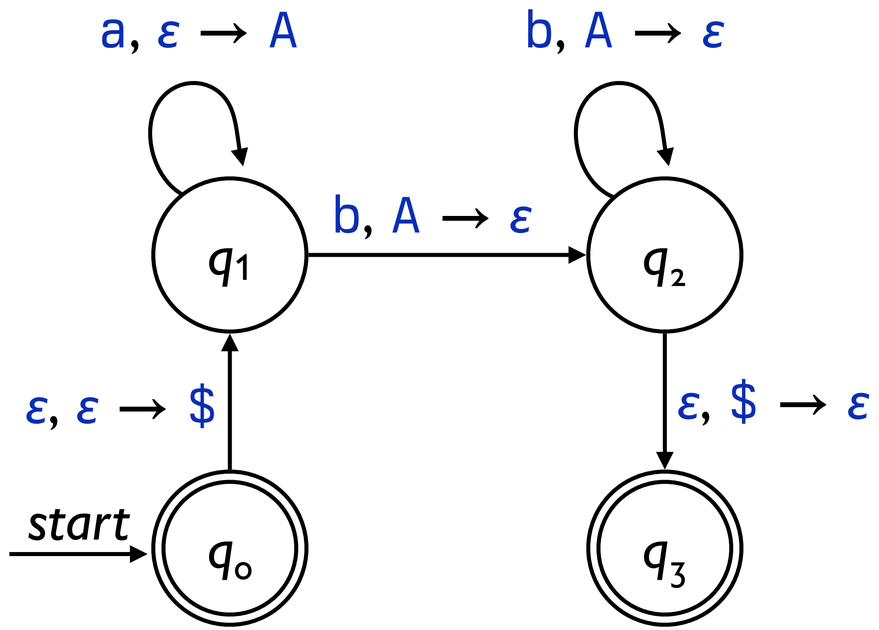
$\vdash (q_1, \texttt{aabb}, \$)$

$\vdash (q_1, \texttt{abb}, \texttt{A}\$)$

$\vdash (q_1, \texttt{bb}, \texttt{AA}\$)$

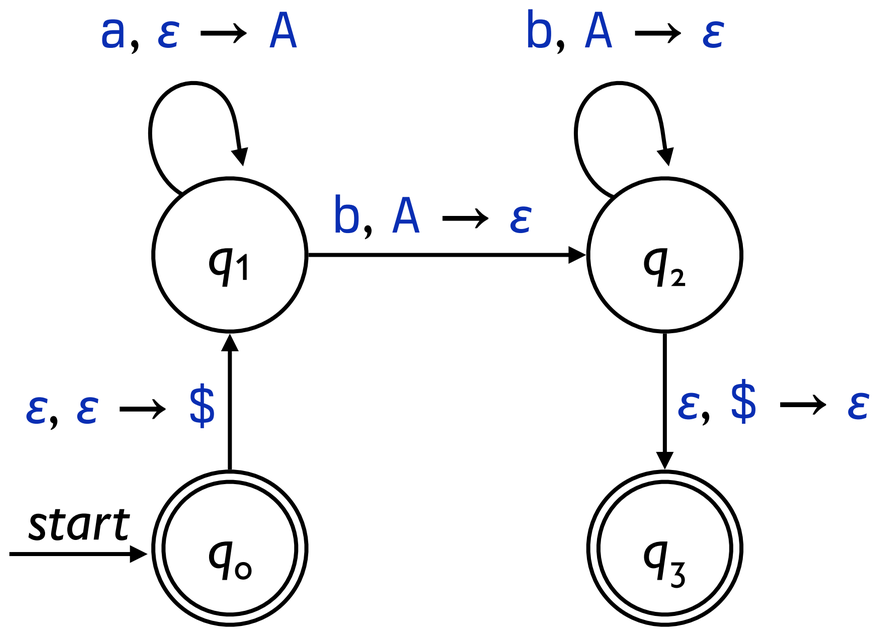
$\vdash (q_2, \texttt{b}, \texttt{A}\$)$

$\vdash (q_2, \varepsilon, \$)$



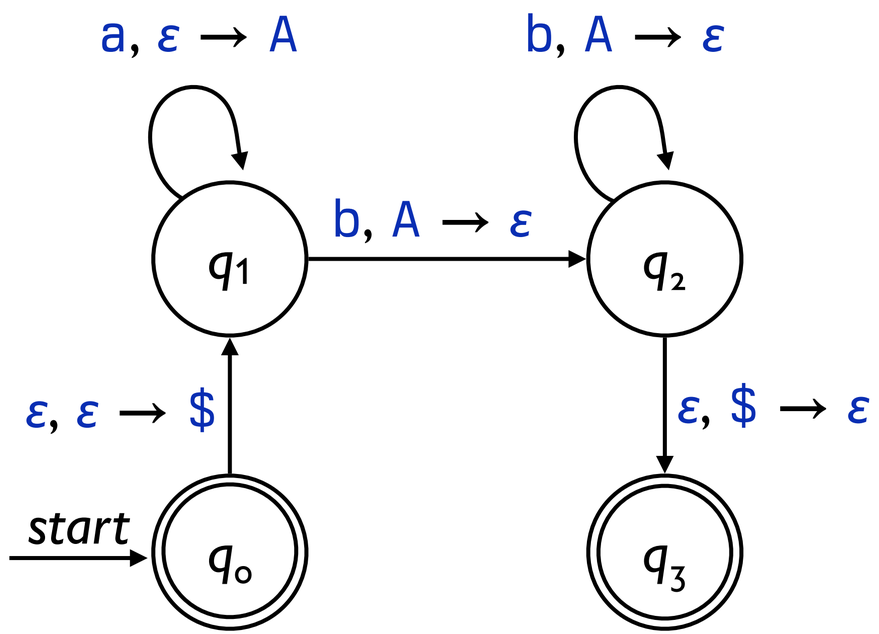
Example

$$
\begin{aligned}
&(q_0, \texttt{aabb}, \varepsilon) \\
\vdash\ &(q_1, \texttt{aabb}, \$) \\
\vdash\ &(q_1, \texttt{abb}, \texttt{A}\$) \\
\vdash\ &(q_1, \texttt{bb}, \texttt{AA}\$) \\
\vdash\ &(q_2, \texttt{b}, \texttt{A}\$) \\
\vdash\ &(q_2, \varepsilon, \$) \\
\vdash\ &(q_3, \varepsilon, \varepsilon)
\end{aligned}
$$



Example

$$(q_0,\ aabb,\ \varepsilon)$$
$$\overset{*}{\vdash} (q_3,\ \varepsilon,\ \varepsilon)$$



Describing the moves of the PDA

If $\delta(q, a, X)$ contains (p, a), then we say

$$(q, aw, X\beta) \vdash (p, w, a\beta)$$

$\vdash$ *is the "goes to" relation*

We extend this relation to $\overset{*}{\vdash}$ for zero or more moves:

An input string w is accepted if $(q_0, w, \varepsilon) \overset{*}{\vdash} (p, \varepsilon, \beta)$ for any accepting state p and any stack contents β.

Equivalence of CFGs and PDAs

We defined the context-free languages as follows:

> A language is *context-free* if and only if there is some context-free grammar (CFG) that generates it.

An important (but non-obvious!) theorem is that

> the class of languages generated by CFGs and
> the class of languages recognized by PDAs

are the *same*.

THEOREM A language is context-free if and only if there is some pushdown automaton (PDA) that recognizes it.

We need to prove both parts of the "if and only if":

CFG → PDA:

If *L* is context-free (i.e., some CFG generates it), then there is a PDA that recognizes *L*.

PDA → CFG:

If a PDA recognizes *L*, then *L* is context-free (i.e., some CFG generates it).

CFG → PDA

THEOREM If G is a CFG for a language L, then there exists a PDA for L as well.

PROOF IDEA Proof by construction.

Build a PDA that simulates expanding out the CFG from the start symbol to some particular string.

Use the stack to hold the part of the string we haven't matched yet.

Example: CFG

Let $\Sigma = \{1, \geq\}$.

Let $GE = \{1^m \geq 1^n \mid m, n \in \mathbb{N}_0 \text{ and } m \geq n\}$.

$$111 \geq 11 \in GE$$

$$11 \geq 11 \in GE$$

$$1111 \geq 11 \in GE$$

$$\geq \in GE$$

Example: CFG

Let $\Sigma = \{1, \geq\}$.

Let $GE = \{1^m \geq 1^n \mid m, n \in \mathbb{N}_0 \text{ and } m \geq n\}$.

$$111 \geq 11 \in GE$$

$$11 \geq 11 \in GE$$

$$1111 \geq 11 \in GE$$

$$\geq \in GE$$

What's a CFG to generate *GE*?

Example: CFG

Let $\Sigma = \{1, \geq\}$.

Let $GE = \{1^m \geq 1^n \mid m, n \in \mathbb{N}_0 \text{ and } m \geq n\}$.

$$111\geq 11 \in GE$$

$$11\geq 11 \in GE$$

$$1111\geq 11 \in GE$$

$$\geq \in GE$$

What's a CFG to generate *GE*?

$$S \rightarrow 1S1 \mid 1S \mid \geq$$

Example: CFG

Let $\Sigma = \{1, \geq\}$.
Let $GE = \{1^m \geq 1^n \mid m, n \in \mathbb{N}_0 \text{ and } m \geq n\}$.

$$111\geq 11 \in GE$$

$$11\geq 11 \in GE$$

$$1111\geq 11 \in GE$$

$$\geq \in GE$$

What's a CFG to generate *GE*?

$$S \rightarrow 1S1 \mid 1S \mid \geq$$

How would we build a PDA for *GE*?

Example: Equivalent PDA

$S \to 1S1$

$S \to 1S$

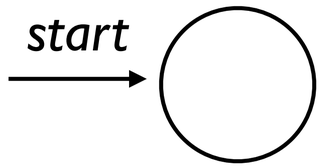
$S \to \geq$

Example: Equivalent PDA

$S \to 1S1$

$S \to 1S$

$S \to \geq$

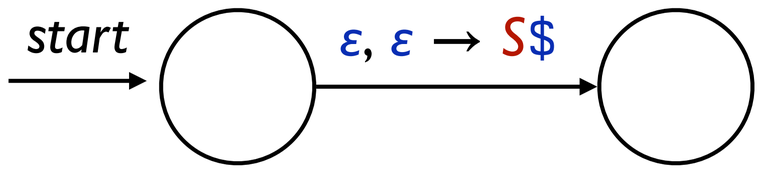


Example: Equivalent PDA

$S \rightarrow 1S1$

$S \rightarrow 1S$

$S \rightarrow \geq$



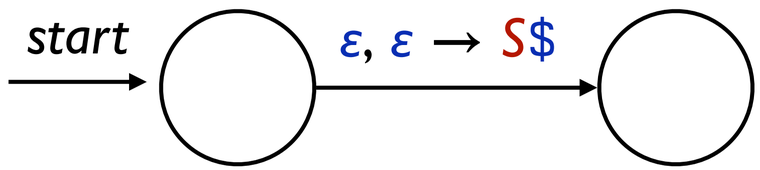
Example: Equivalent PDA

$S \rightarrow 1S1$

$S \rightarrow 1S$

$S \rightarrow \geq$

We start by pushing a bottom-of-stack symbol $ and, on top of it, the start symbol of the grammar (S), so we can begin applying productions.



Example: Equivalent PDA

$S \to 1S1$

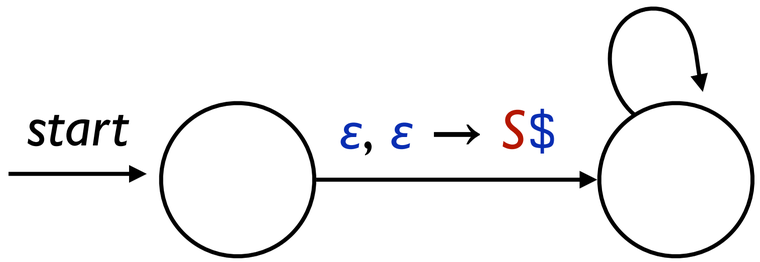
$S \to 1S$

$S \to \geq$

$\varepsilon, S \to 1S1$

$\varepsilon, S \to 1S$

$\varepsilon, S \to \geq$

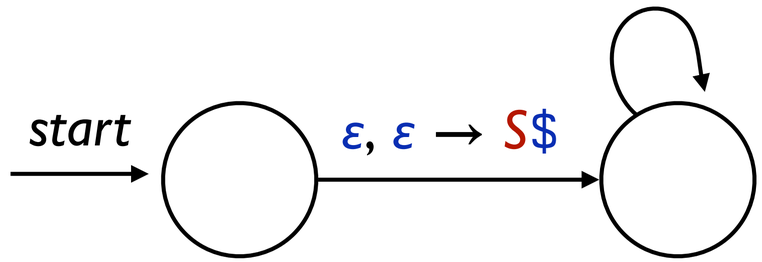


Example: Equivalent PDA

$S \to 1S1$

$S \to 1S$

$S \to \geq$



ε, S → 1S1
ε, S → 1S
ε, S → ≥

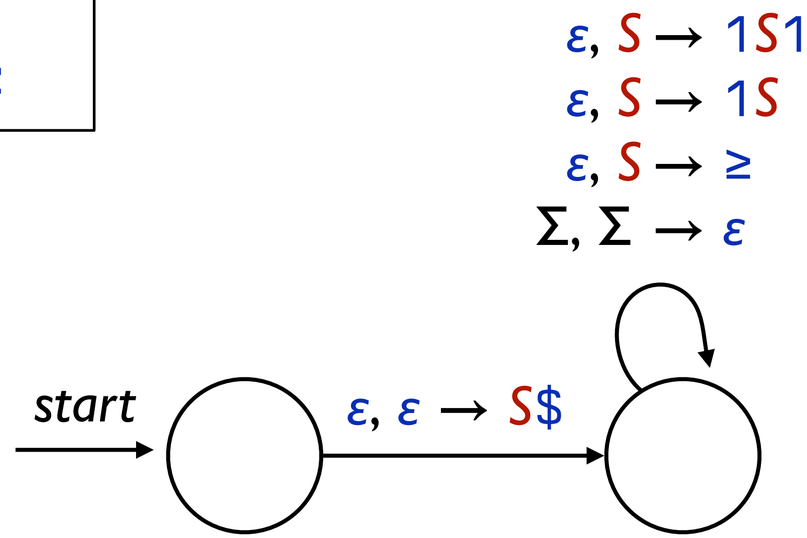
These transitions allow us to nondeterministically guess which production to use when the top of the stack is a nonterminal.

Example: Equivalent PDA

$S \to 1S1$

$S \to 1S$

$S \to \geq$

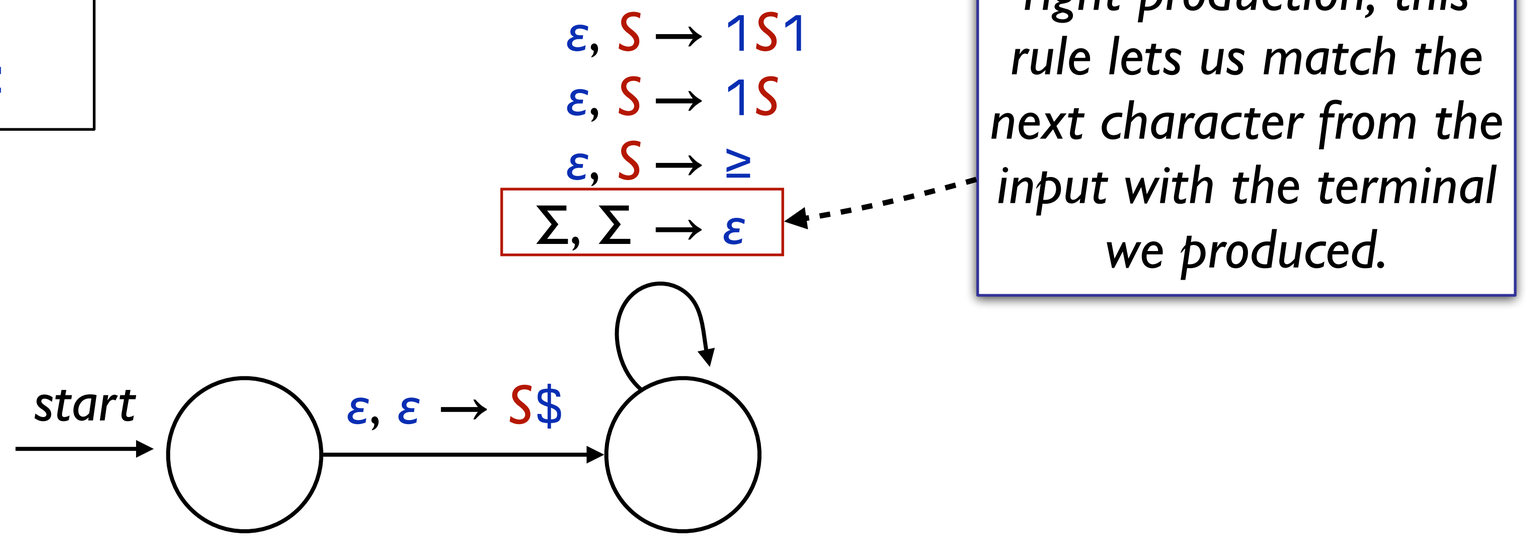


Example: Equivalent PDA

$S \rightarrow 1S1$

$S \rightarrow 1S$

$S \rightarrow \geq$



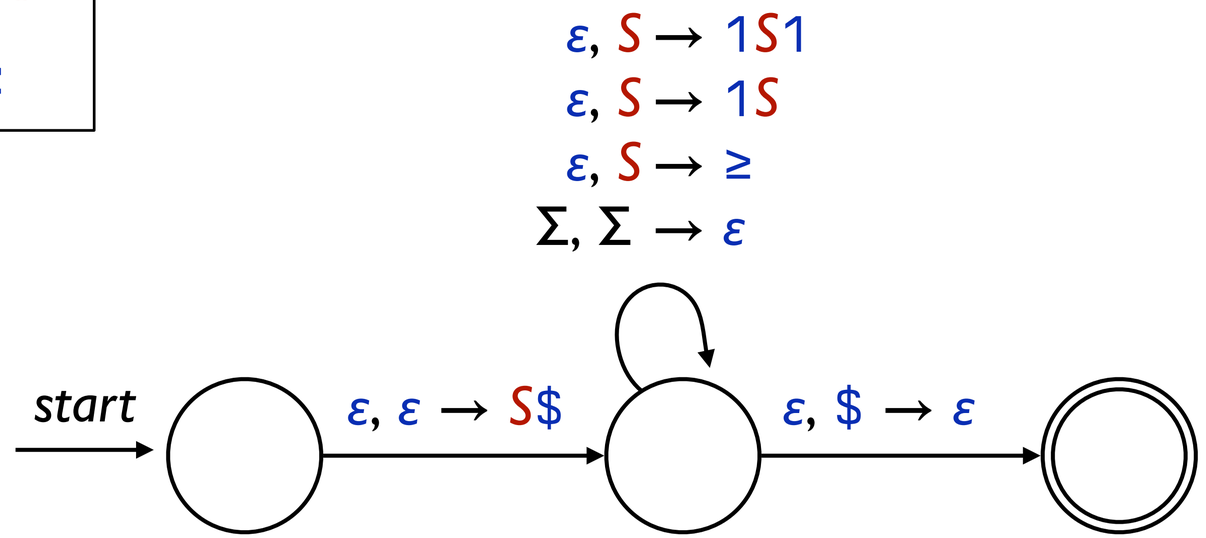
Once we've guessed the right production, this rule lets us match the next character from the input with the terminal we produced.

Example: Equivalent PDA

$S \to 1S1$

$S \to 1S$

$S \to \geq$

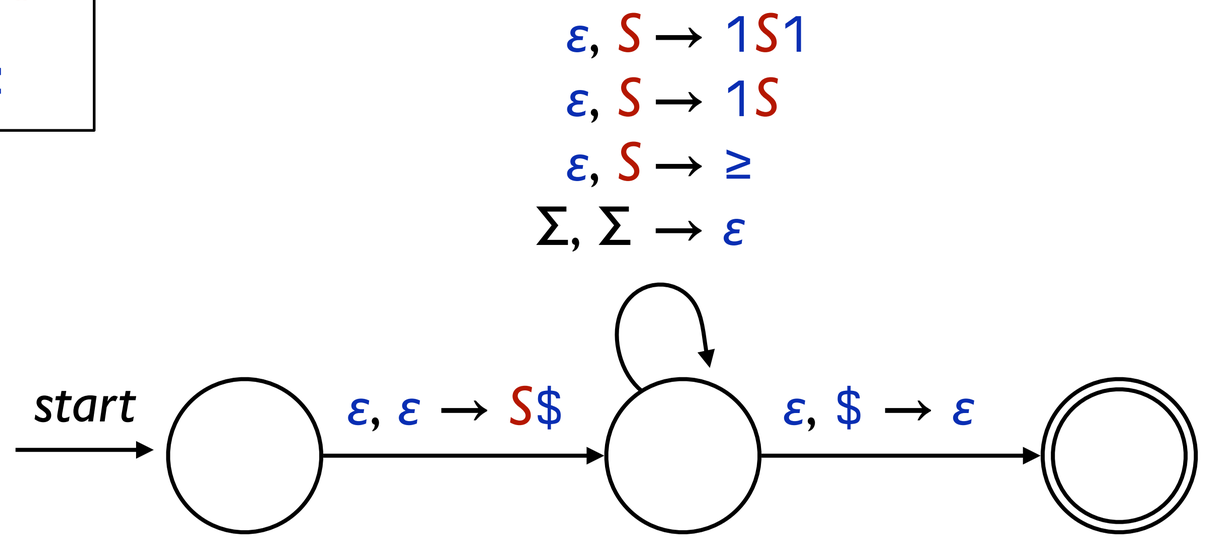


Example: Equivalent PDA

$S \to 1S1$

$S \to 1S$

$S \to \geq$

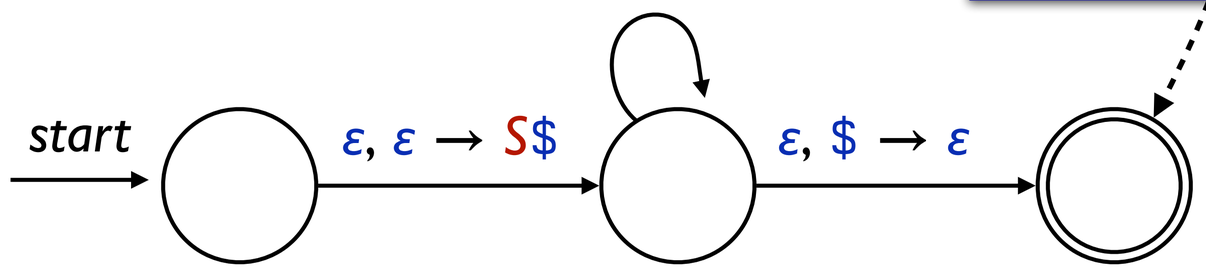


Example: Equivalent PDA

$S \rightarrow 1S1$

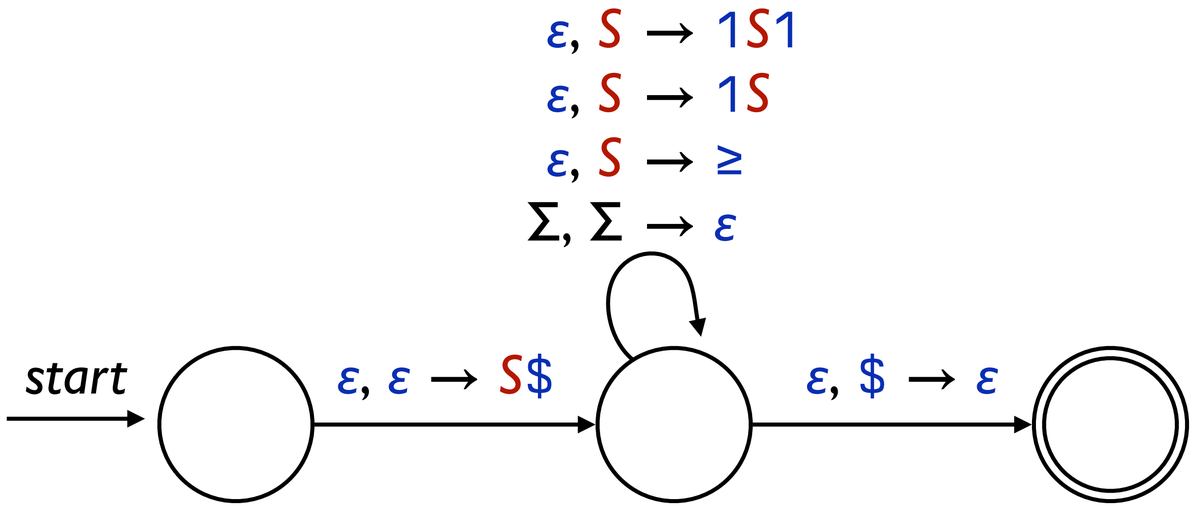
$S \rightarrow 1S$

$S \rightarrow \geq$



Once we've fully expanded out all nonterminals and matched all the terminals on the stack, we can transition into the accept state

