Automata Learning Reading Group

Different types of Automata

Jari Stenman

2013-11-26

Introduction

- Purpose:
  - Introduce (review?) some automata models
  - Show a simple but very useful model for undecidability results

Overview

- Turing Machines
- Linear Bounded Automata
- Pushdown Automata
- Counter Machines

Overview

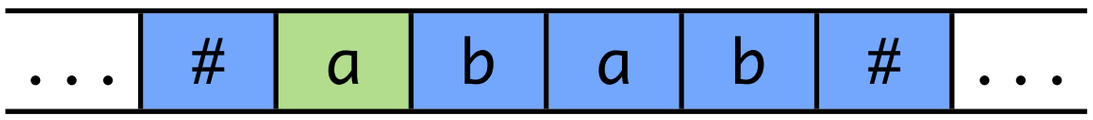
- Turing Machines
- Linear Bounded Automata
- Pushdown Automata
- Counter Machines

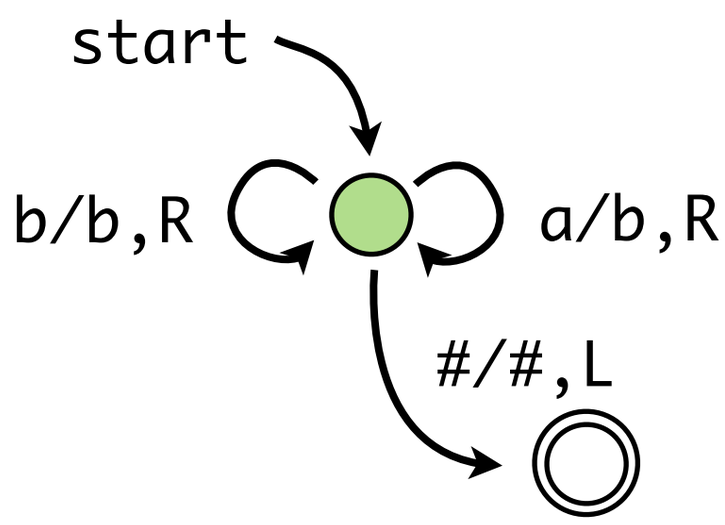
Turing Machines

Mandatory Picture

Turing Machines

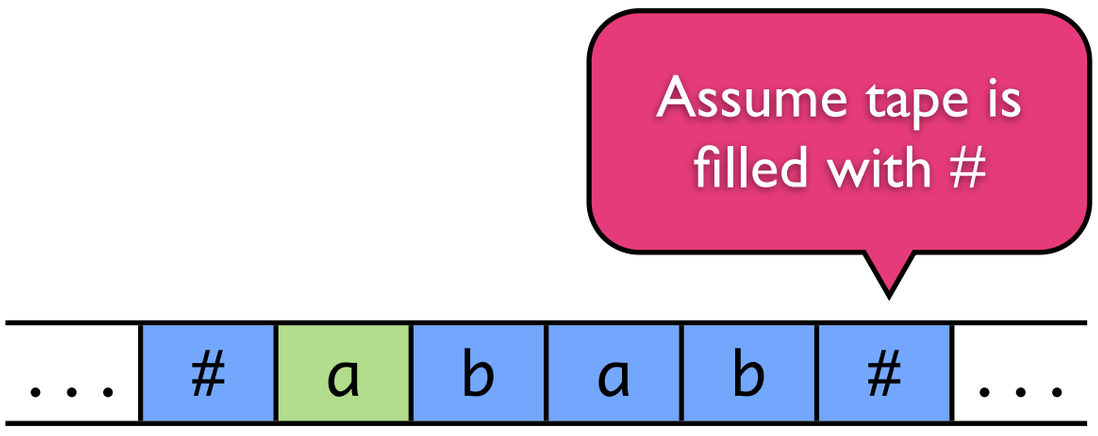
- Finite control
- Infinite storage tape