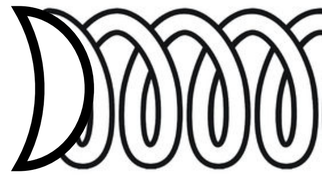
Example: Equivalent PDA



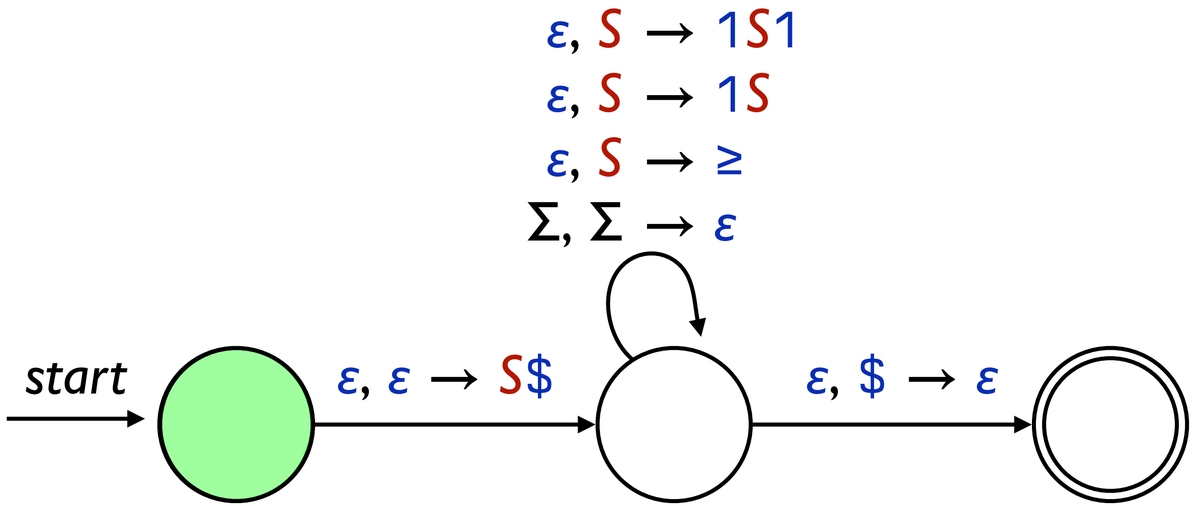
Input

Stack

1 1 1 ≥ 1 1

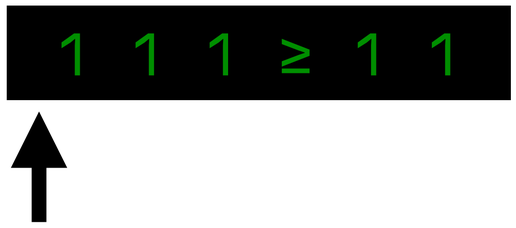


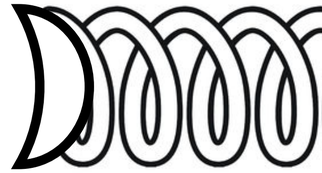
Example: Equivalent PDA



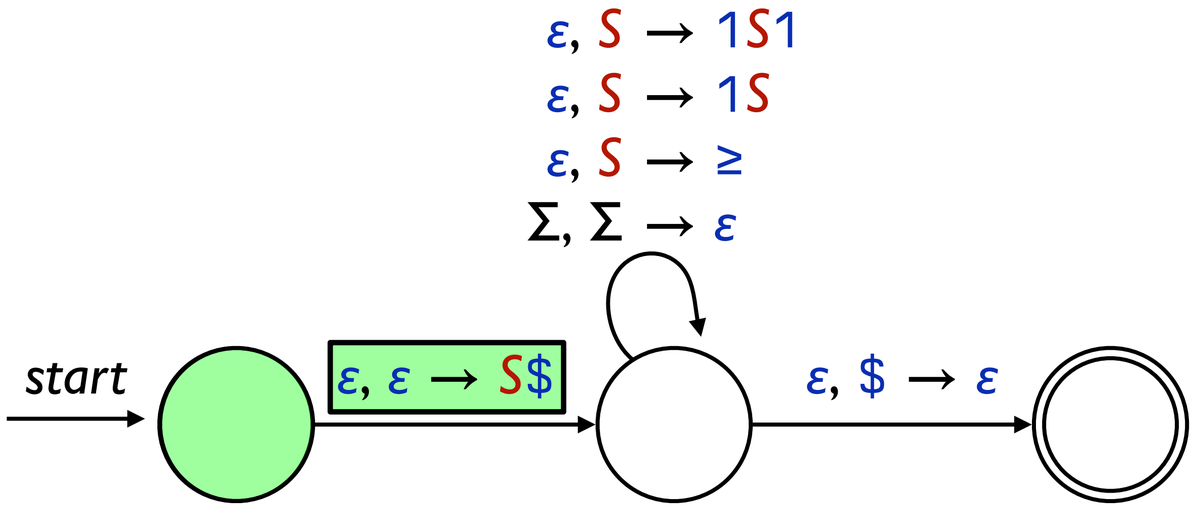
Input

Stack



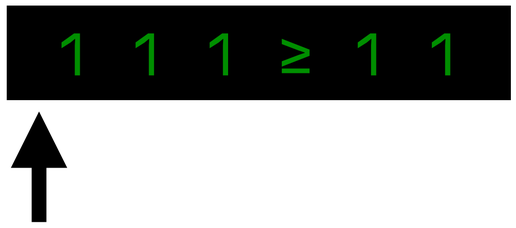


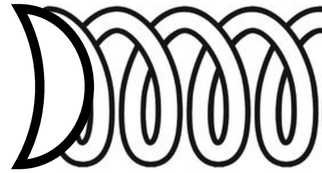
Example: Equivalent PDA



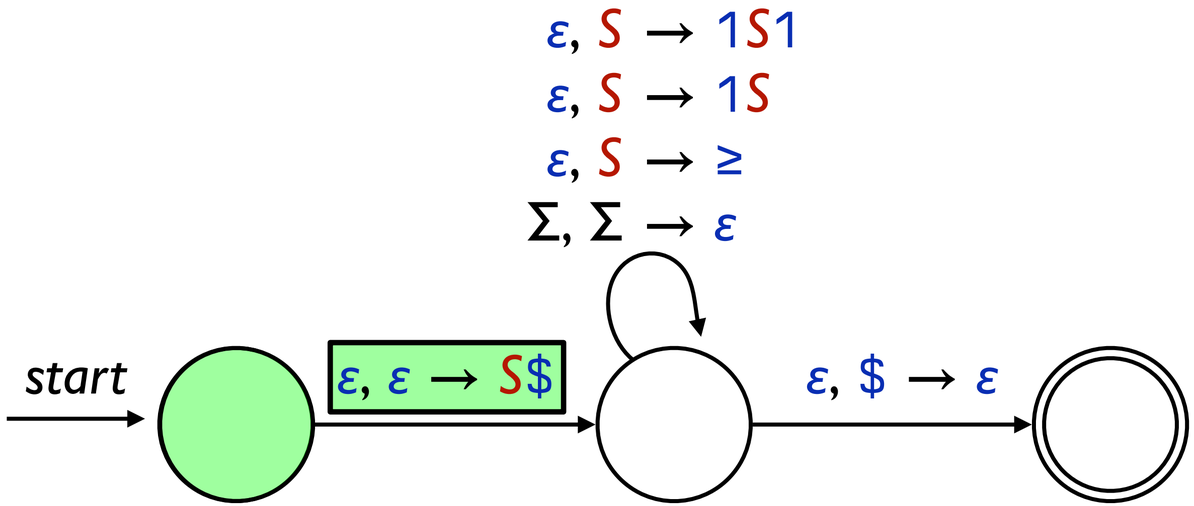
Input

Stack

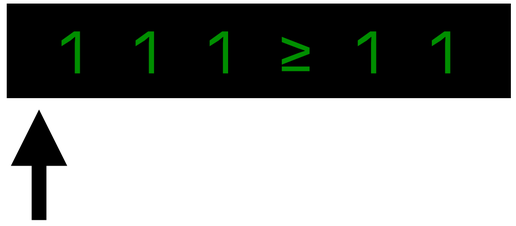




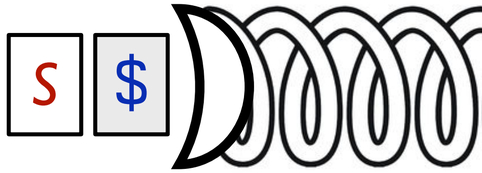
Example: Equivalent PDA



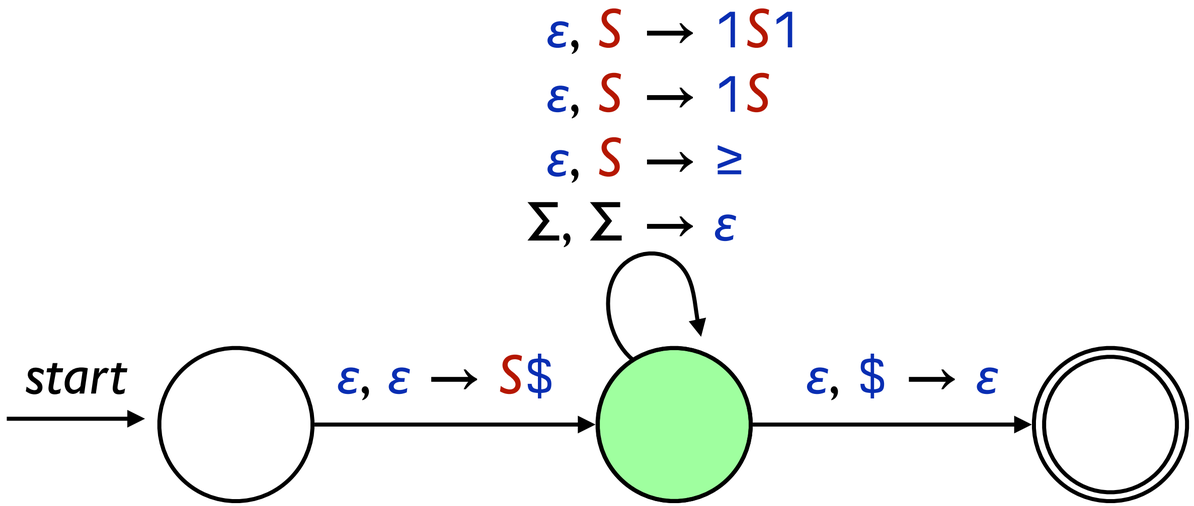
Input



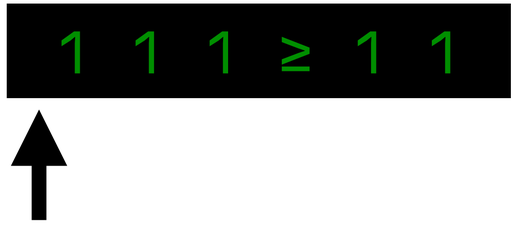
Stack



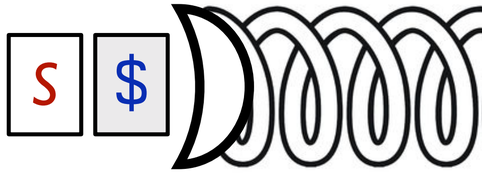
Example: Equivalent PDA



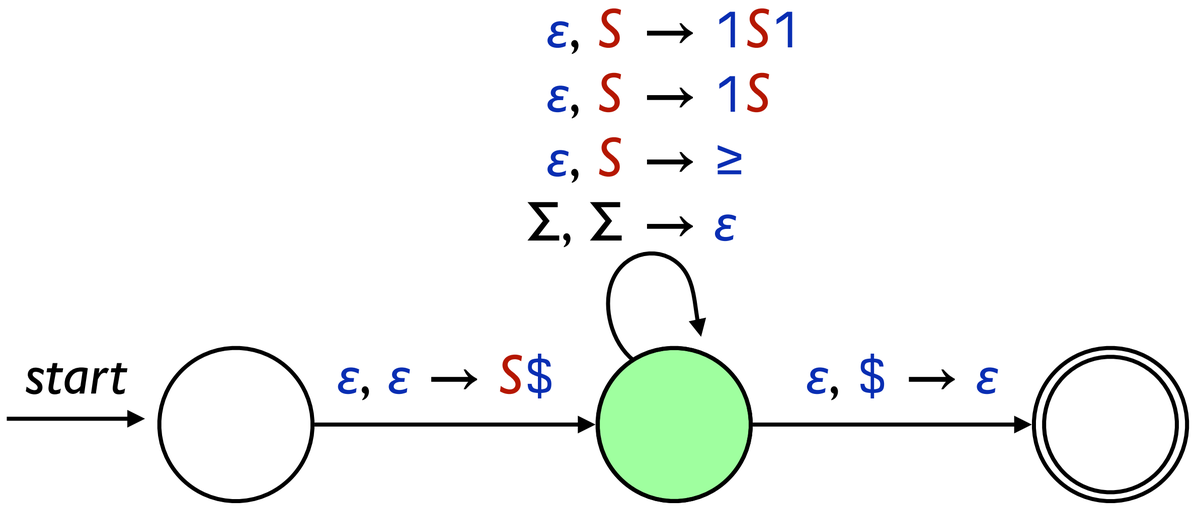
Input



Stack

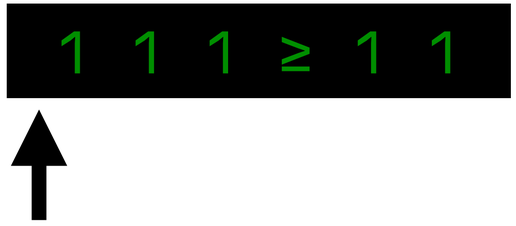


Example: Equivalent PDA

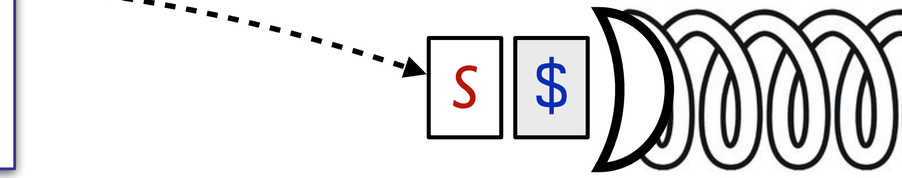


The symbol on top of the stack is a nonterminal, so we guess which production to use.