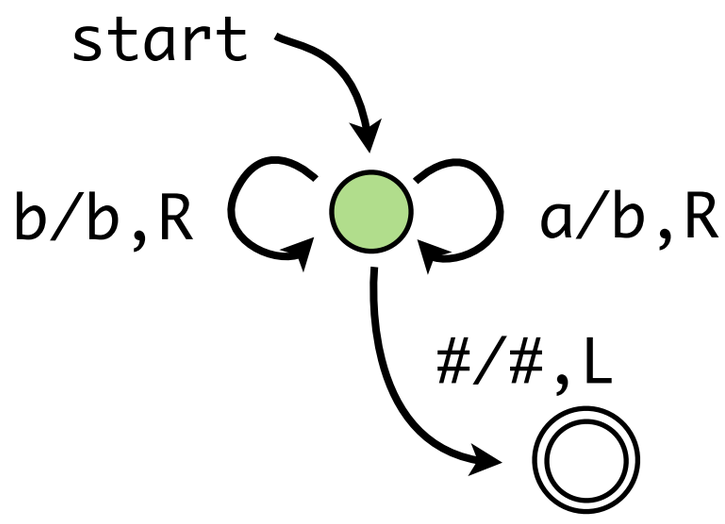




Turing Machines

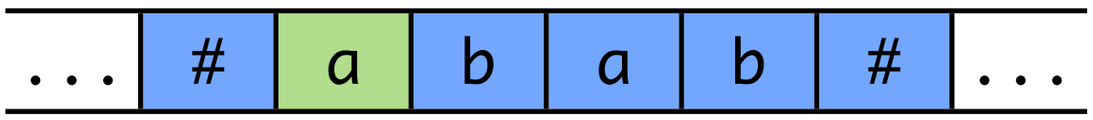
- Finite control
- Infinite storage tape

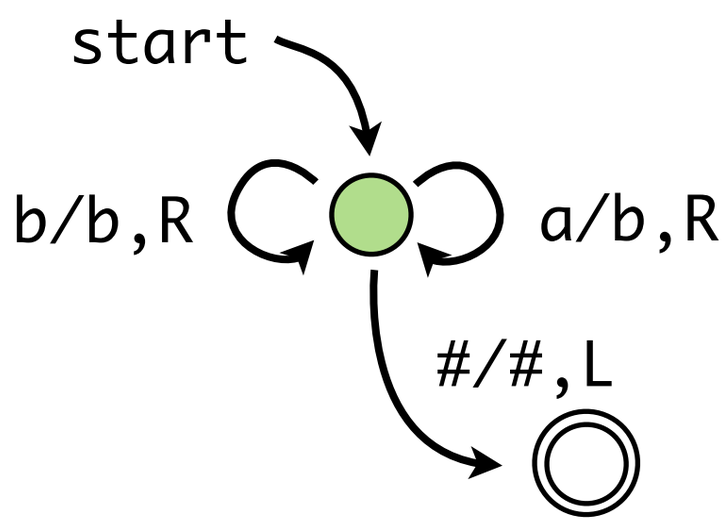




Turing Machines

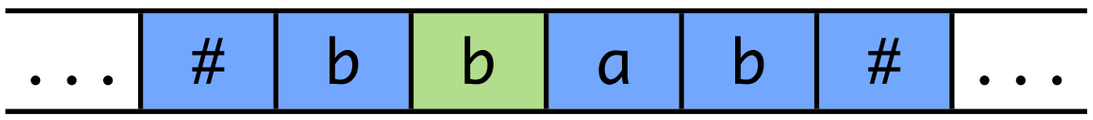
- Finite control
- Infinite storage tape