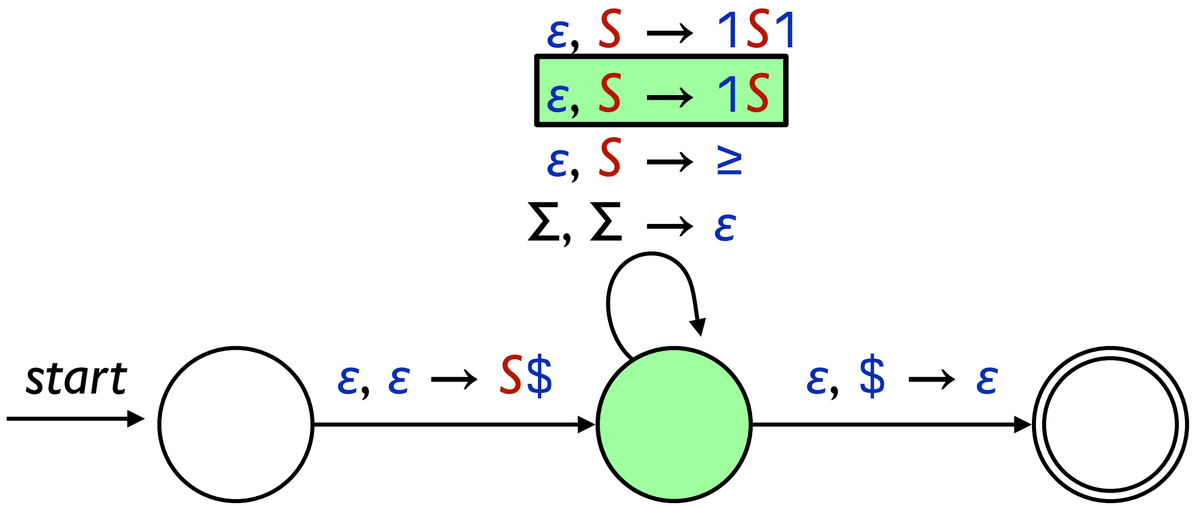
Input



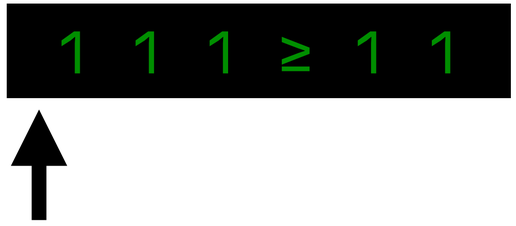
Stack



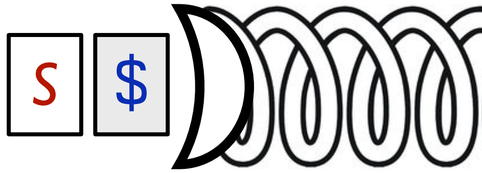
Example: Equivalent PDA



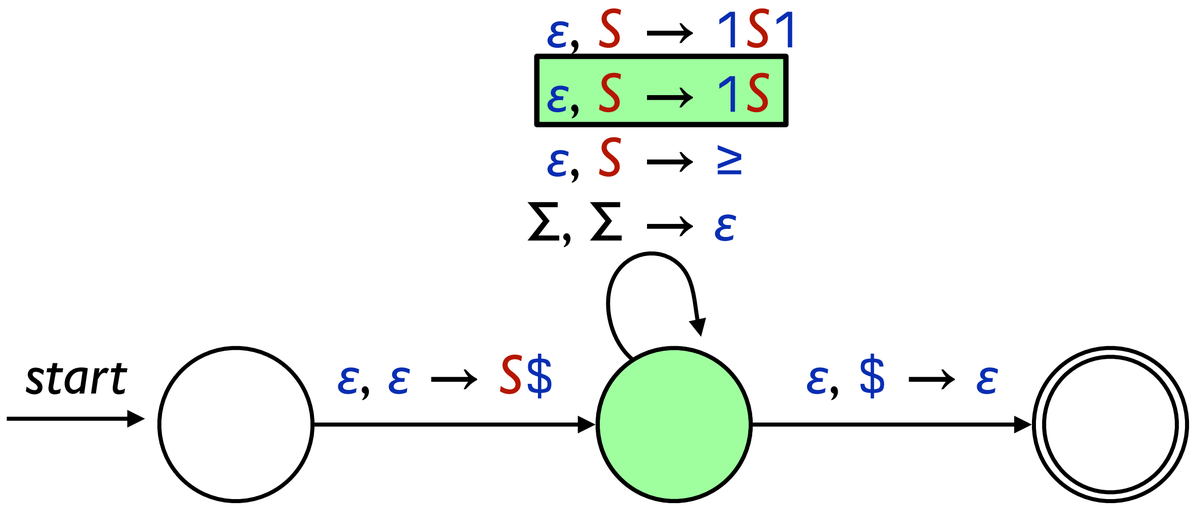
Input



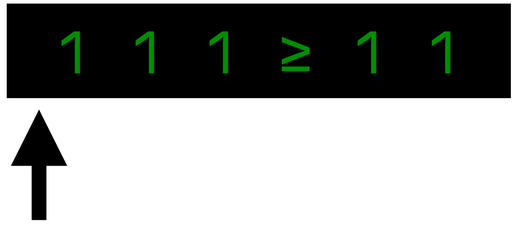
Stack



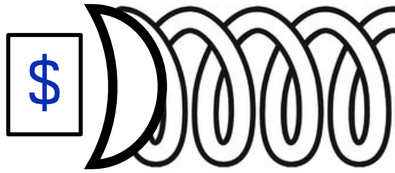
Example: Equivalent PDA



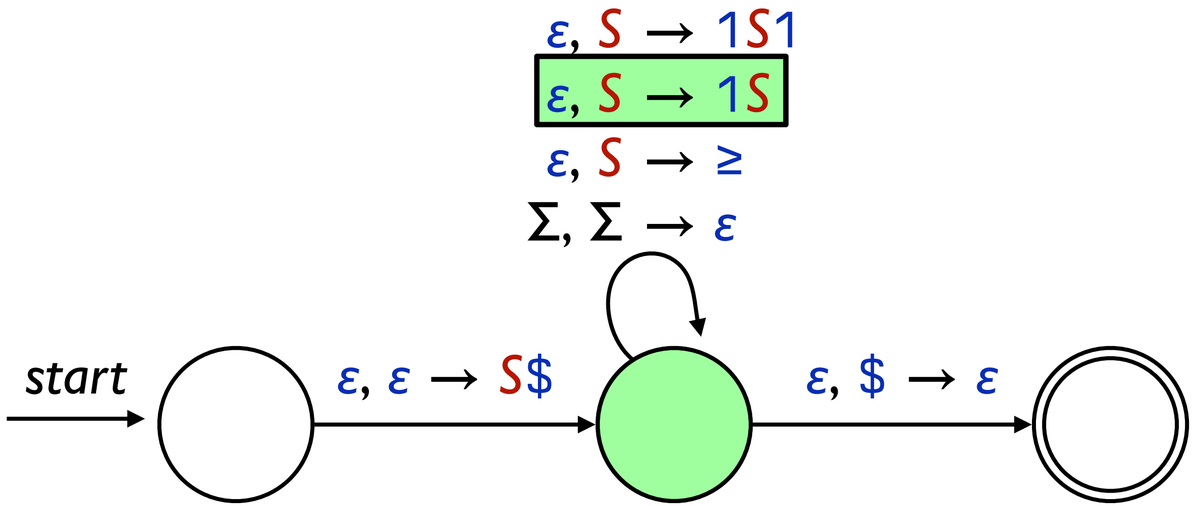
Input



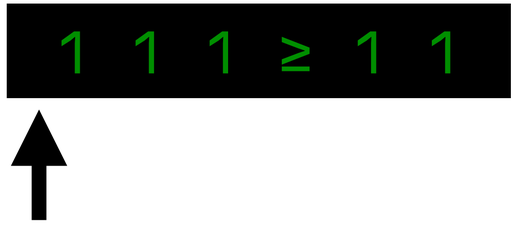
Stack



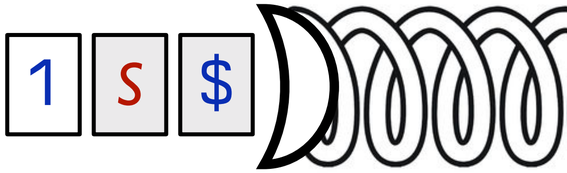
Example: Equivalent PDA



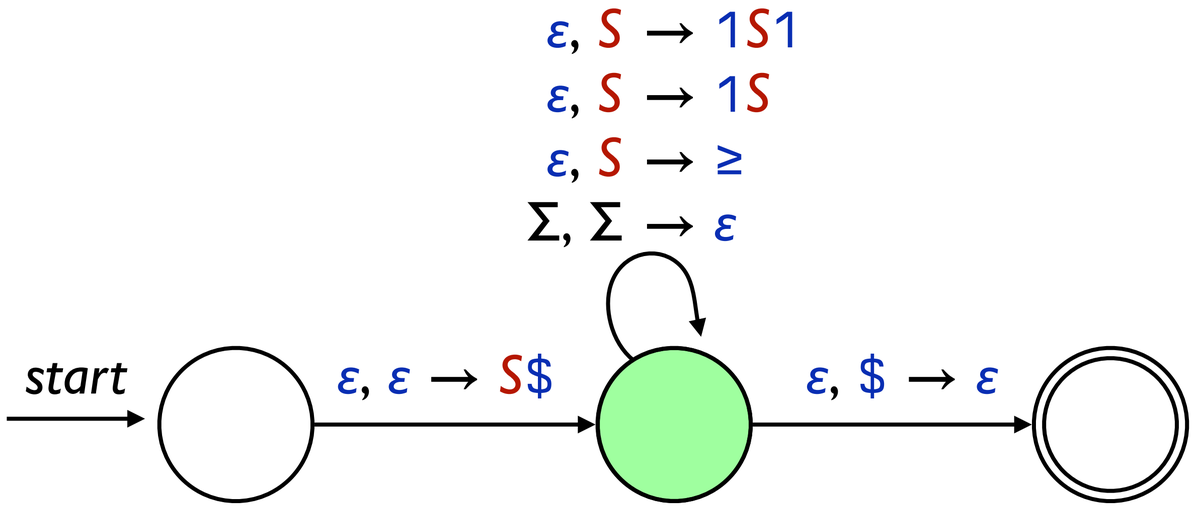
Input



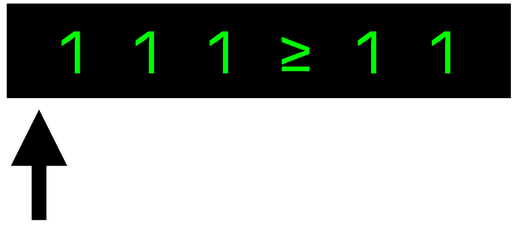
Stack



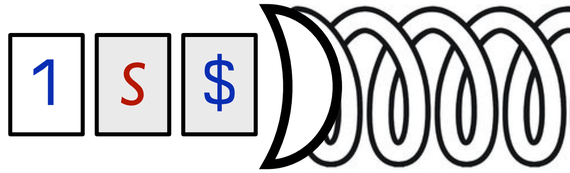
Example: Equivalent PDA



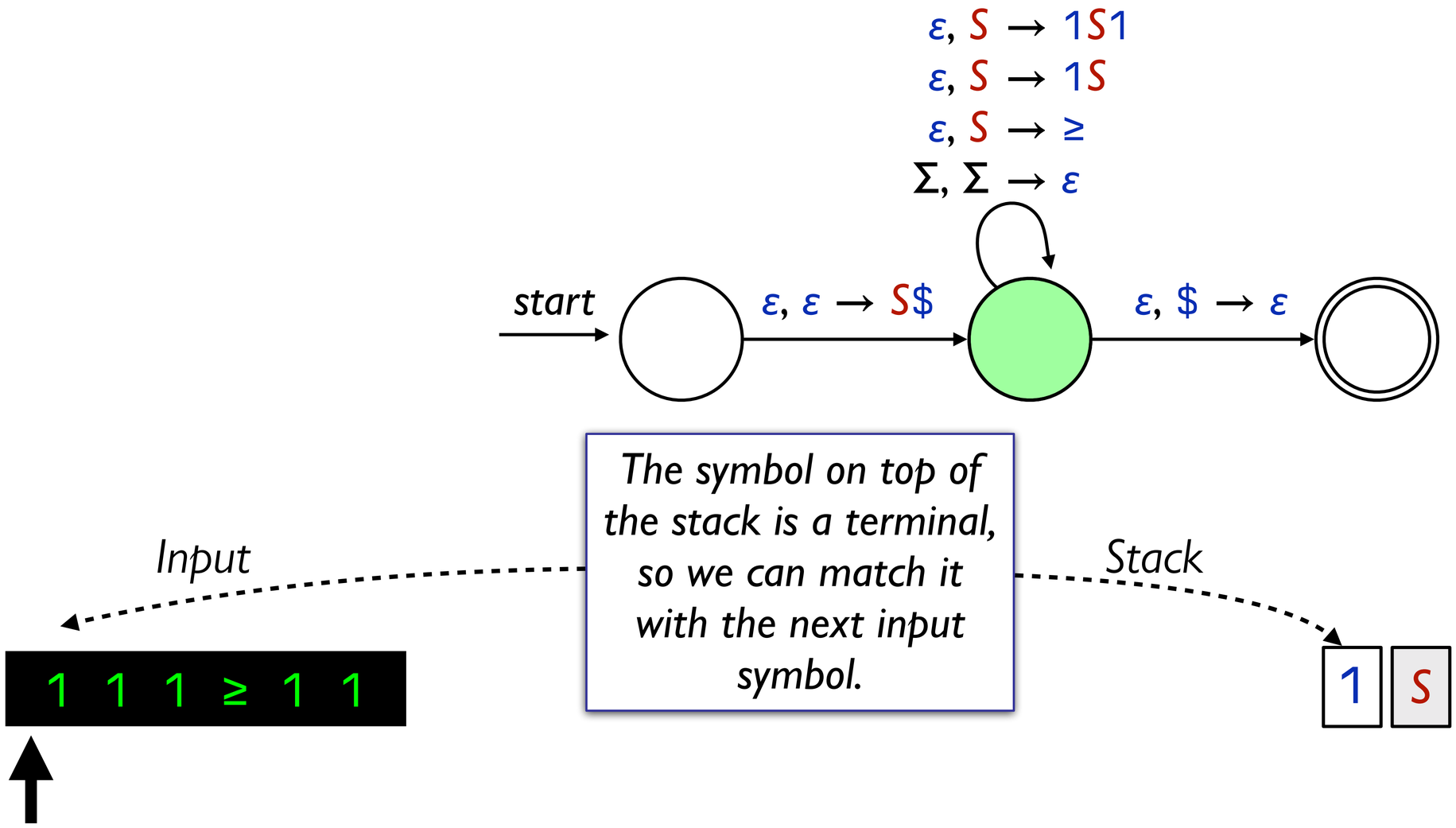
Input



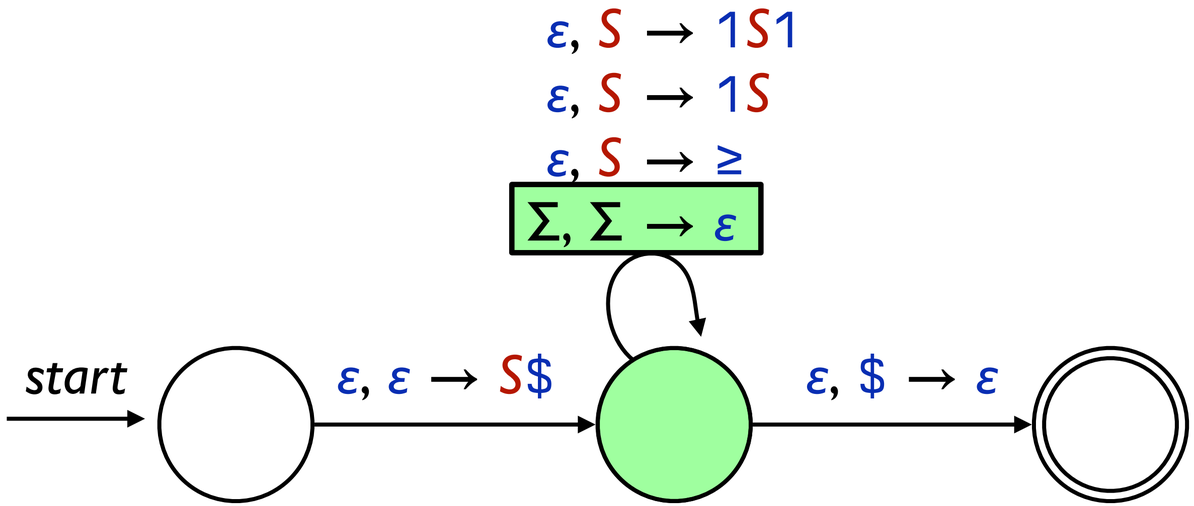
Stack



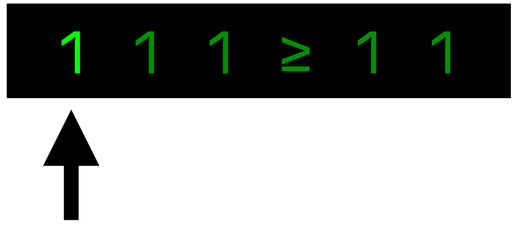
Example: Equivalent PDA



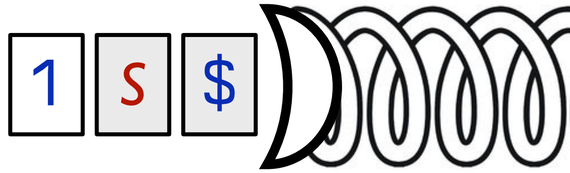
Example: Equivalent PDA



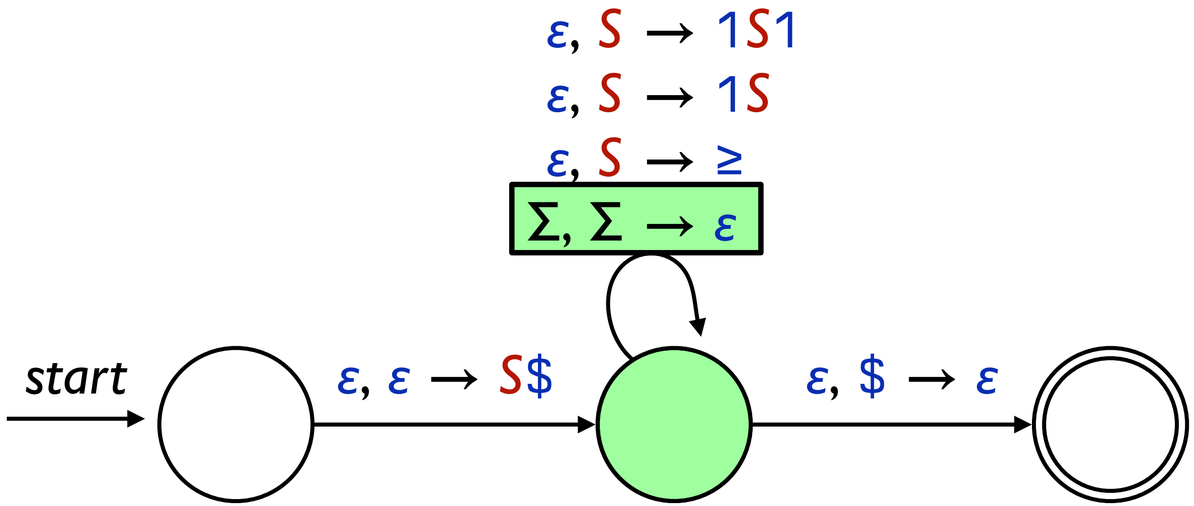
Input



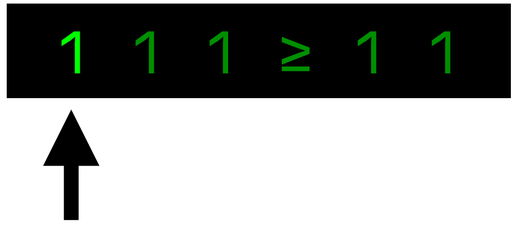
Stack



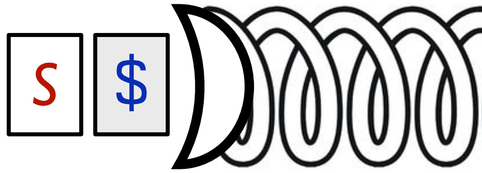
Example: Equivalent PDA



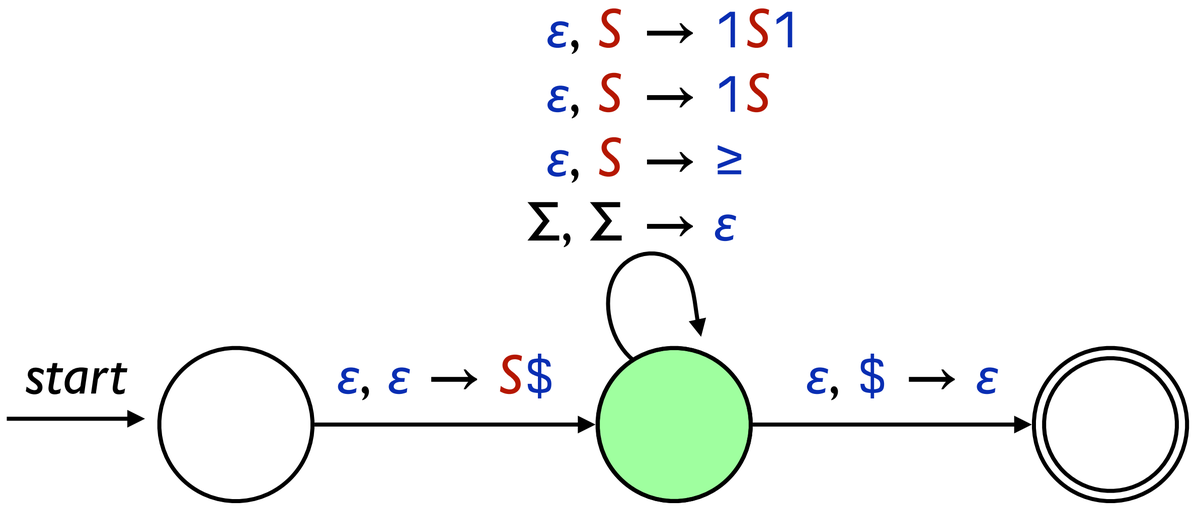
Input



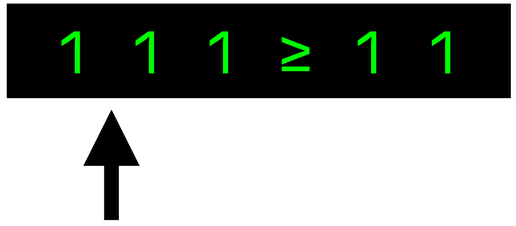
Stack



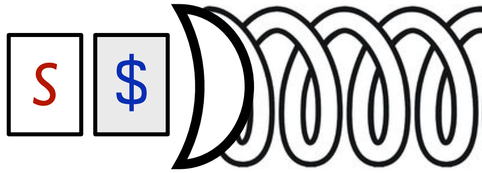
Example: Equivalent PDA



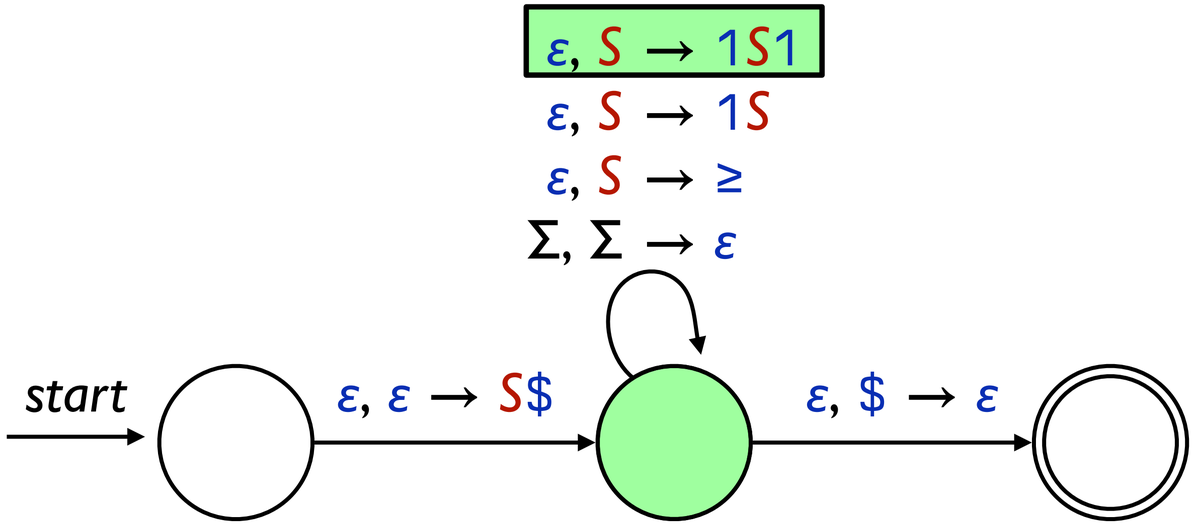
Input



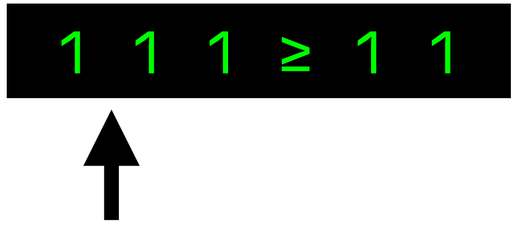
Stack



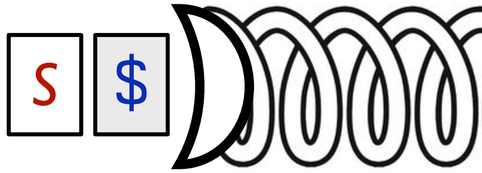
Example: Equivalent PDA



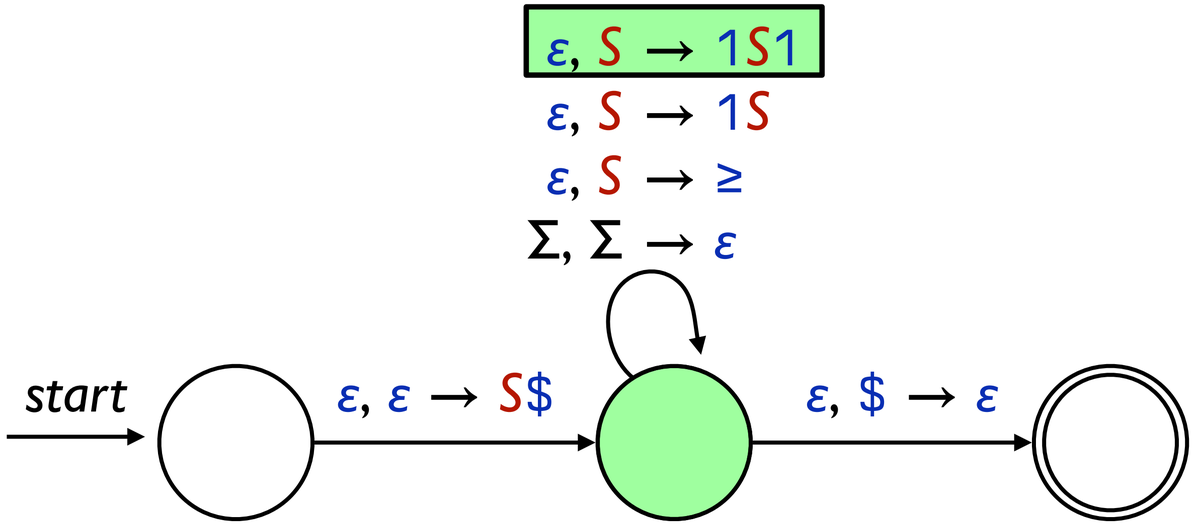
Input



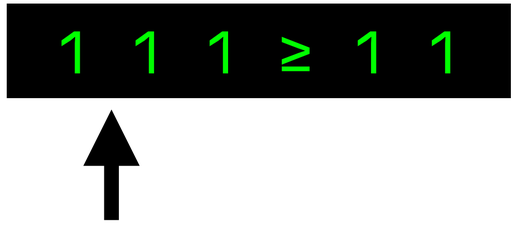
Stack



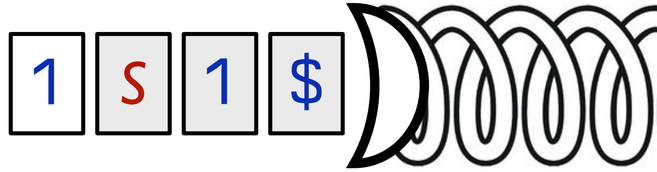
Example: Equivalent PDA



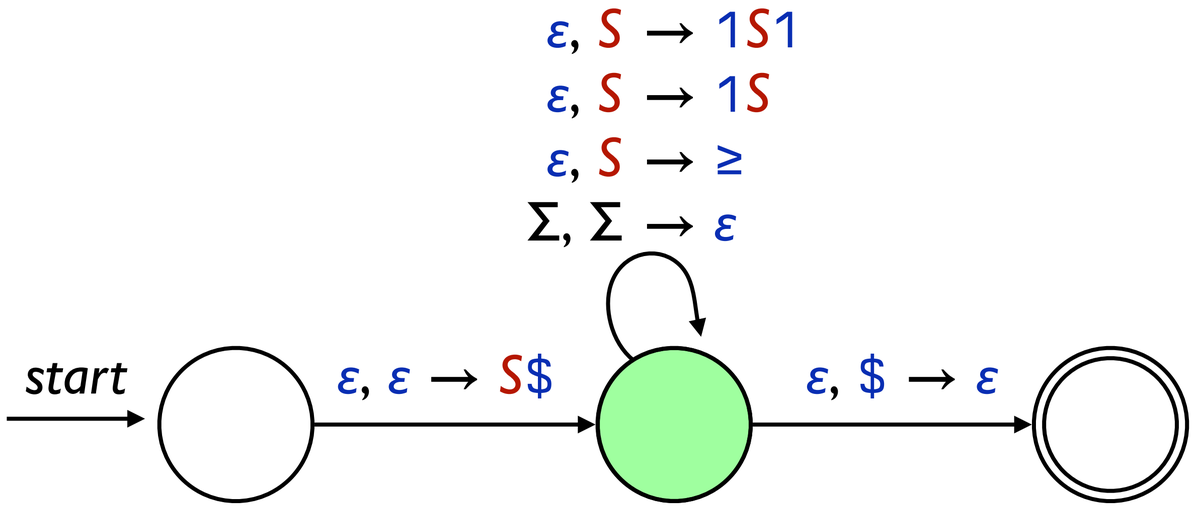
Input



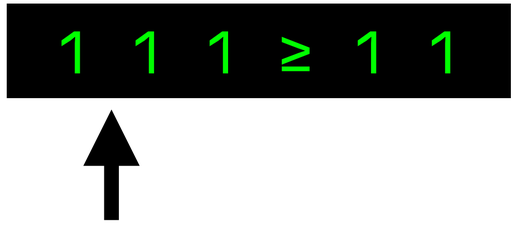
Stack



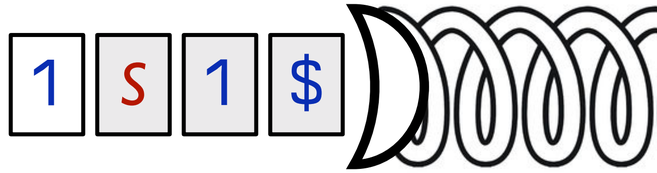
Example: Equivalent PDA



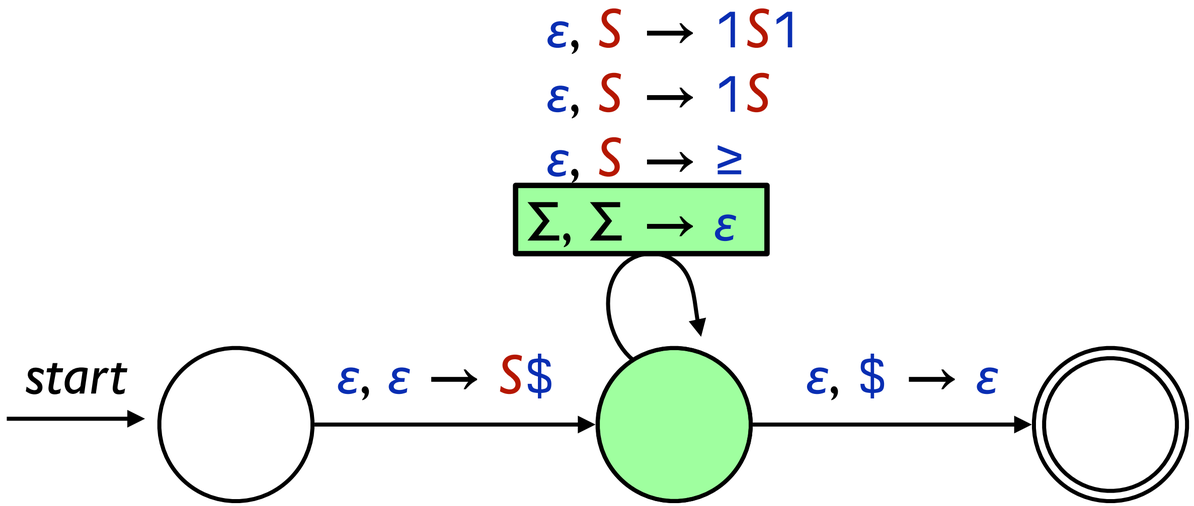
Input



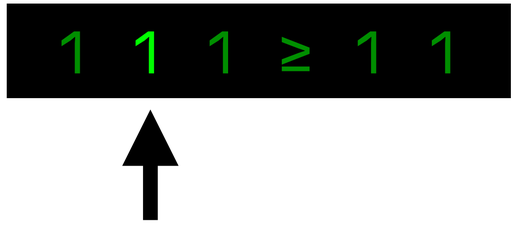
Stack



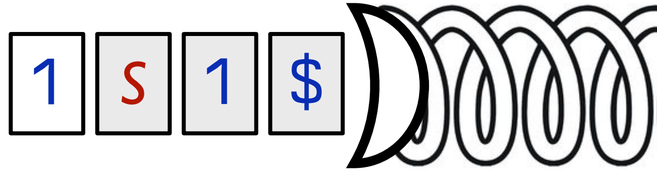
Example: Equivalent PDA



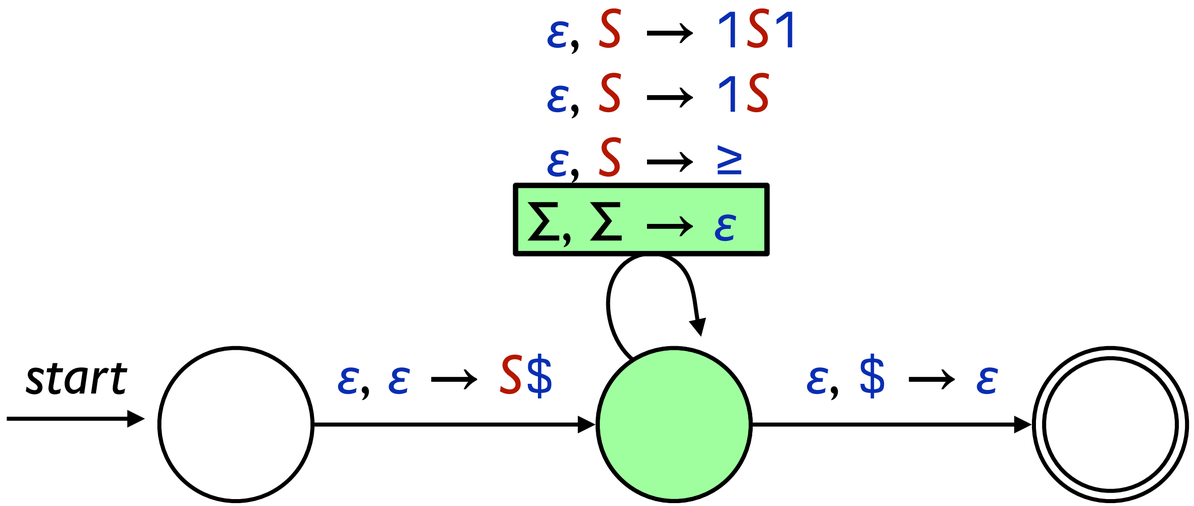
Input



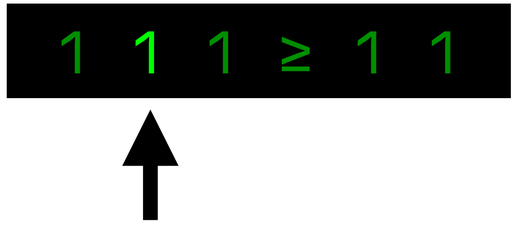
Stack



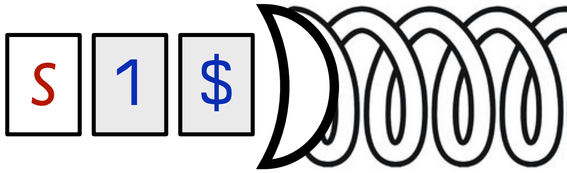
Example: Equivalent PDA



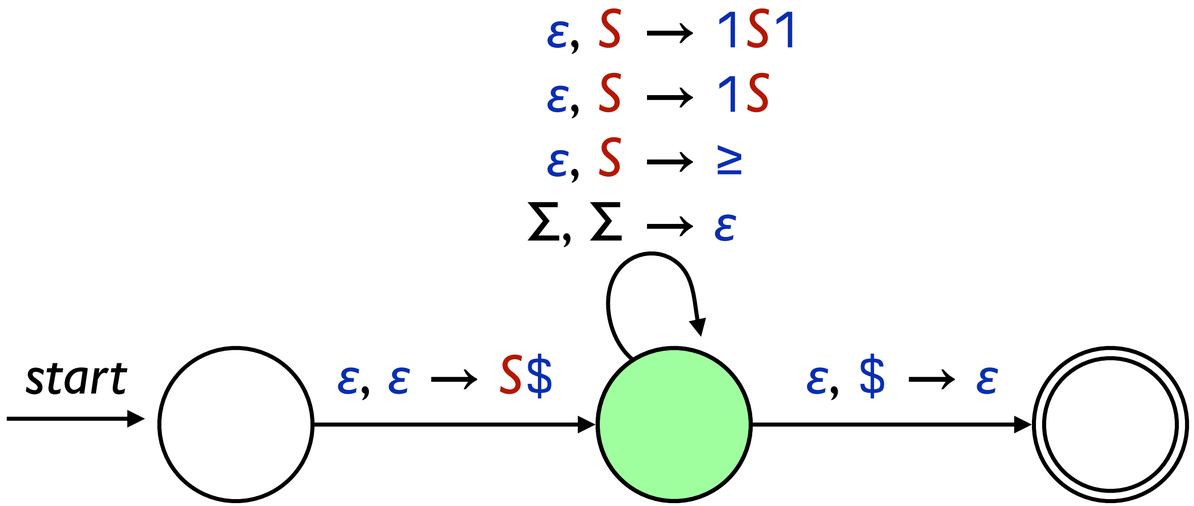
Input



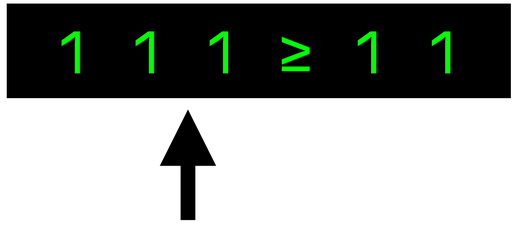
Stack



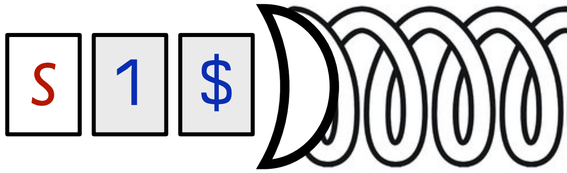
Example: Equivalent PDA



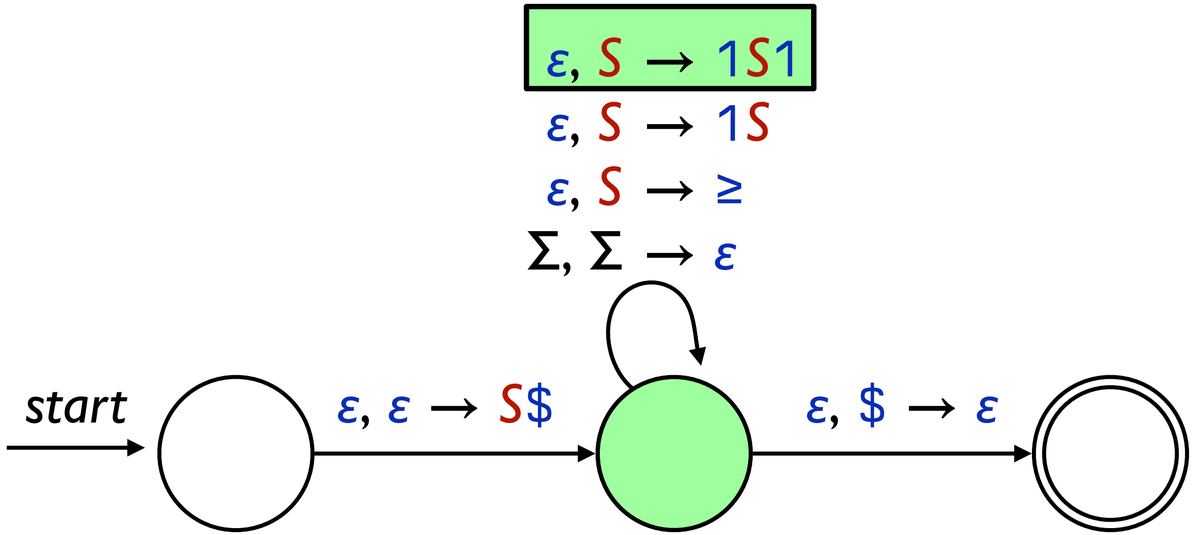
Input



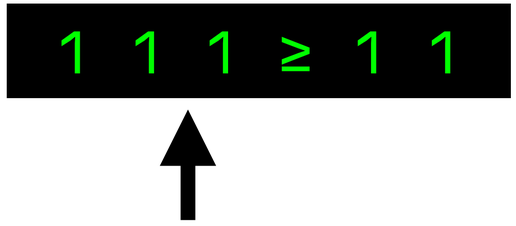
Stack



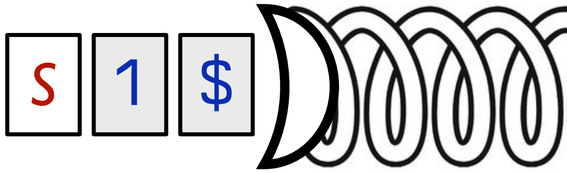
Example: Equivalent PDA



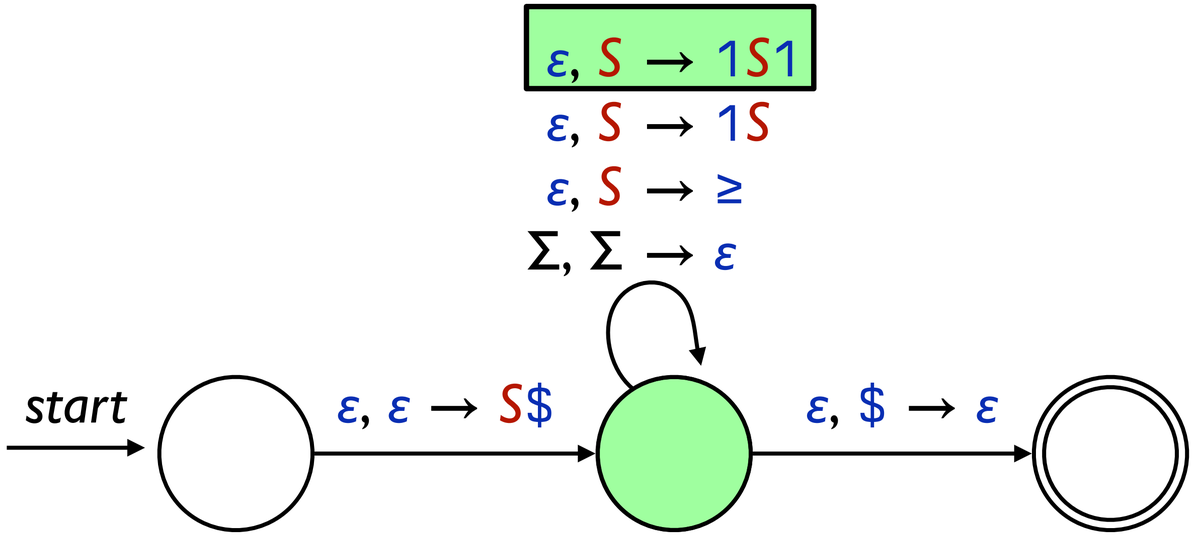
Input



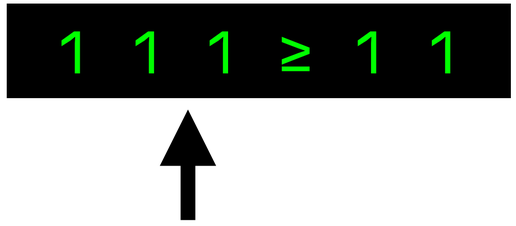
Stack



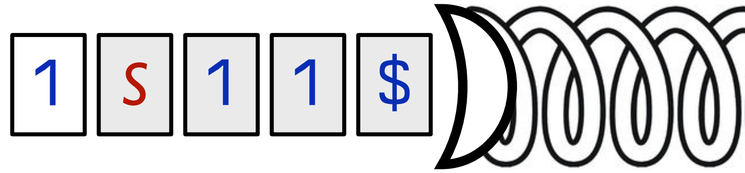
Example: Equivalent PDA



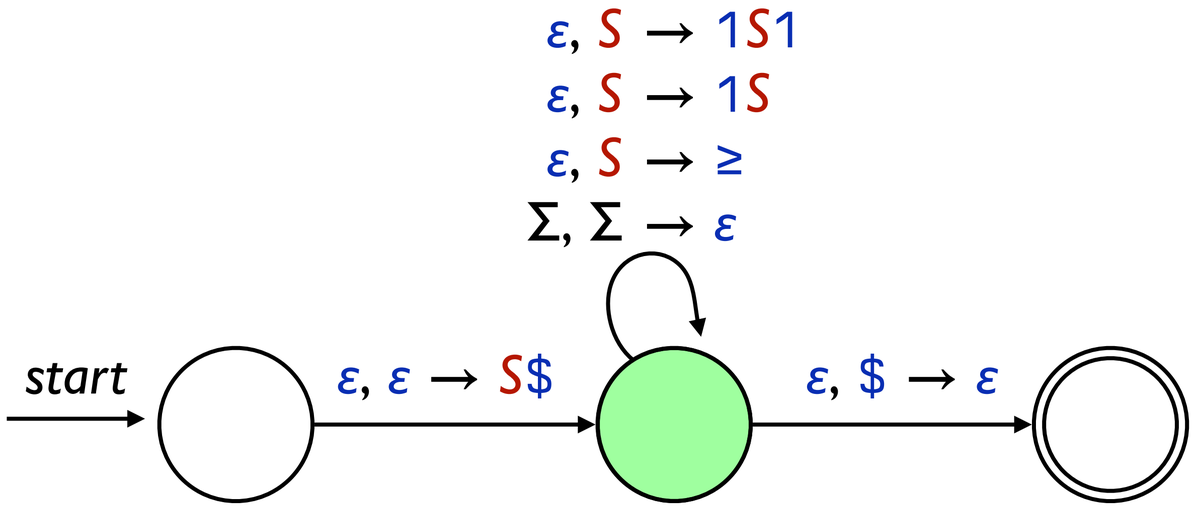
Input



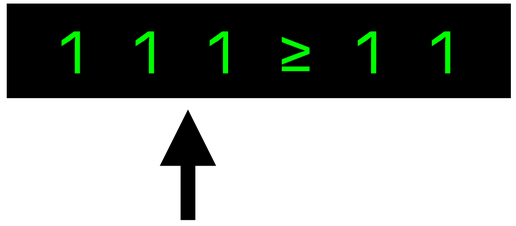
Stack



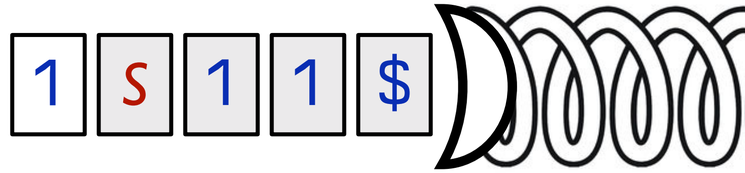
Example: Equivalent PDA



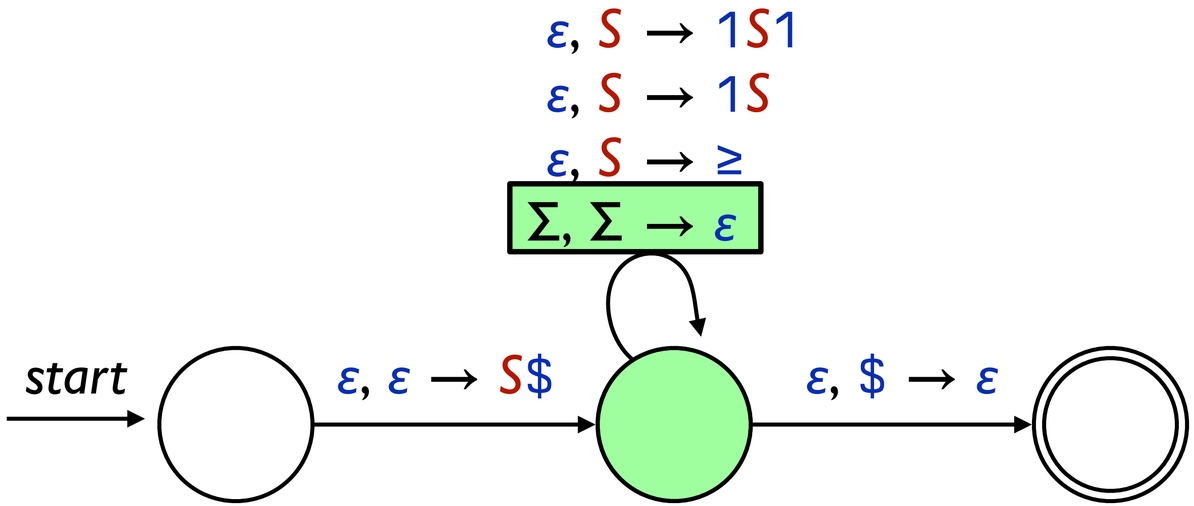
Input



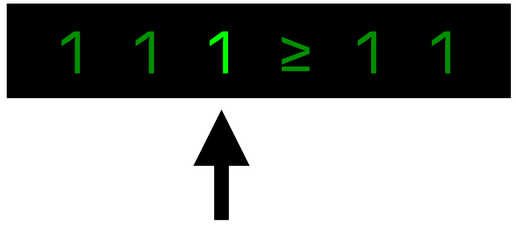
Stack



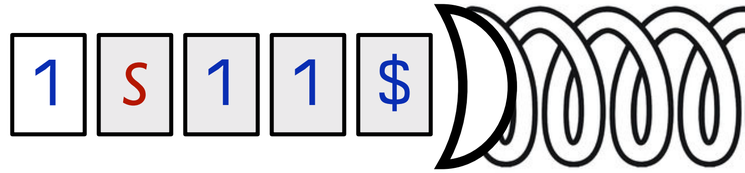
Example: Equivalent PDA



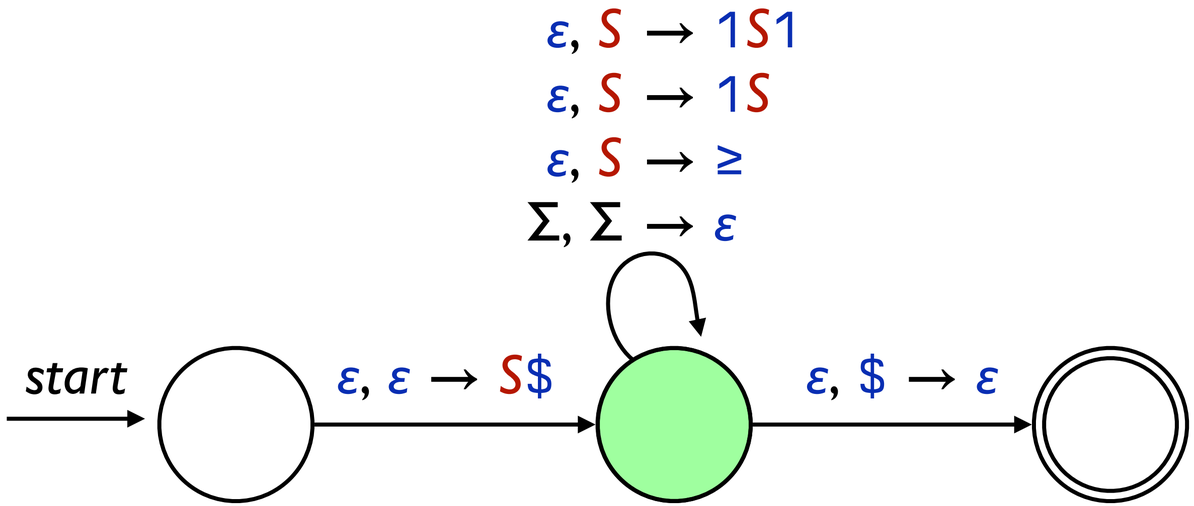
Input



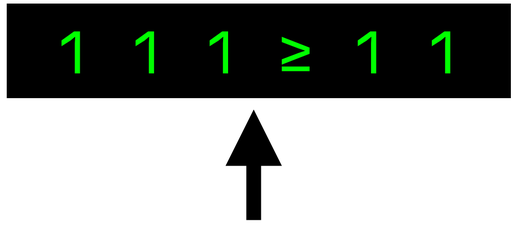
Stack



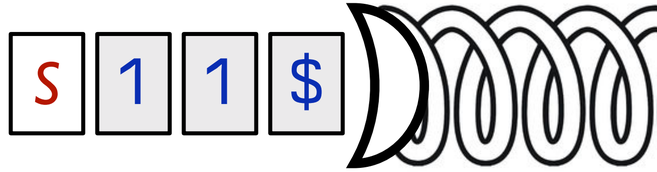
Example: Equivalent PDA



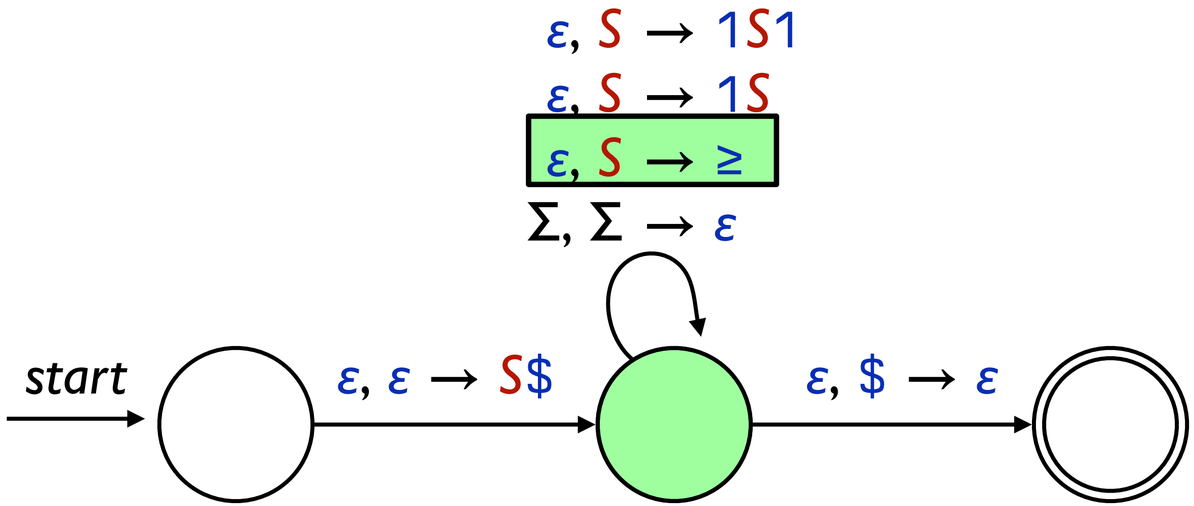
Input



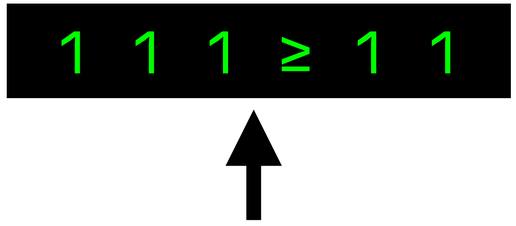
Stack



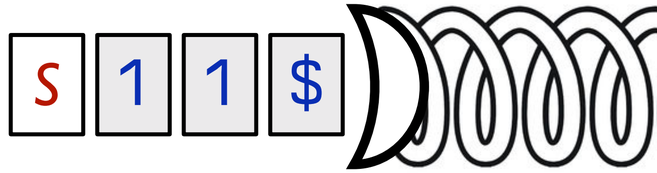
Example: Equivalent PDA



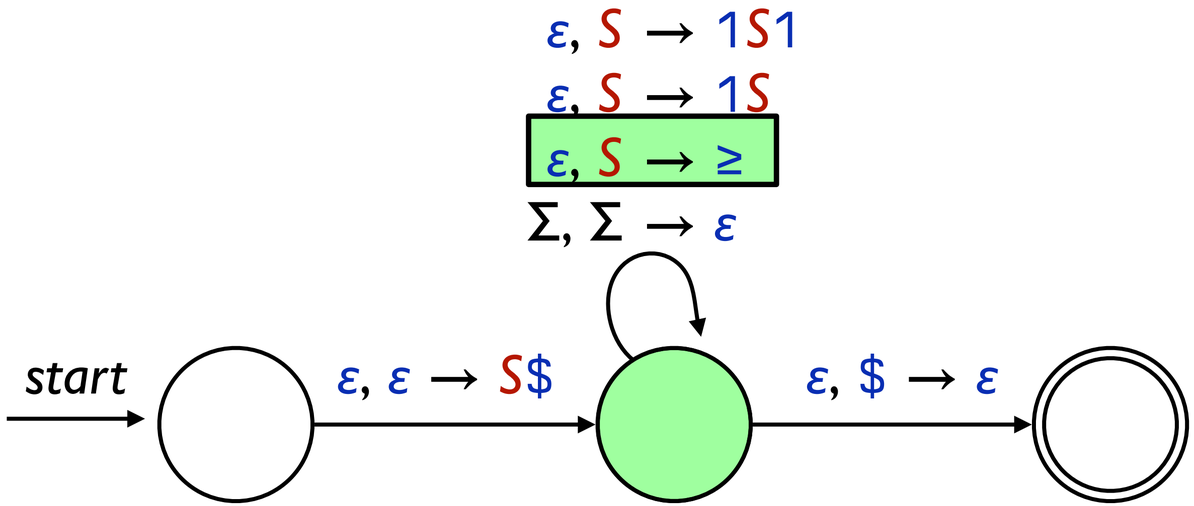
Input



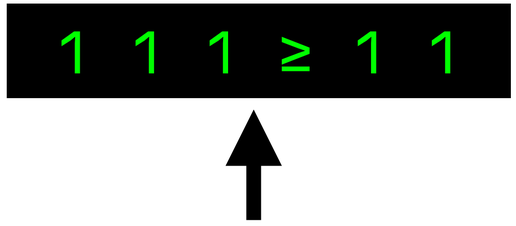