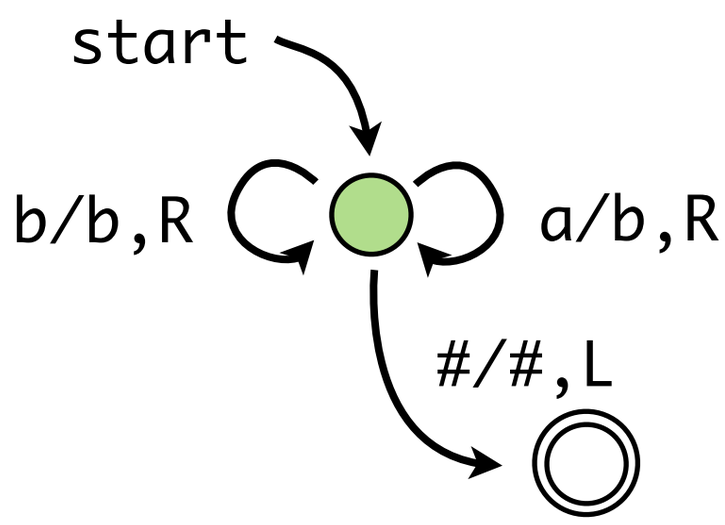




Turing Machines

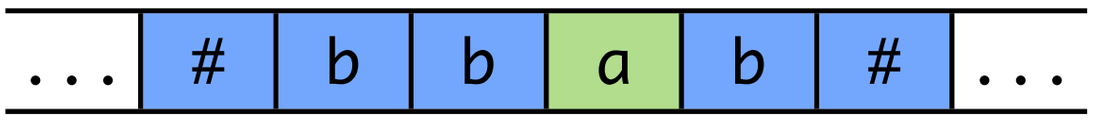
- Finite control
- Infinite storage tape

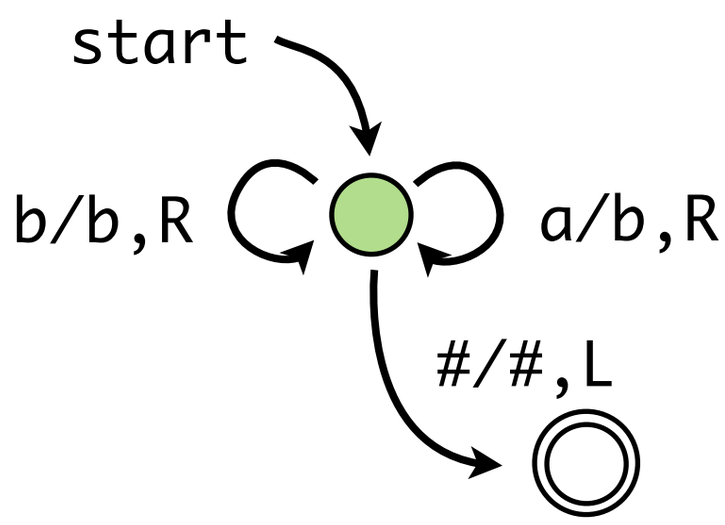




Turing Machines

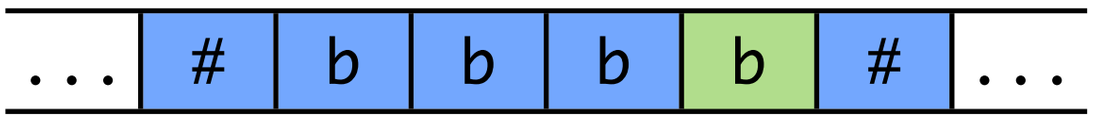
- Finite control
- Infinite storage tape