Stack



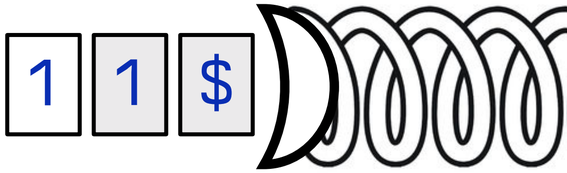
Example: Equivalent PDA



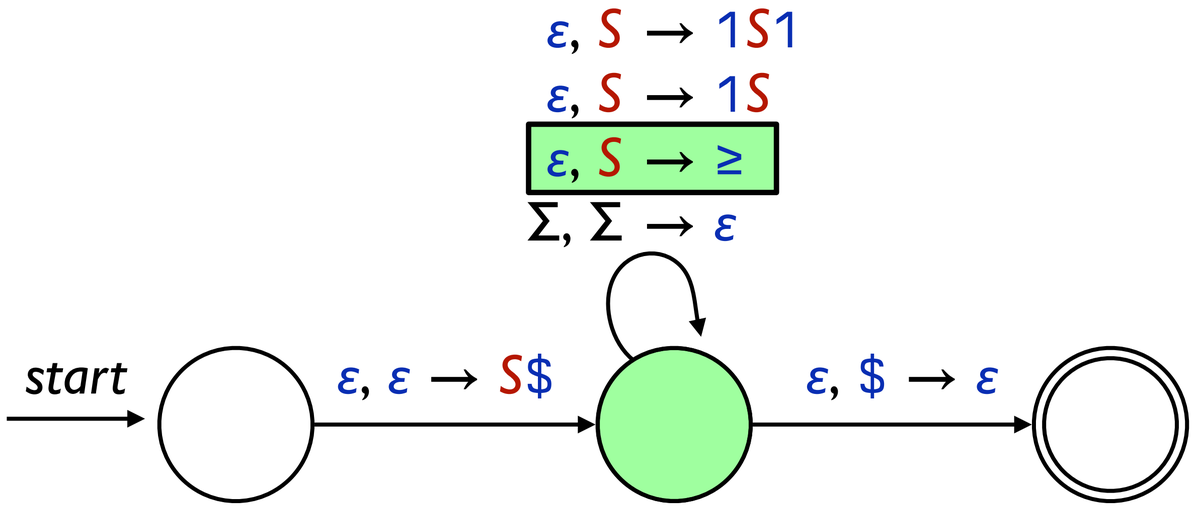
Input



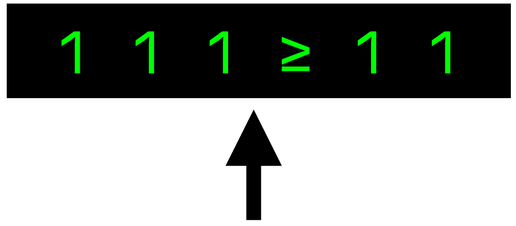
Stack



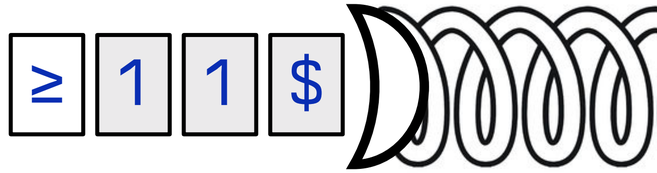
Example: Equivalent PDA



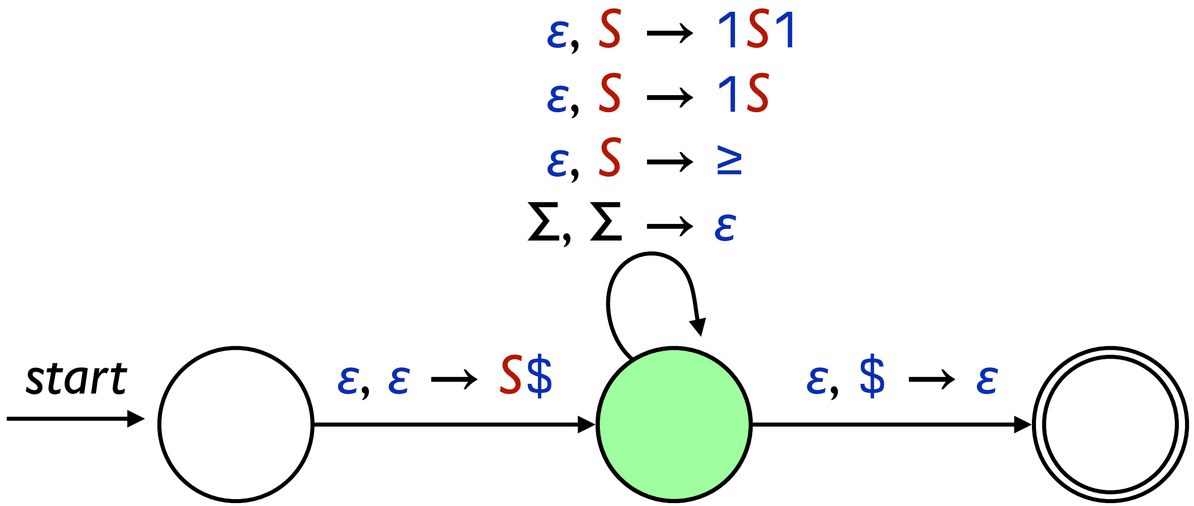
Input



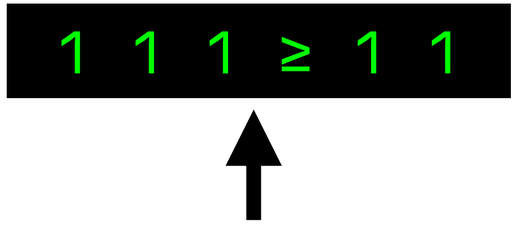
Stack



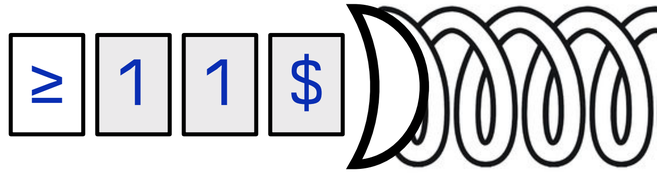
Example: Equivalent PDA



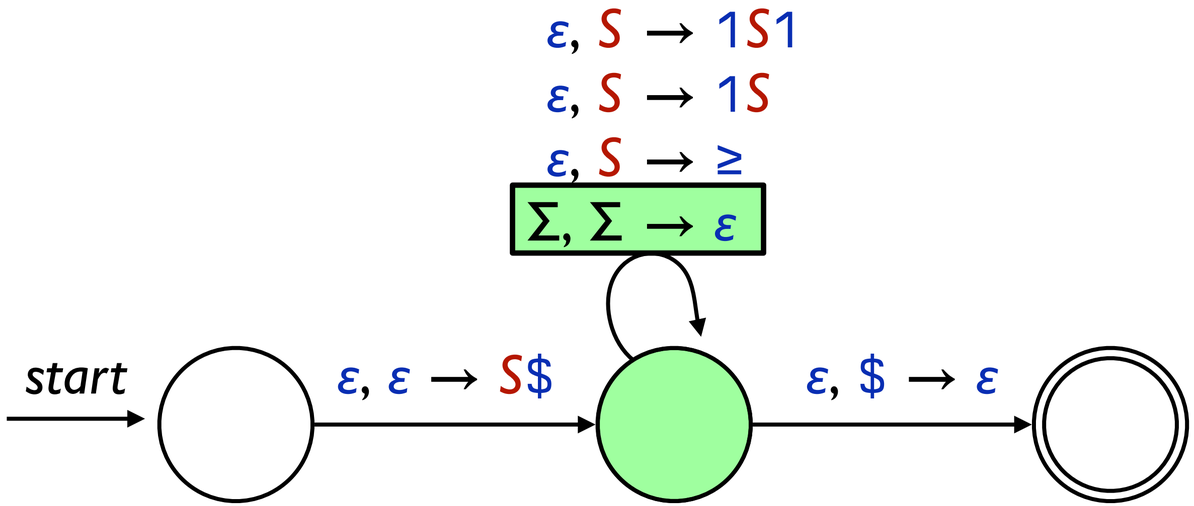
Input



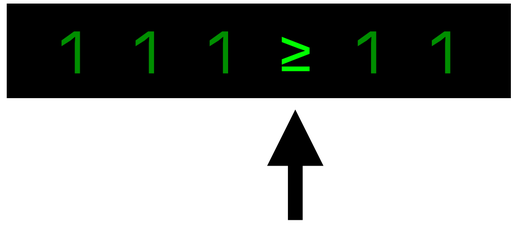
Stack



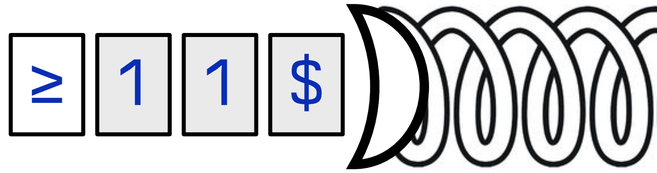
Example: Equivalent PDA



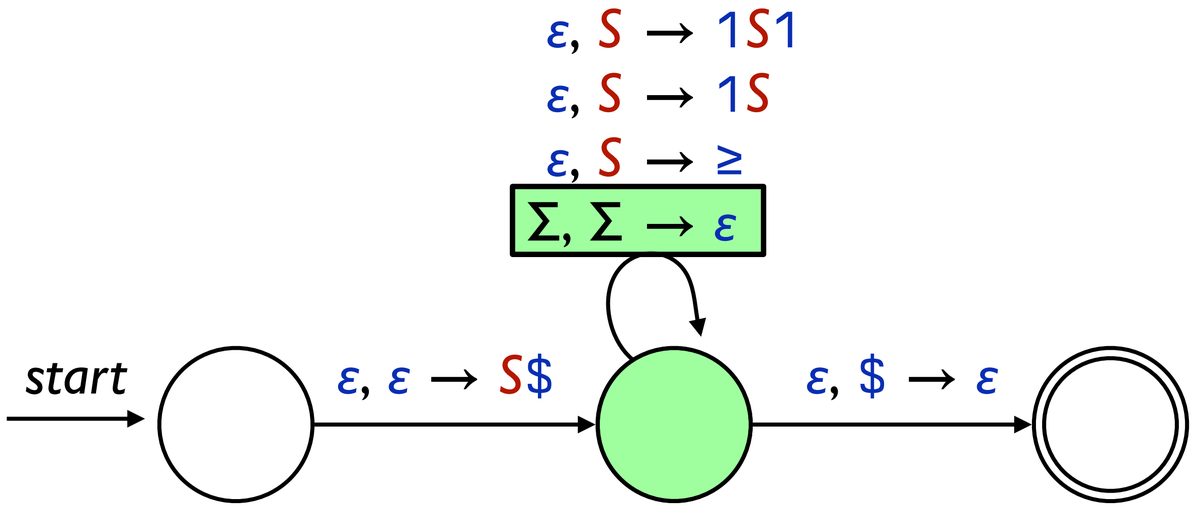
Input



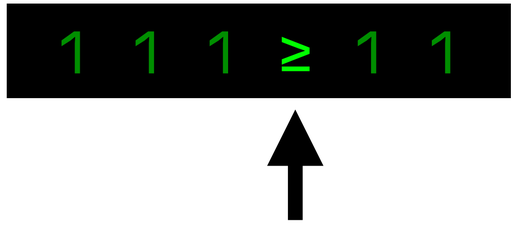
Stack



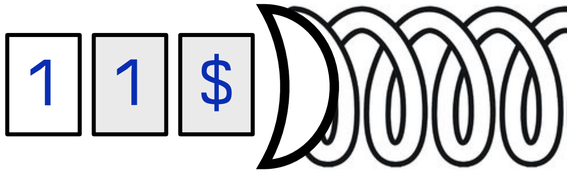
Example: Equivalent PDA



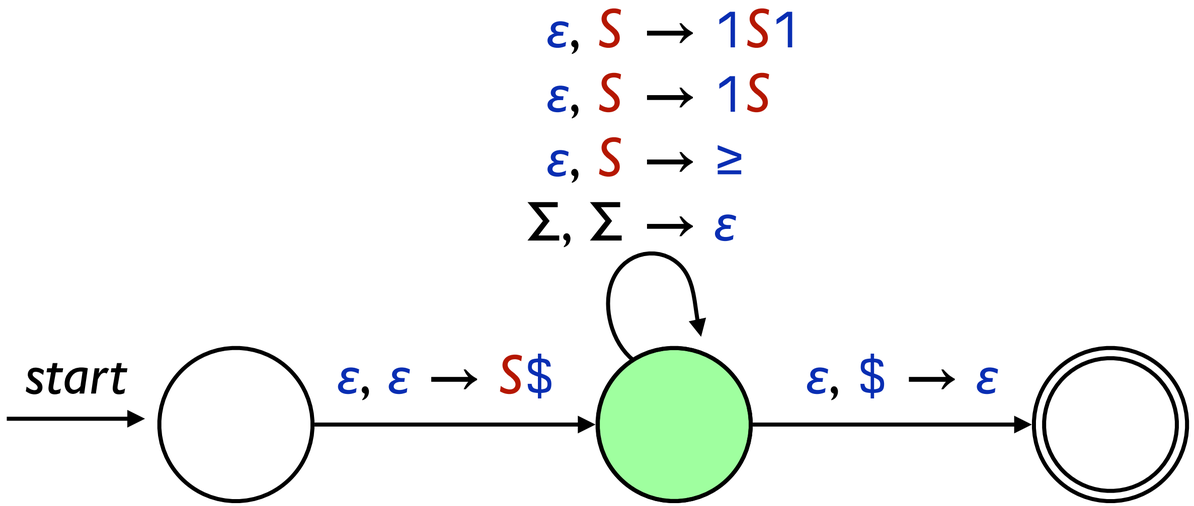
Input



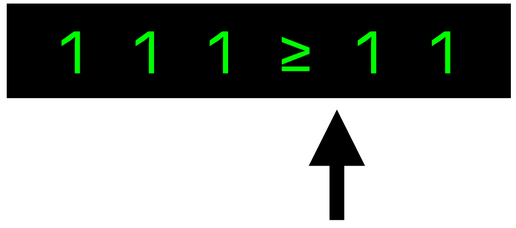
Stack



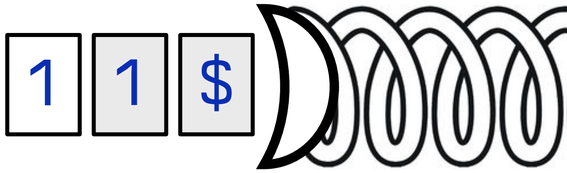
Example: Equivalent PDA



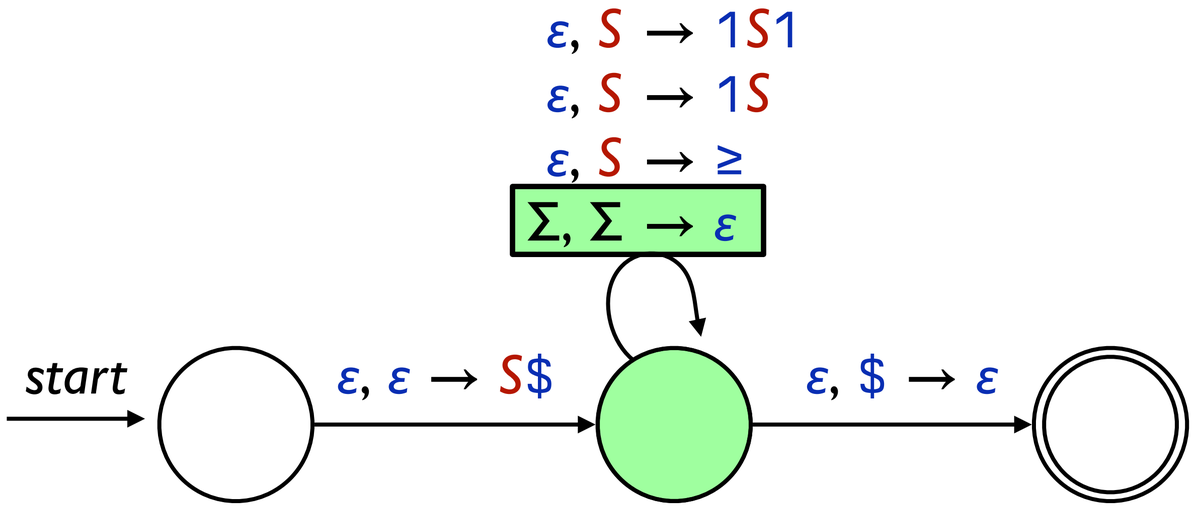
Input



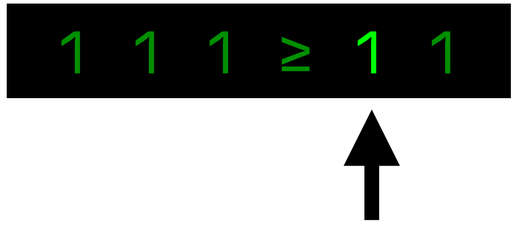
Stack



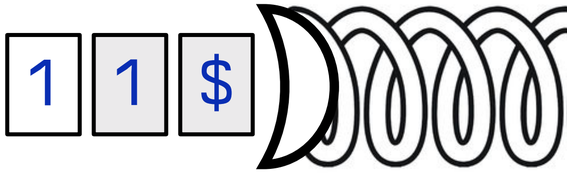
Example: Equivalent PDA



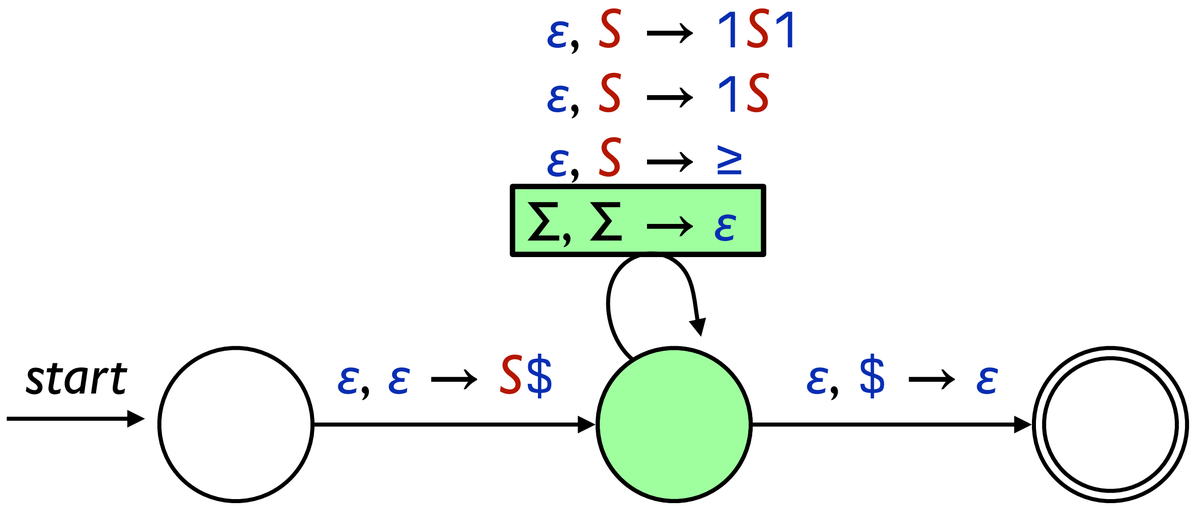
Input



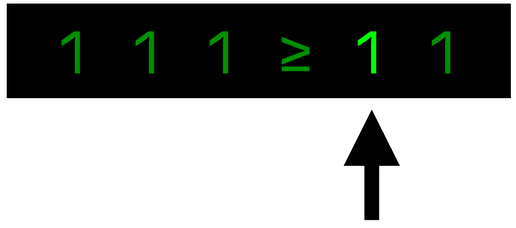
Stack



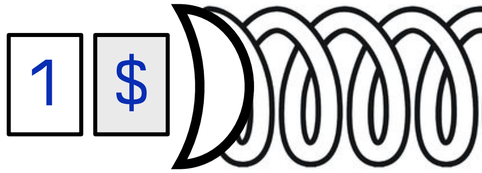
Example: Equivalent PDA



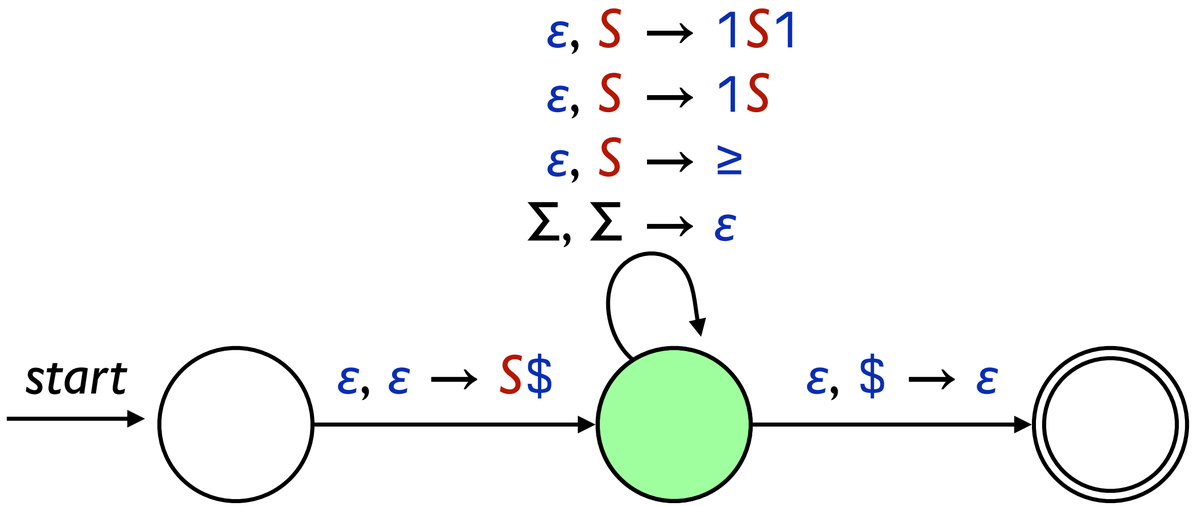
Input



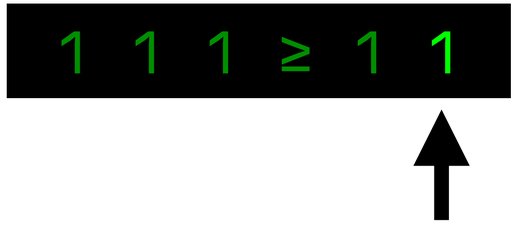
Stack



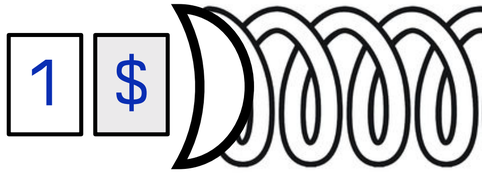
Example: Equivalent PDA



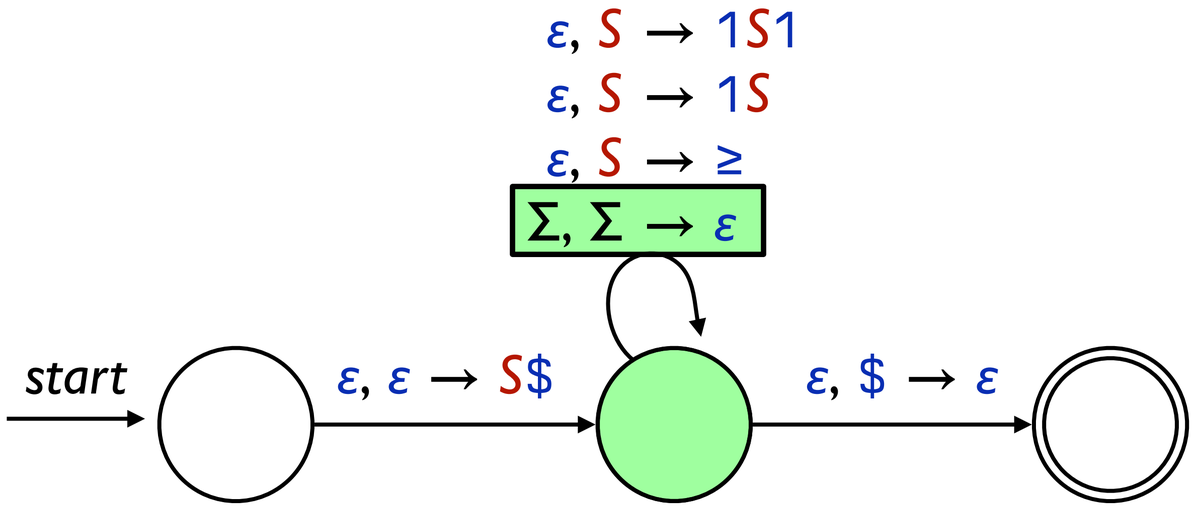
Input



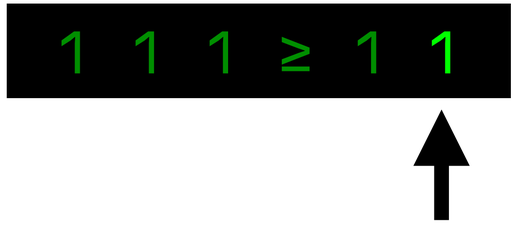
Stack



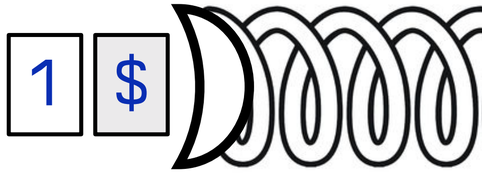
Example: Equivalent PDA



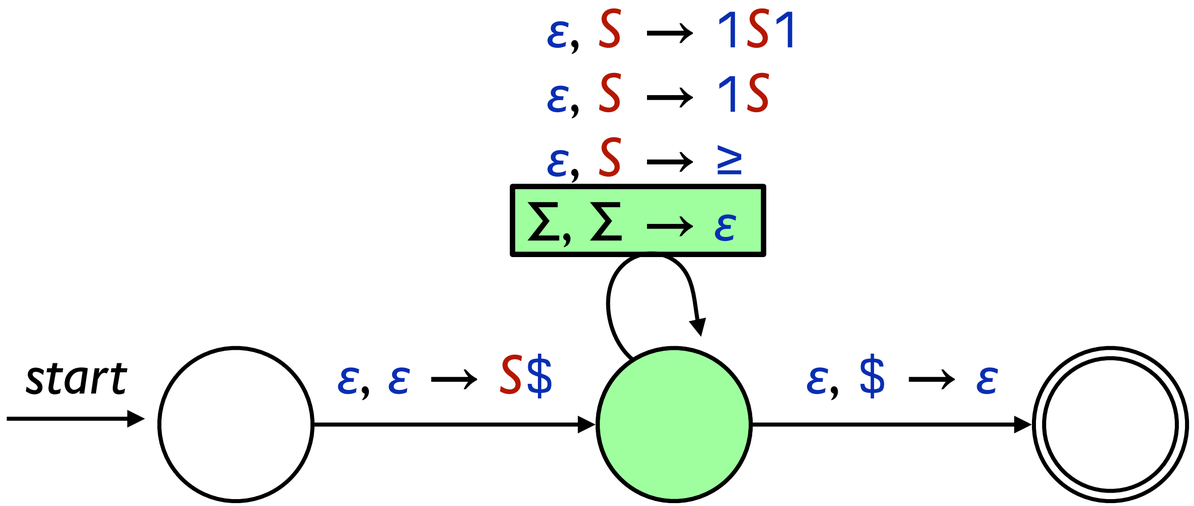
Input



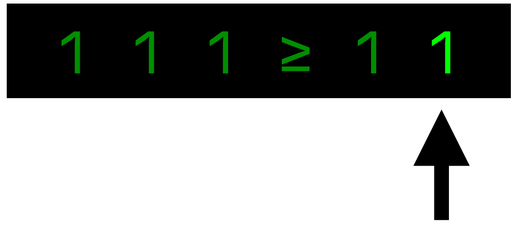
Stack



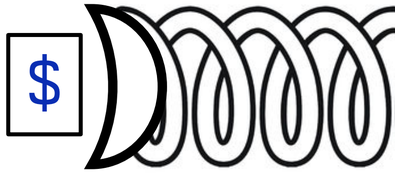
Example: Equivalent PDA



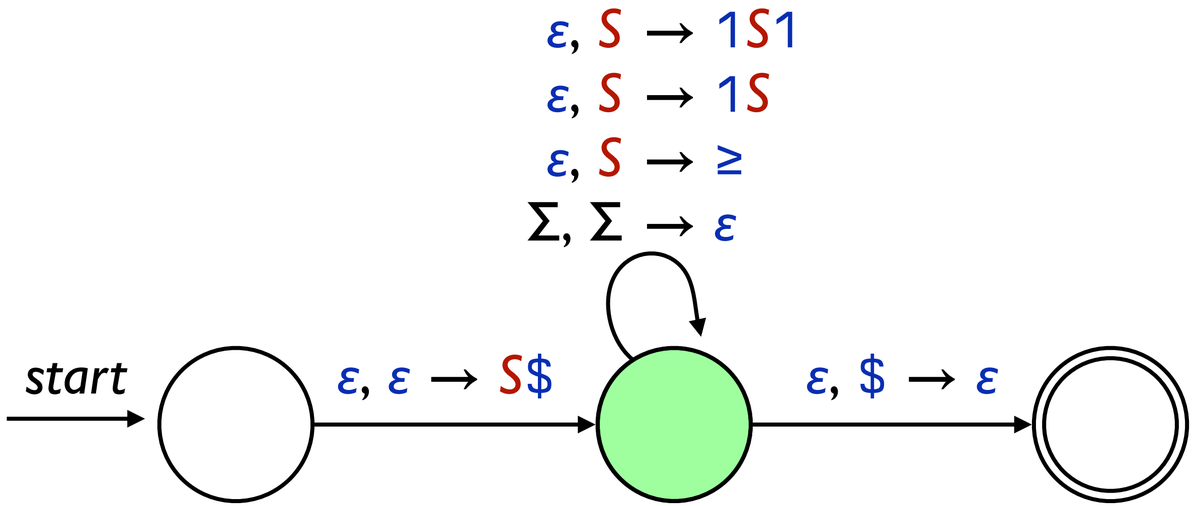
Input



Stack

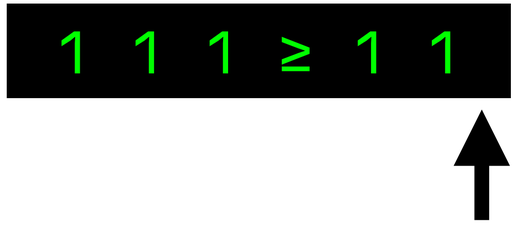


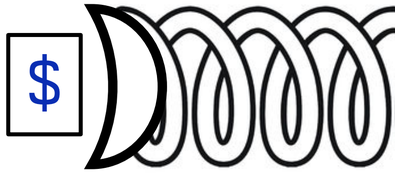
Example: Equivalent PDA



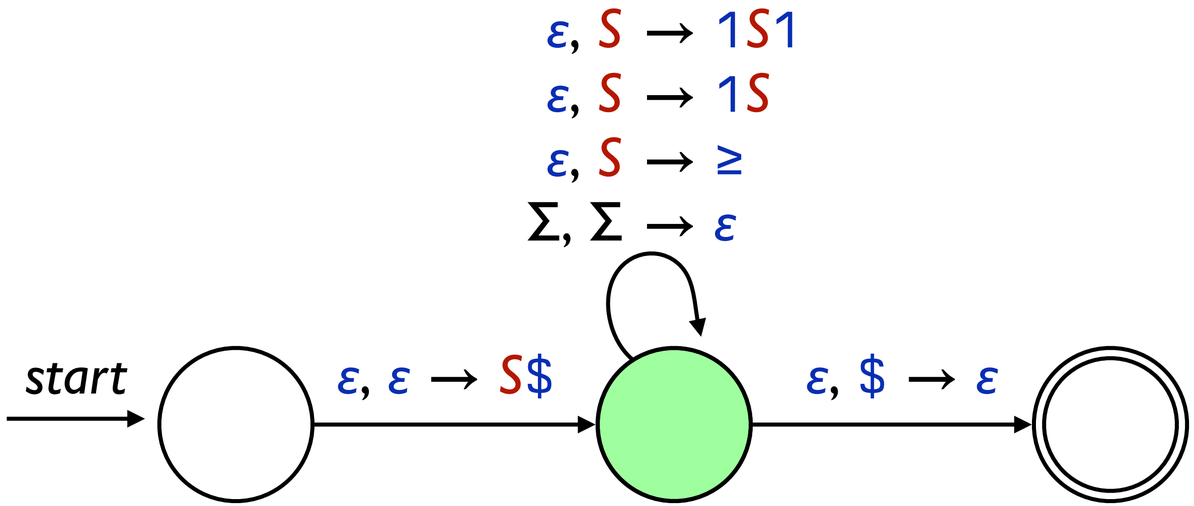
Input

Stack

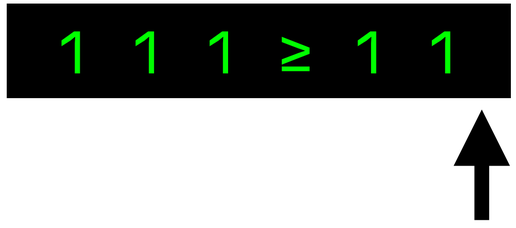




Example: Equivalent PDA

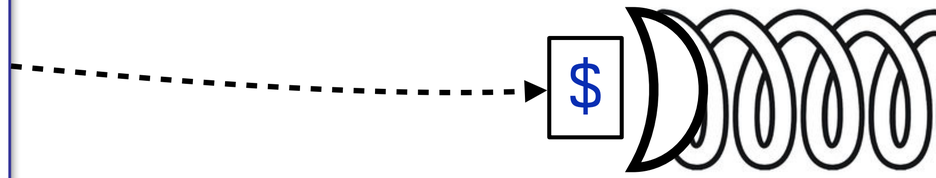


Input

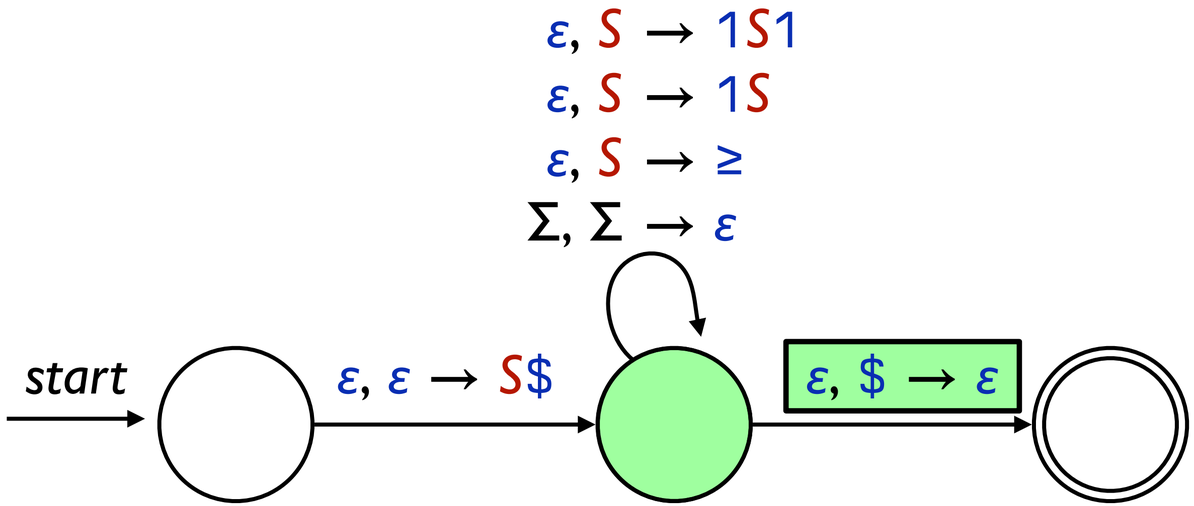


At this point, we've completely matched the string, so it's time to transition to the accepting state.

Stack

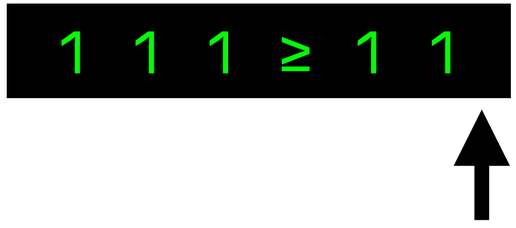


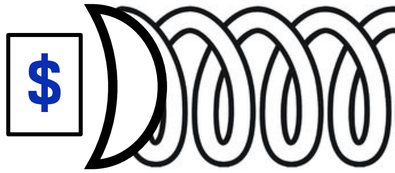
Example: Equivalent PDA



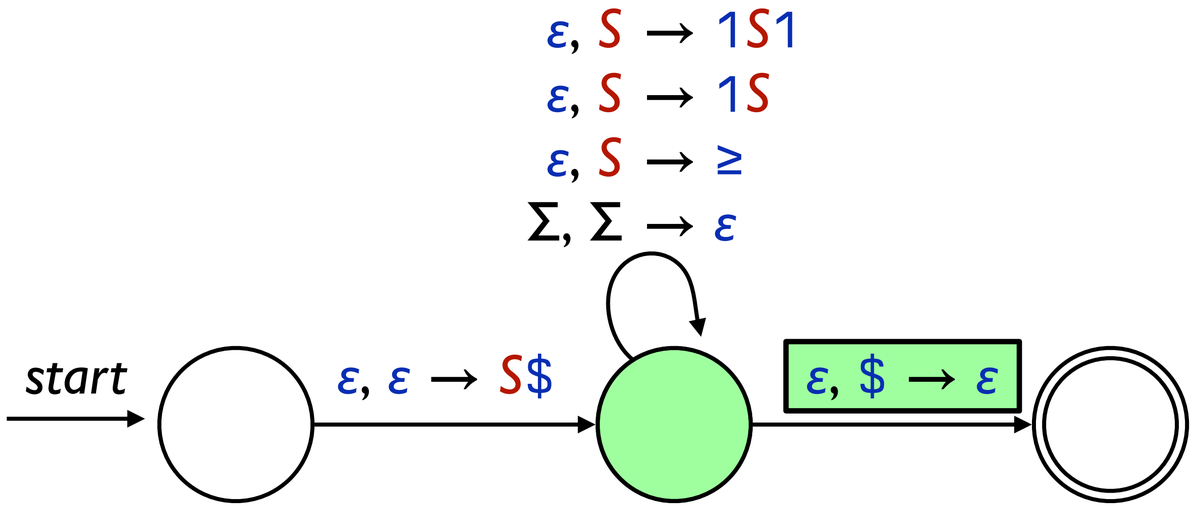
Input

Stack



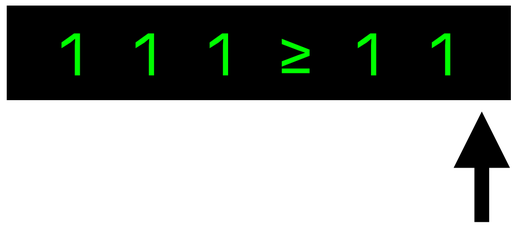


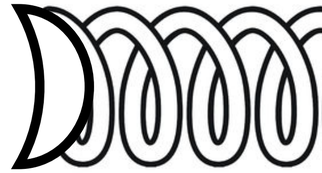
Example: Equivalent PDA



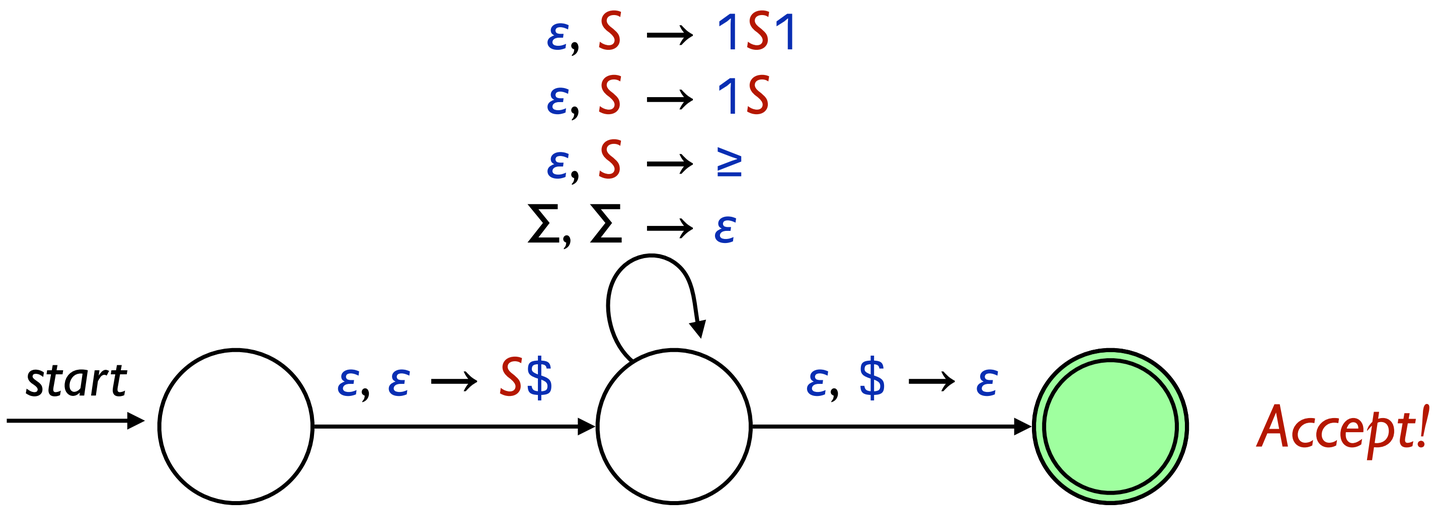
Input

Stack



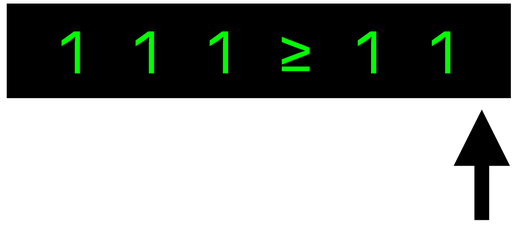


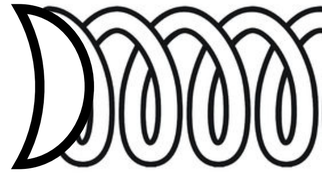
Example: Equivalent PDA



Input

Stack





Creating a PDA for a CFG

Make three states: *start*, *loop*, and *accepting*.

There is a transition ε, $\varepsilon \rightarrow S\$$ from *start* to *loop*.

Corresponds to starting off with the start symbol S.

There is a transition ε, $A \rightarrow \omega$ from *loop* to itself for each production $A \rightarrow \omega$.

Corresponds to predicting which production to use.

There is a transition Σ, $\Sigma \rightarrow \varepsilon$ from *loop* to itself.

Corresponds to matching a character of the input.

There is a transition ε, $\$ \rightarrow \varepsilon$ from *loop* to *accepting*.