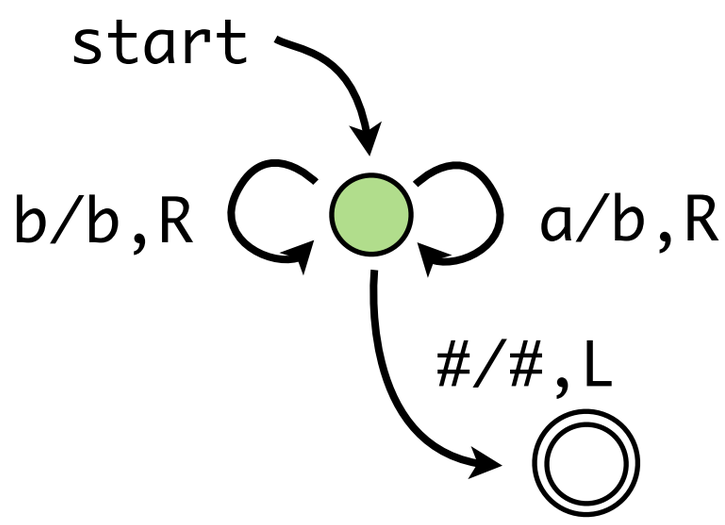




Turing Machines

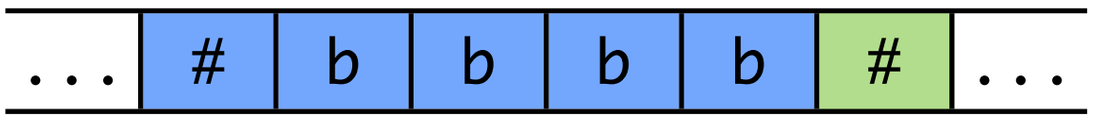
- Finite control
- Infinite storage tape

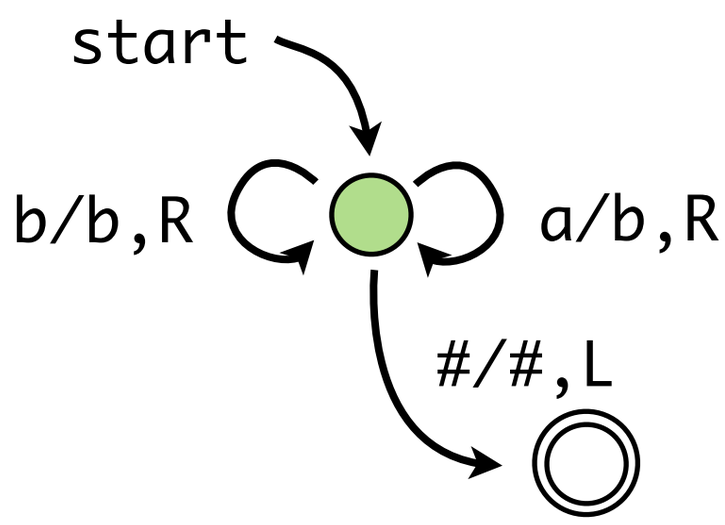




Turing Machines

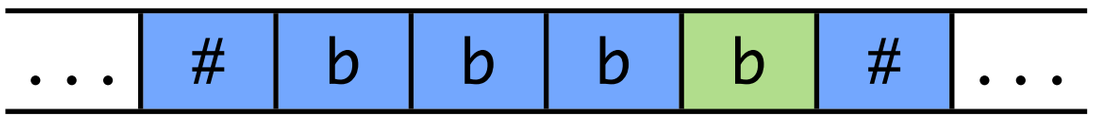
- Finite control
- Infinite storage tape