Corresponds to completely matching the input.

Moves of the PDA

Push the empty stack symbol $ and the start variable onto the stack.

Repeat forever:

- If a terminal a is on top of the stack, pop it and compare it to the next symbol from the input; repeat until the top of the stack is not a terminal.
- If a variable *B* is on top of the stack, non-deterministically select a production rule with *B* as the head and substitute for *B* on the stack.
- If $ is on top of the stack and all input has been read, accept.

Example: CFG

$R \rightarrow XRX \mid S$

$S \rightarrow aTb \mid bTa$

$T \rightarrow XTX \mid X \mid \varepsilon$

$X \rightarrow a \mid b$

Define the PDA

$$R \rightarrow XRX \mid S$$
$$S \rightarrow aTb \mid bTa$$
$$T \rightarrow XTX \mid X \mid \varepsilon$$
$$X \rightarrow a \mid b$$

$N = (\{q_{start}, q_{loop}, q_{accept}\}, \{a, b\}, \{a, b, R, S, T, X, \$\}, \delta, q_{start}, \{q_{accept}\})$

$\delta(q_{start}, \varepsilon, \varepsilon) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \quad \varepsilon)$

Parse

PDA

$$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$$

$$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$$

$$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$$

$$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$$

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

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

$$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$$

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

Transitions

$$(q_{start}, aabba, \varepsilon)$$
$$\vdash (q_{loop}, aabba, R\$)$$

Parse

R

PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, \mathtt{a}T\mathtt{b}), (q_{loop}, \mathtt{b}T\mathtt{a})\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

$\delta(q_{loop}, \varepsilon, X) = \{(q_{loop}, \mathtt{a}), (q_{loop}, \mathtt{b})\}$

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

$\delta(q_{loop}, \mathtt{b}, \mathtt{b}) = \{(q_{loop}, \varepsilon)\}$

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

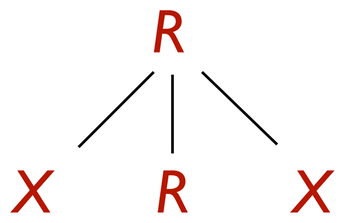
Transitions

$(q_{start}, \mathtt{aabba}, \varepsilon)$

$\vdash (q_{loop}, \mathtt{aabba}, R\$)$

$\vdash (q_{loop}, \mathtt{aabba}, XRX\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

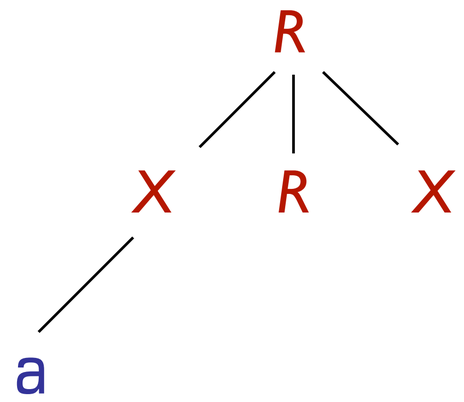
$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

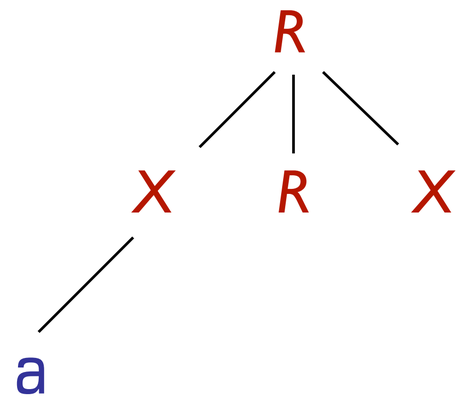
$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

$\vdash (q_{loop}, abba, RX\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

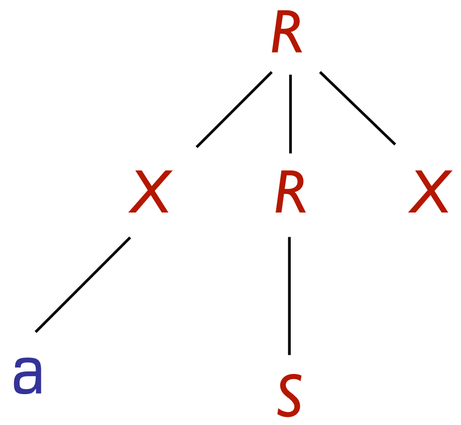
$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

$\vdash (q_{loop}, abba, RX\$)$

$\vdash (q_{loop}, abba, SX\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

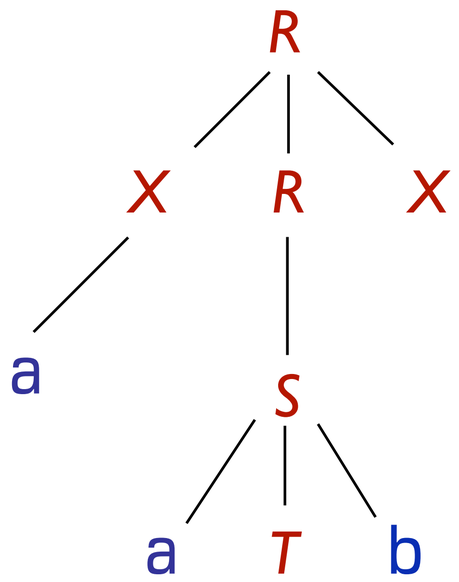
$\vdash (q_{loop}, aabba, aRX\$)$

$\vdash (q_{loop}, abba, RX\$)$

$\vdash (q_{loop}, abba, SX\$)$

$\vdash (q_{loop}, abba, aTbX\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

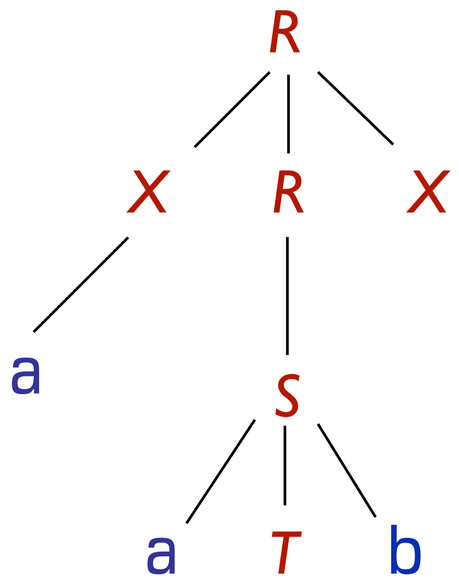
$\vdash (q_{loop}, abba, RX\$)$

$\vdash (q_{loop}, abba, SX\$)$

$\vdash (q_{loop}, abba, aTbX\$)$

$\vdash (q_{loop}, bba, TbX\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

$\vdash (q_{loop}, abba, RX\$)$

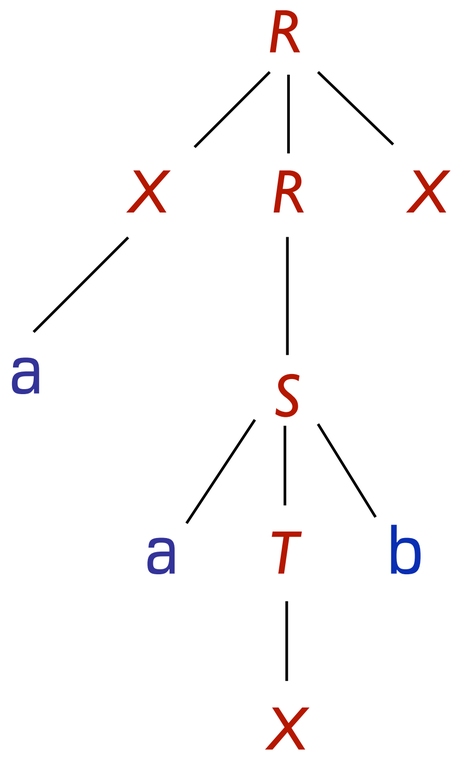
$\vdash (q_{loop}, abba, SX\$)$

$\vdash (q_{loop}, abba, aTbX\$)$

$\vdash (q_{loop}, bba, TbX\$)$

$\vdash (q_{loop}, bba, XbX\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

$\vdash (q_{loop}, abba, RX\$)$

$\vdash (q_{loop}, abba, SX\$)$

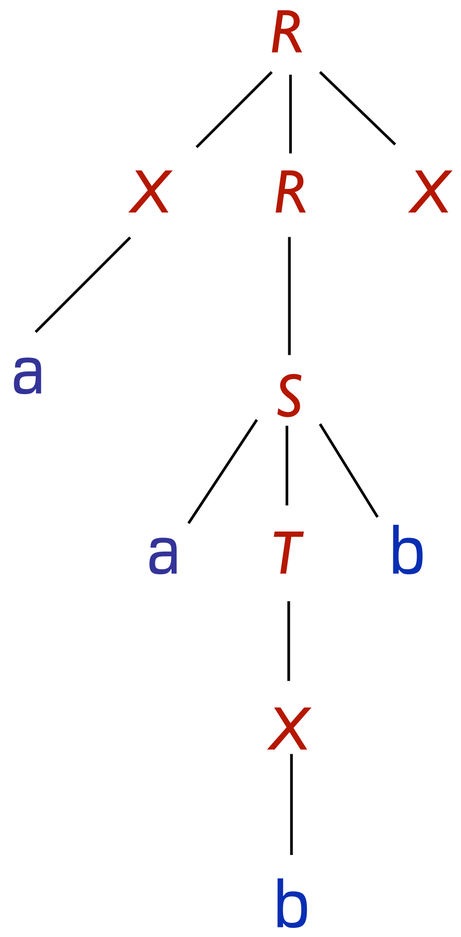
$\vdash (q_{loop}, abba, aTbX\$)$

$\vdash (q_{loop}, bba, TbX\$)$

$\vdash (q_{loop}, bba, XbX\$)$

$\vdash (q_{loop}, bba, bbX\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

$\vdash (q_{loop}, abba, RX\$)$

$\vdash (q_{loop}, abba, SX\$)$

$\vdash (q_{loop}, abba, aTbX\$)$

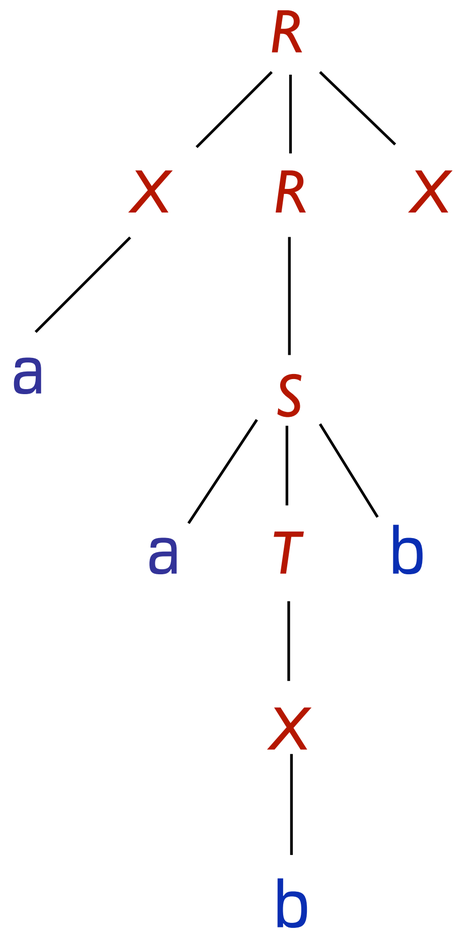
$\vdash (q_{loop}, bba, TbX\$)$

$\vdash (q_{loop}, bba, XbX\$)$

$\vdash (q_{loop}, bba, bbX\$)$

$\vdash (q_{loop}, ba, bX\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

$\vdash (q_{loop}, abba, RX\$)$

$\vdash (q_{loop}, abba, SX\$)$

$\vdash (q_{loop}, abba, aTbX\$)$

$\vdash (q_{loop}, bba, TbX\$)$

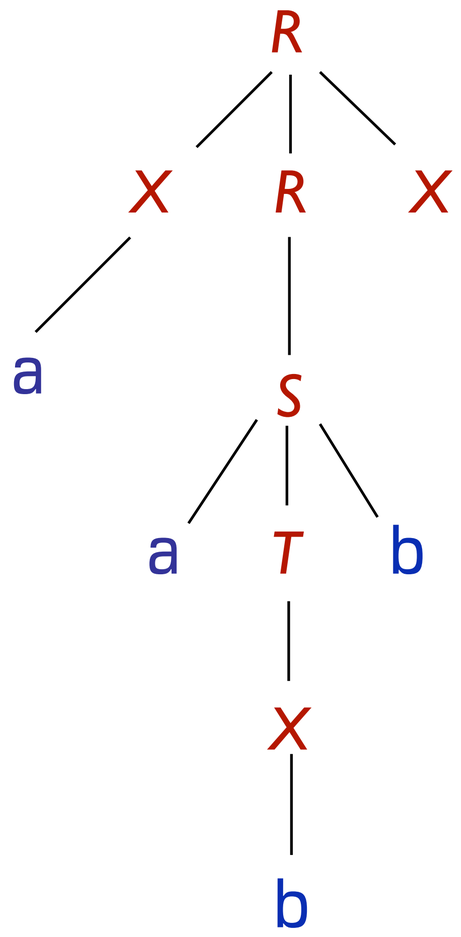
$\vdash (q_{loop}, bba, XbX\$)$

$\vdash (q_{loop}, bba, bbX\$)$

$\vdash (q_{loop}, ba, bX\$)$

$\vdash (q_{loop}, a, X\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

$\vdash (q_{loop}, abba, RX\$)$

$\vdash (q_{loop}, abba, SX\$)$

$\vdash (q_{loop}, abba, aTbX\$)$

$\vdash (q_{loop}, bba, TbX\$)$

$\vdash (q_{loop}, bba, XbX\$)$

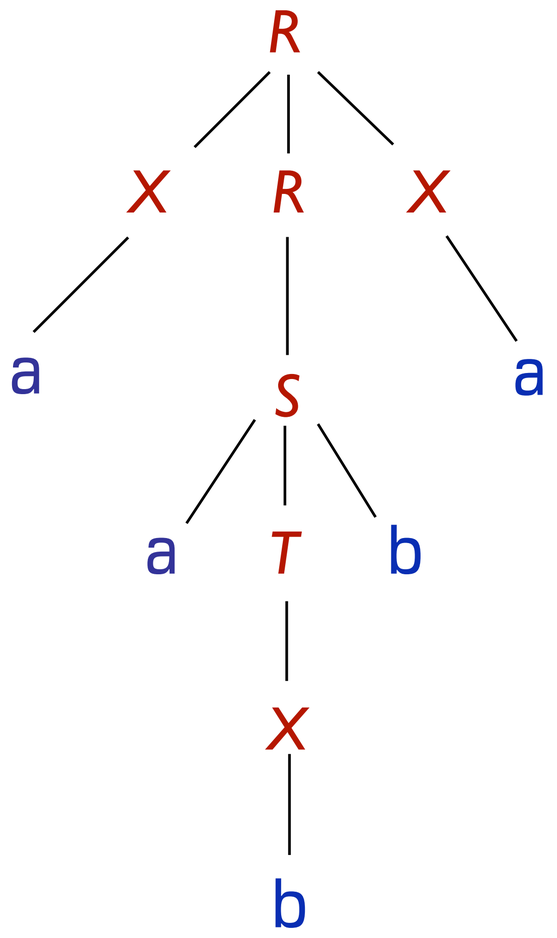
$\vdash (q_{loop}, bba, bbX\$)$

$\vdash (q_{loop}, ba, bX\$)$

$\vdash (q_{loop}, a, X\$)$

$\vdash (q_{loop}, a, a\$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

$\vdash (q_{loop}, abba, RX\$)$

$\vdash (q_{loop}, abba, SX\$)$

$\vdash (q_{loop}, abba, aTbX\$)$

$\vdash (q_{loop}, bba, TbX\$)$

$\vdash (q_{loop}, bba, XbX\$)$

$\vdash (q_{loop}, bba, bbX\$)$

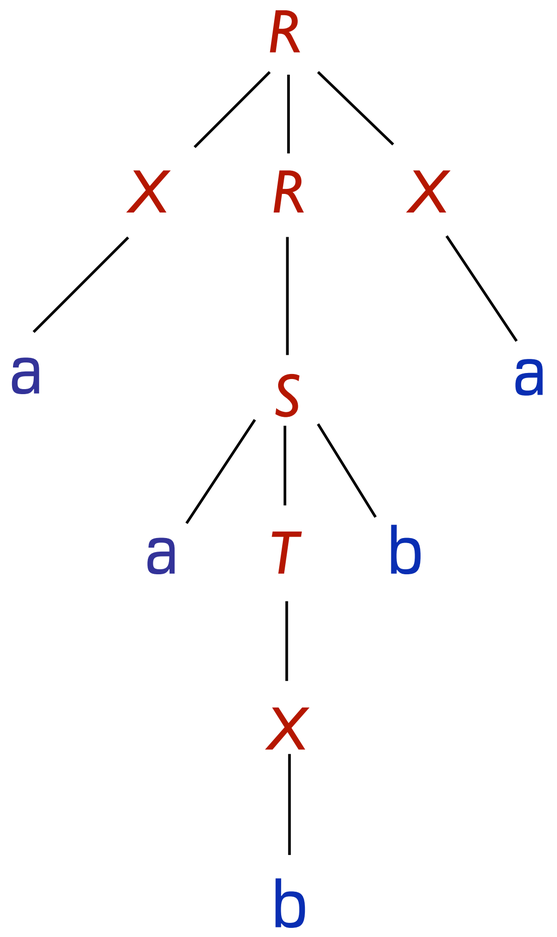
$\vdash (q_{loop}, ba, bX\$)$

$\vdash (q_{loop}, a, X\$)$

$\vdash (q_{loop}, a, a\$)$

$\vdash (q_{loop}, \varepsilon, \$)$

Parse



PDA

$\delta(q_{start}, \varepsilon, R) = \{(q_{loop}, R\$)\}$

$\delta(q_{loop}, \varepsilon, R) = \{(q_{loop}, XRX), (q_{loop}, S)\}$

$\delta(q_{loop}, \varepsilon, S) = \{(q_{loop}, aTb), (q_{loop}, bTa)\}$

$\delta(q_{loop}, \varepsilon, T) = \{(q_{loop}, XTX), (q_{loop}, X), (q_{loop}, \varepsilon)\}$

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

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

$\delta(q_{loop}, b, b) = \{(q_{loop}, \varepsilon)\}$

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

Transitions

$(q_{start}, aabba, \varepsilon)$

$\vdash (q_{loop}, aabba, R\$)$

$\vdash (q_{loop}, aabba, XRX\$)$

$\vdash (q_{loop}, aabba, aRX\$)$

$\vdash (q_{loop}, abba, RX\$)$

$\vdash (q_{loop}, abba, SX\$)$

$\vdash (q_{loop}, abba, aTbX\$)$

$\vdash (q_{loop}, bba, TbX\$)$

$\vdash (q_{loop}, bba, XbX\$)$

$\vdash (q_{loop}, bba, bbX\$)$

$\vdash (q_{loop}, ba, bX\$)$

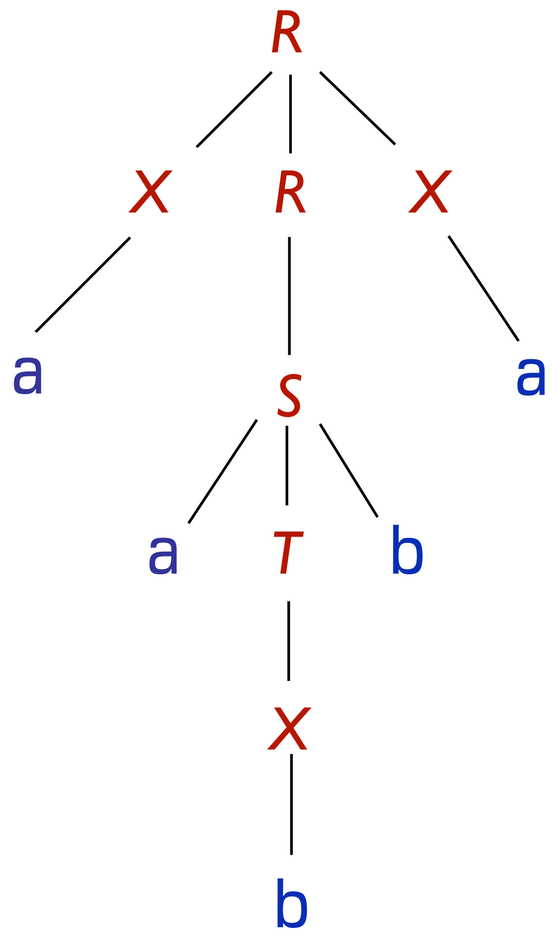
$\vdash (q_{loop}, a, X\$)$

$\vdash (q_{loop}, a, a\$)$

$\vdash (q_{loop}, \varepsilon, \$)$

$\vdash (q_{accept}, \varepsilon, \varepsilon)$

Parse



PDA → CFG

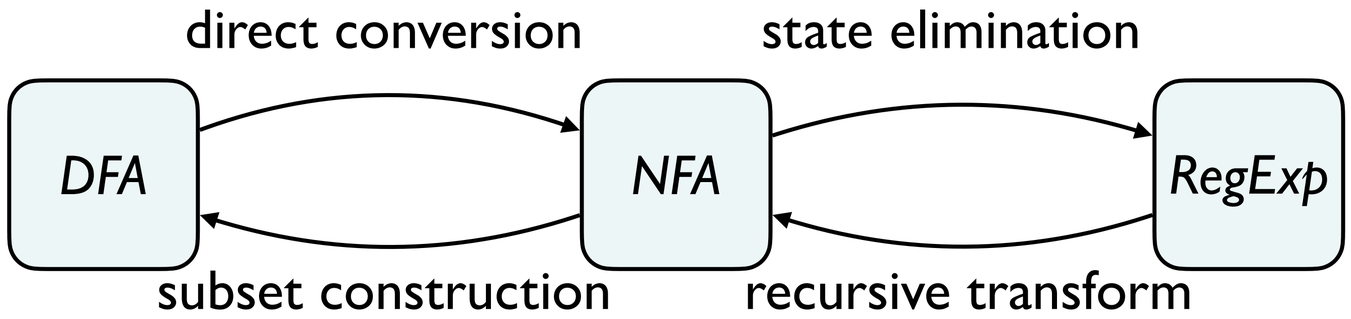
The other direction of the proof, converting a PDA to a CFG, is interesting – but much harder!

Intuitively, we create a CFG representing *paths* between states in the PDA.

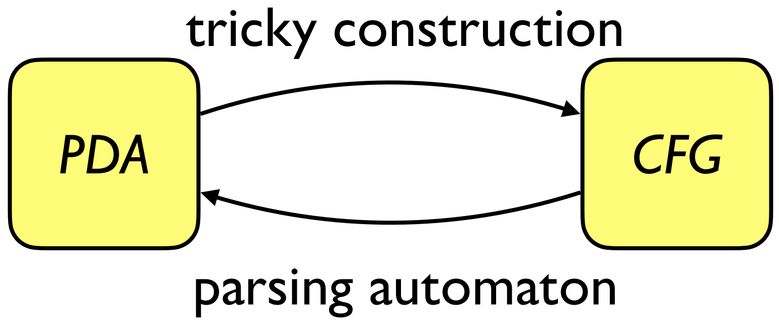
Details are in Sipser.