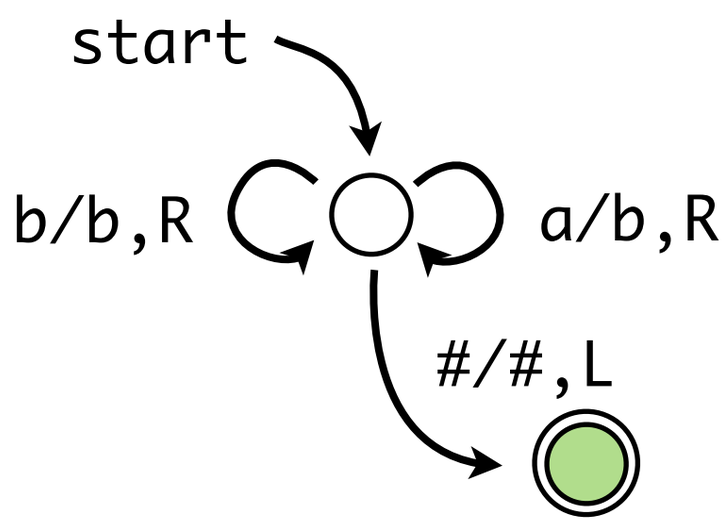




Turing Machines

- Finite control
- Infinite storage tape





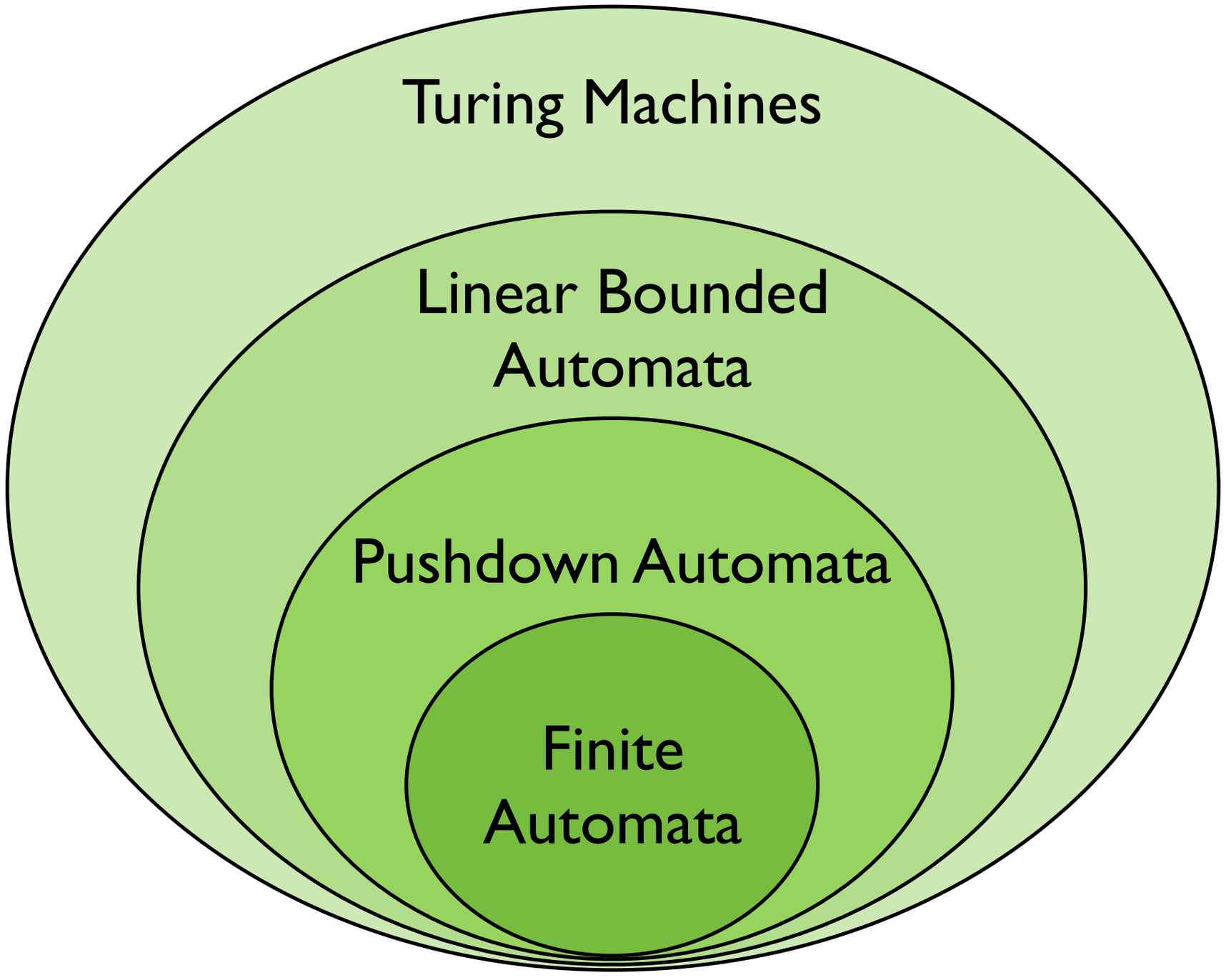
Turing Machines

- Finite control
- Infinite storage tape
- Capture the intuitive notion of computation
- “Everything” undecidable

Some History

1936	Turing	Turing Machines
1943	McCulloch & Pitts	Finite Automata
1955, 1956	Mealy, Moore	Finite State Transducers
1956	Chomsky	Chomsky Hierarchy
1960-1964	Myhill, Landweber, Kuroda	Linear Bounded Automata