Our transformations

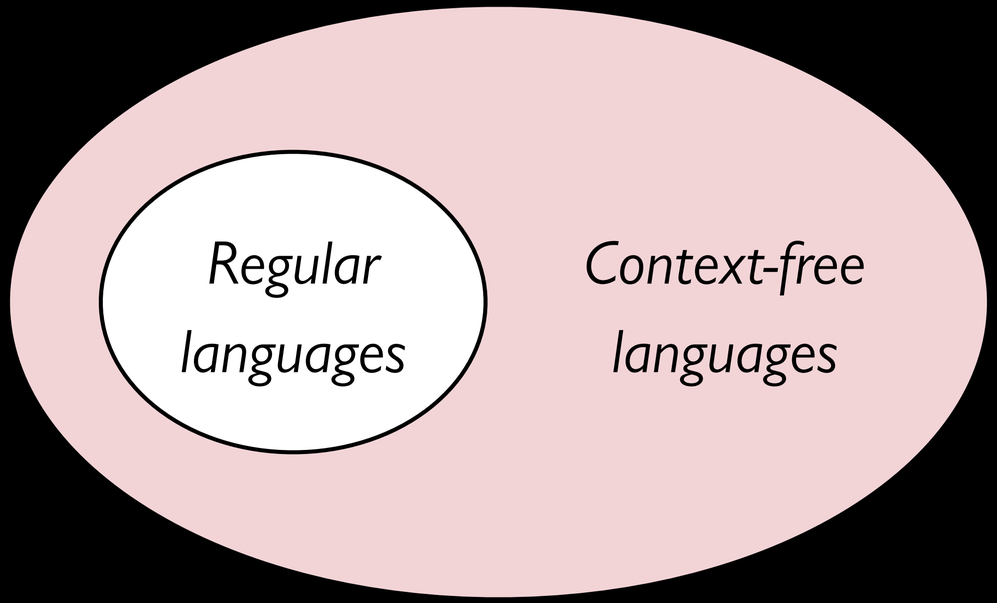
Regular languages



Context-free languages



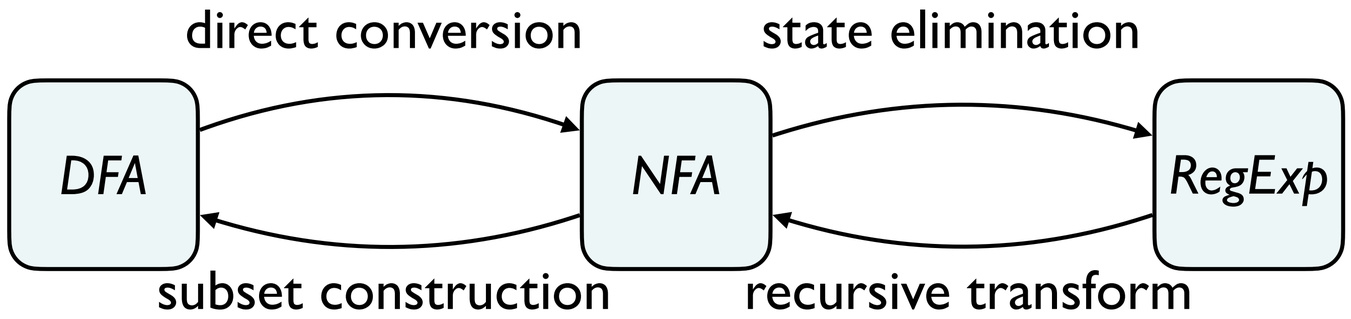
Regular and context-free languages



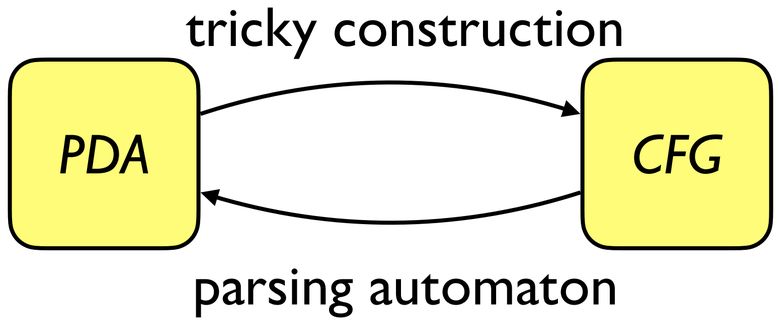
All languages

Our transformations

Regular languages



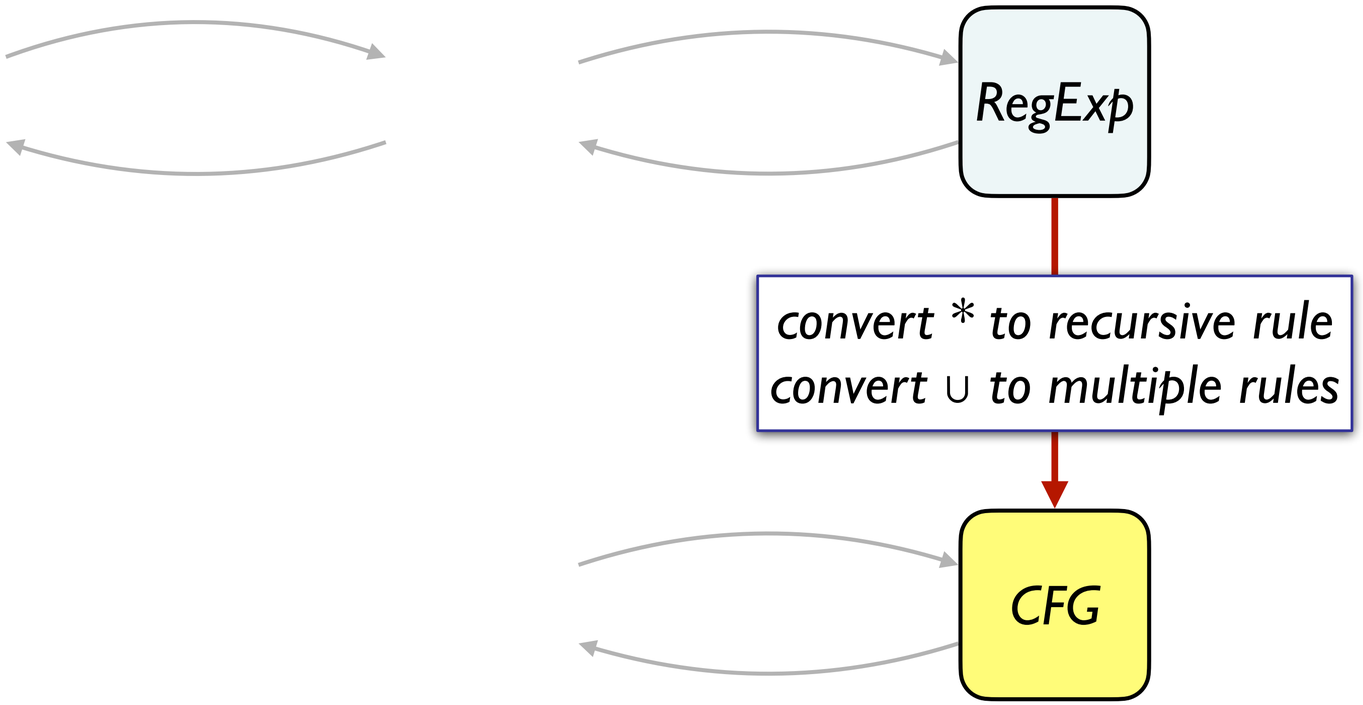
Context-free languages



Our transformations

Regular languages

Context-free languages



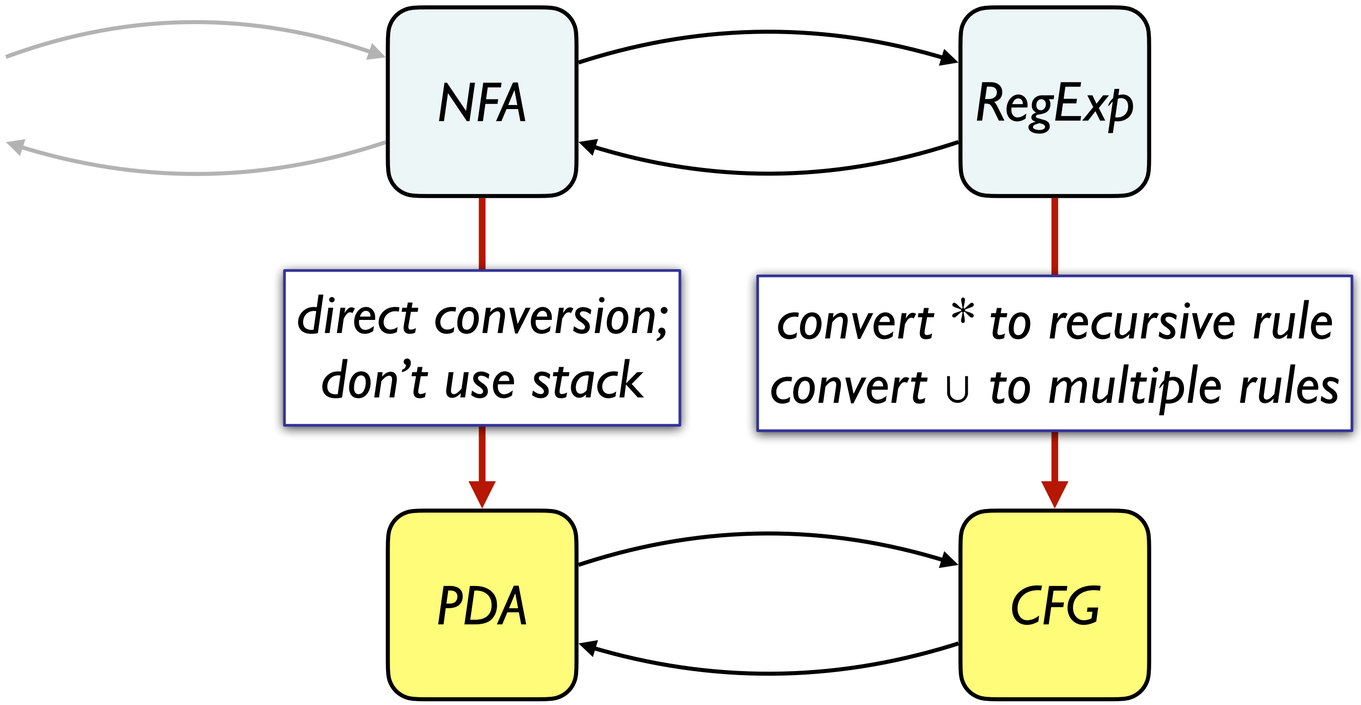
THEOREM Any regular language is context-free.

PROOF SKETCH Let L be any regular language and consider an NFA N for L. Then we can convert N into a PDA for L by converting any transition on a symbol a into a transition $a, \varepsilon \rightarrow \varepsilon$ that ignores the stack. This new PDA accepts L, so L is context-free.

Our transformations

Regular languages

Context-free languages



PDAs in the real world: Getting deterministic

With finite automata, we considered both deterministic (DFA) and nondeterministic (NFA) versions.

So far, what we've been looking at are nondeterministic PDAs (NPDAs).

A *deterministic PDA* (*DPDA*) is a PDA with this extra property:

> For each state in the PDA, and for any combination of a current input symbol and a current stack symbol, there is *at most* one transition defined.

In other words, there's *at most* one legal sequence of transitions that can be followed for any input.

Being deterministic doesn't preclude ε-transitions – as long as there's never a conflict between following the ε-transition or another transition.

> There can be at most one ε-transition that could be followed at a time!

Being deterministic also doesn't preclude a DPDA "dying" from having no transitions defined.

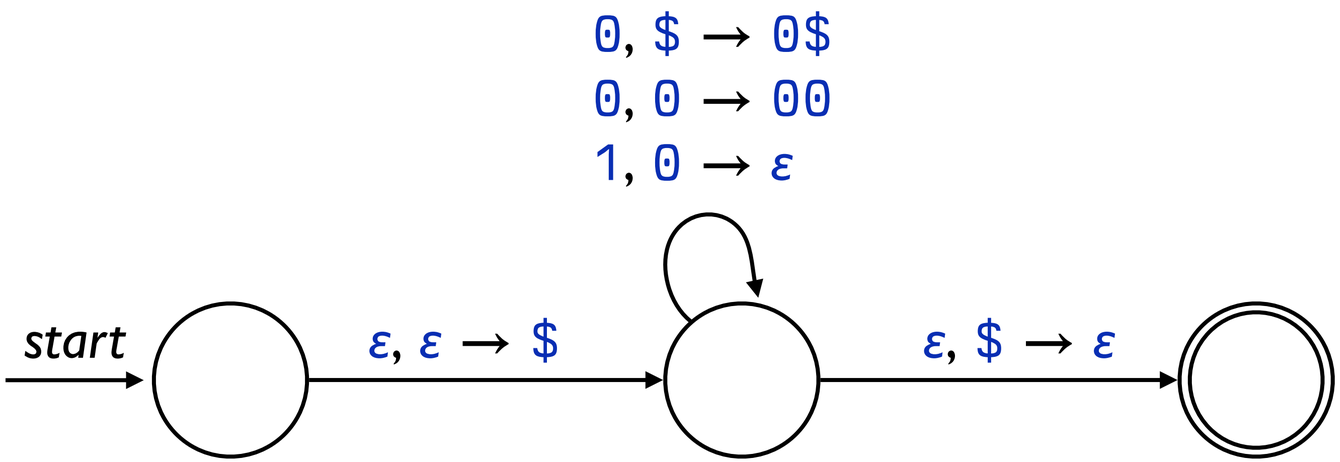
Being deterministic doesn't preclude ε-transitions – as long as there's never a conflict between following the ε-transition or another transition.

There can be at most one ε-transition that could be followed at a time!

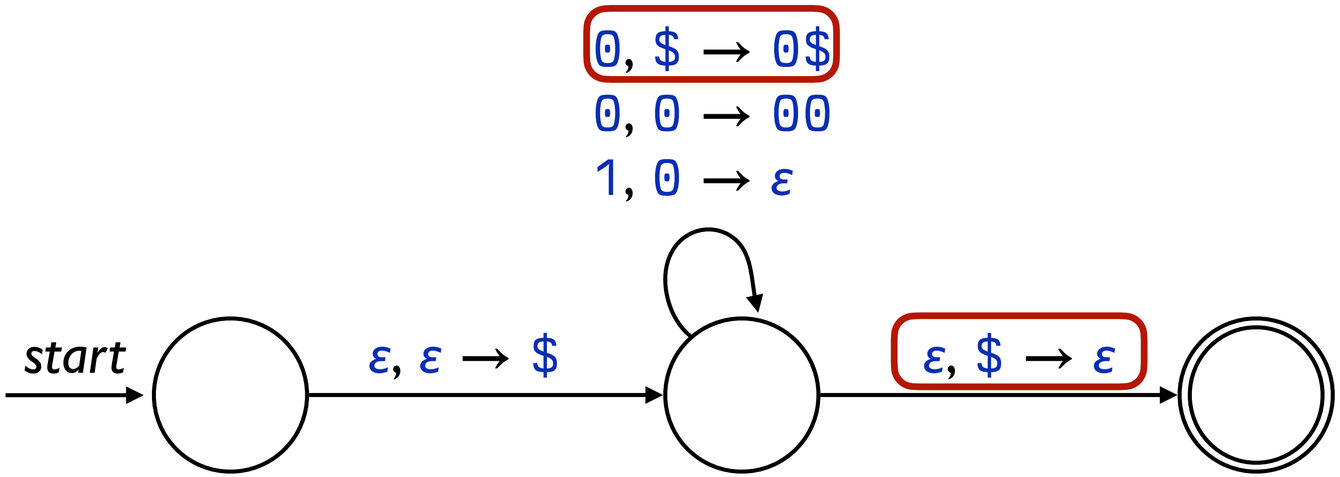
Being deterministic also doesn't preclude a DPDA "dying" from having no transitions defined.

Note: Sipser's formulation of DPDAs requires every transition be defined. For CMPU 240, we'll allow them to die when the transition's undefined (but we also won't use DPDAs very much)!

Is this a DPDA?

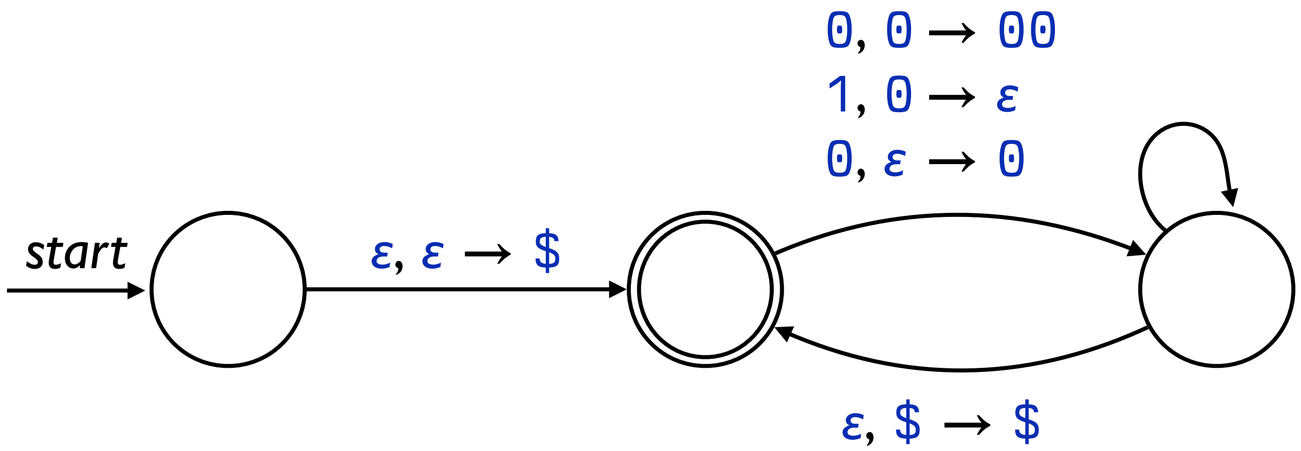


Is this a DPDA? ✗



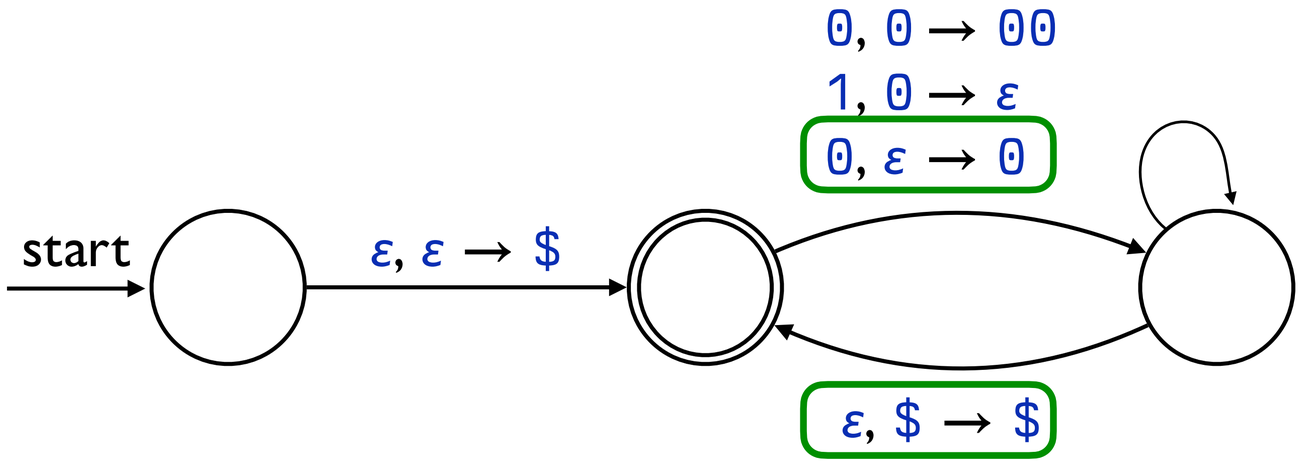
No – there's a choice of what to do when there's a 0 input and a $ on the stack

Is this a DPDA?



Is this a DPDA? ✓

This ε-transition is allowed because no other transitions in this state use the input symbol `0`



This ε-transition is allowed because no other transitions in this state use the stack symbol $.

When dealing with finite automata, there is no difference in the power of NFAs and DFAs.

However, when dealing with PDAs, there are CFLs that can be recognized by NPDAs that *cannot* be recognized by DPDAs.

That is, NPDAs are strictly *more powerful* than DPDAs.

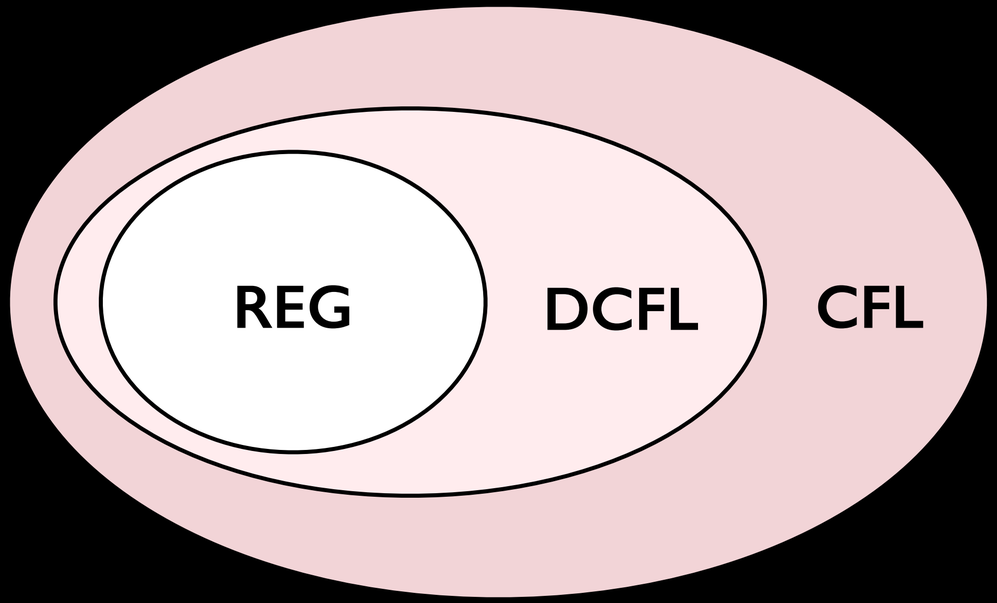
An example of a context-free language that can be recognized by an NPDA but not by a DPDA is the language of all palindromes (without a dividing marker).

Intuition: Without nondeterminism, how would you know when you've read half the string?

A context-free language *L* is called a ***deterministic context-free language*** (DCFL) if there is some DPDA that recognizes *L*.

Not all CFLs are DCFLs, though many important ones are.

It's extremely difficult to actually *prove* that a given CFL is not a DCFL!



All languages

Why should we care about deterministic PDAs if they're less powerful?

Because they're deterministic, they can be simulated efficiently:

- Keep track of the top of the stack.
- Store an *action/goto table* that says what operations to perform on the stack and what state to enter on each input–stack pair.
- Loop over the input, processing input–stack pairs until the automaton rejects or ends in an accepting state with all input consumed.

If we can find a DPDA for a CFL, then we can recognize strings in that language efficiently.

Parsers are DPDAs!

Thus, the question of what languages a DPDA can accept is really the question of what programming language syntax can be parsed conveniently.

Unambiguous, deterministic CFGs are complicated and too restricted, so *real parsers cheat by looking ahead one token.*

This places certain restrictions on the grammar, but not as many

Stay tuned: Learn about LL(1) and LR(1) grammars in CMPU 331

Some ambiguities (e.g., dangling `else`) are easily handled with one token lookahead

Summary

Automata can be augmented with a memory storage to increase their power.

PDAs are finite automata equipped with a stack.

PDAs recognize precisely the context-free languages, which are a strict superset of the regular languages.

Deterministic PDAs are strictly weaker than nondeterministic PDAs.

Acknowledgments

This lecture incorporates material from:

Nancy Ide, Vassar College

Keith Schwarz, Stanford University

Michael Sipser, *Introduction to the Theory of Computation*

Jeffrey Ullman, Stanford University

Jennifer Walter, Vassar College