Chomsky Hierarchy

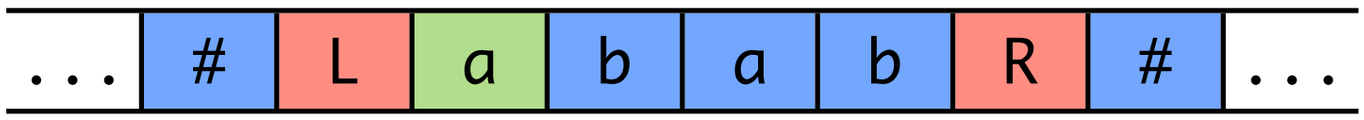


Overview

- Turing Machines
- Linear Bounded Automata
- Pushdown Automata
- Counter Machines

Linear Bounded Automata

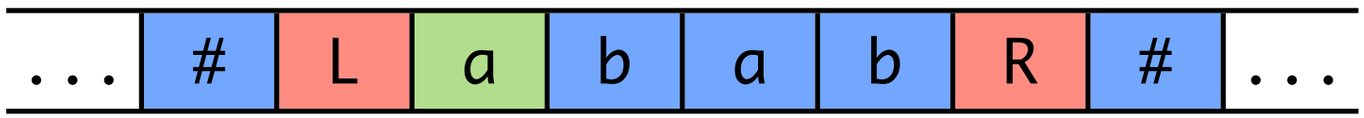
- Turing Machines with bounded tape



- Nondet. LBA accept exactly the context-sensitive languages

Linear Bounded Automata

- Turing Machines with bounded tape



- Nondet. LBA accept exactly the context-sensitive languages
- Open problem: Are det. LBA and nondet. LBA equivalent?

Overview

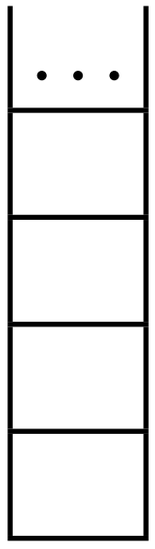
- Turing Machines
- Linear Bounded Automata
- Pushdown Automata
- Counter Machines

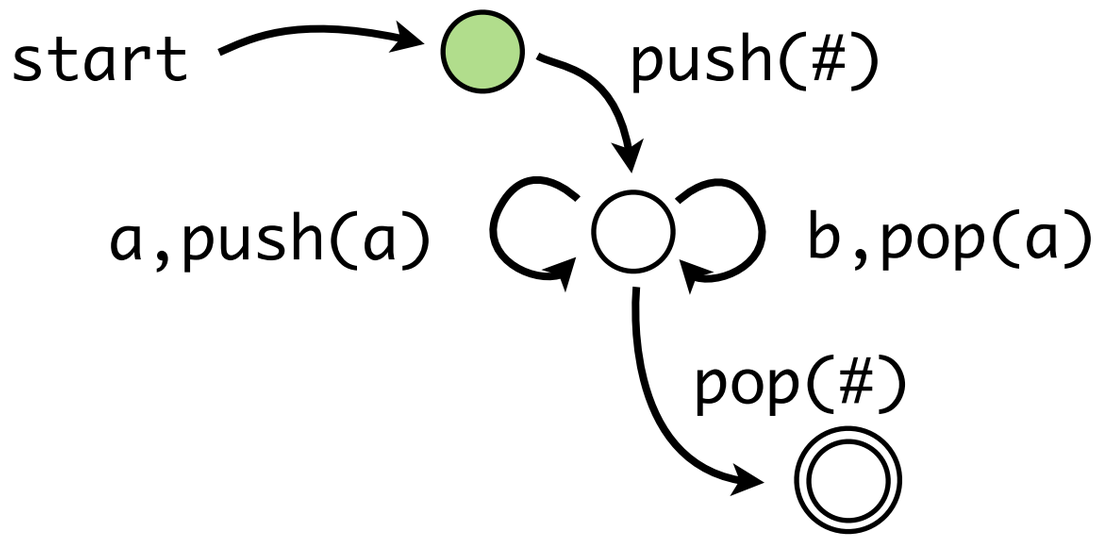
Pushdown Automata

- Finite control
- Unbounded stack

Input:

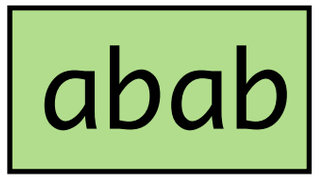
abab

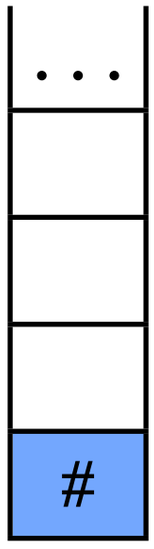


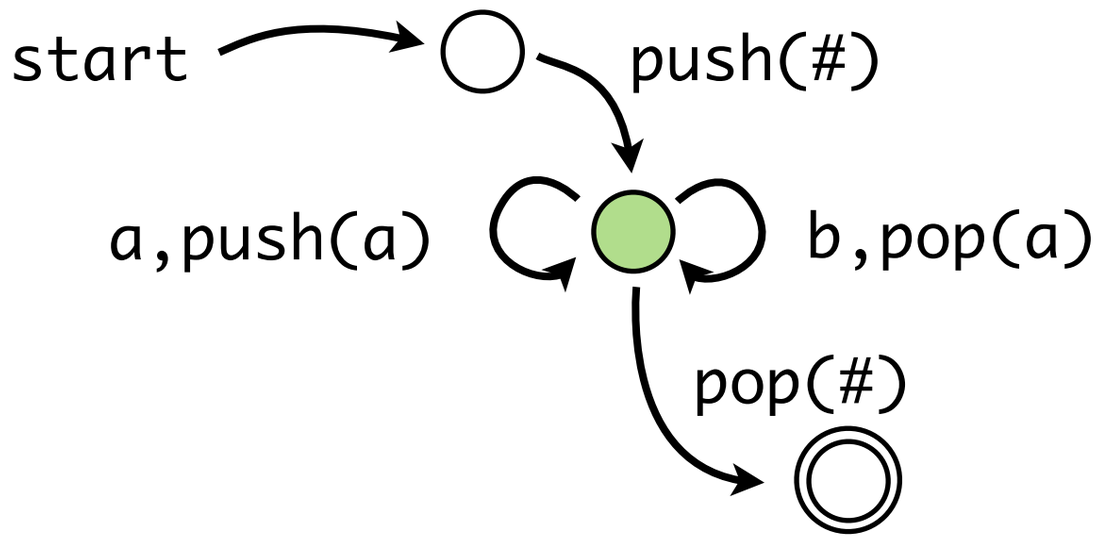


Pushdown Automata

- Finite control
- Unbounded stack

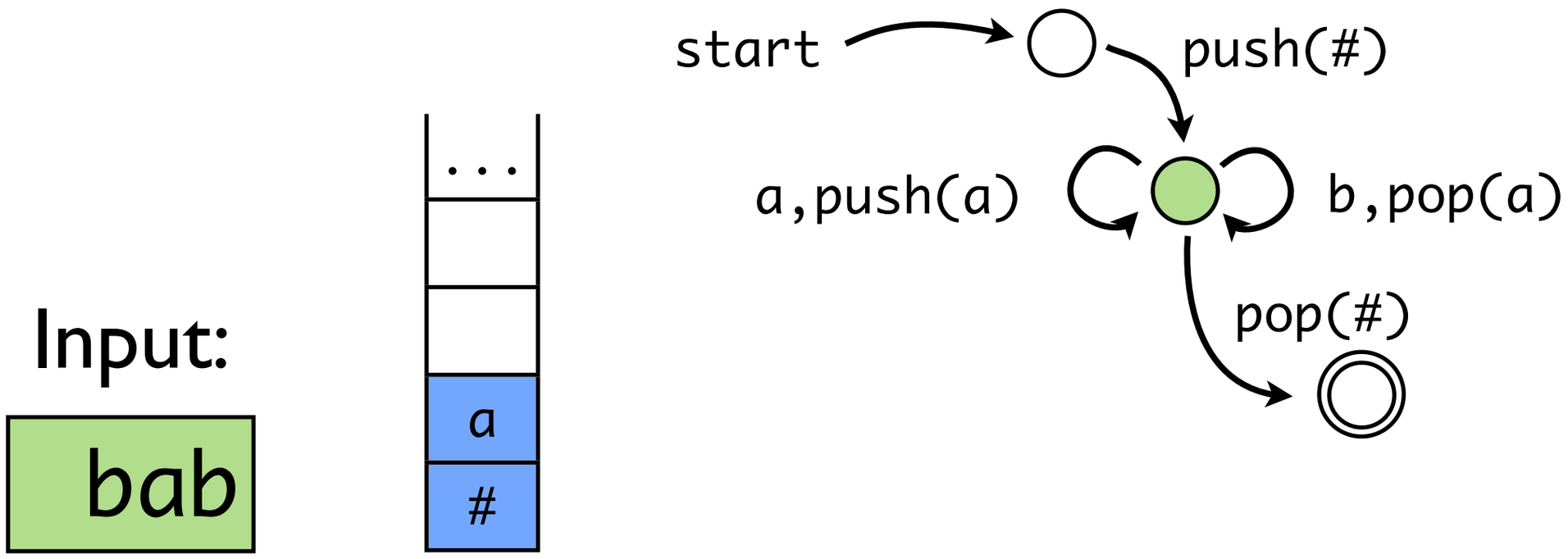






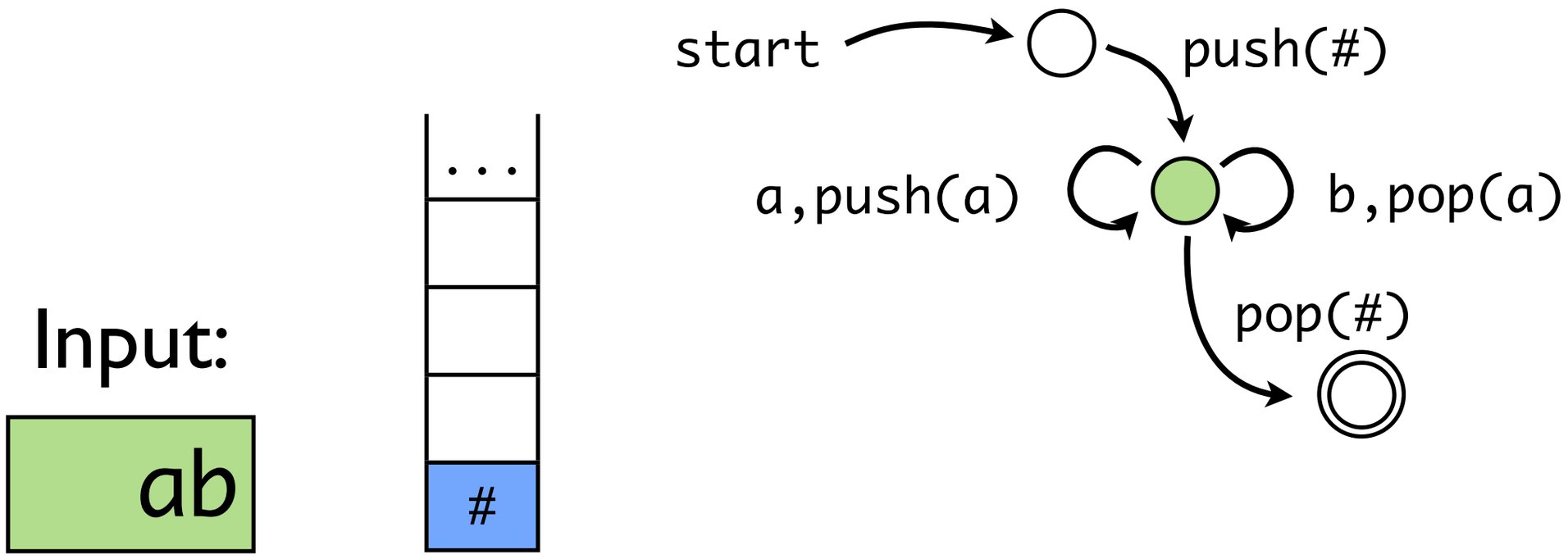
Pushdown Automata

- Finite control
- Unbounded stack



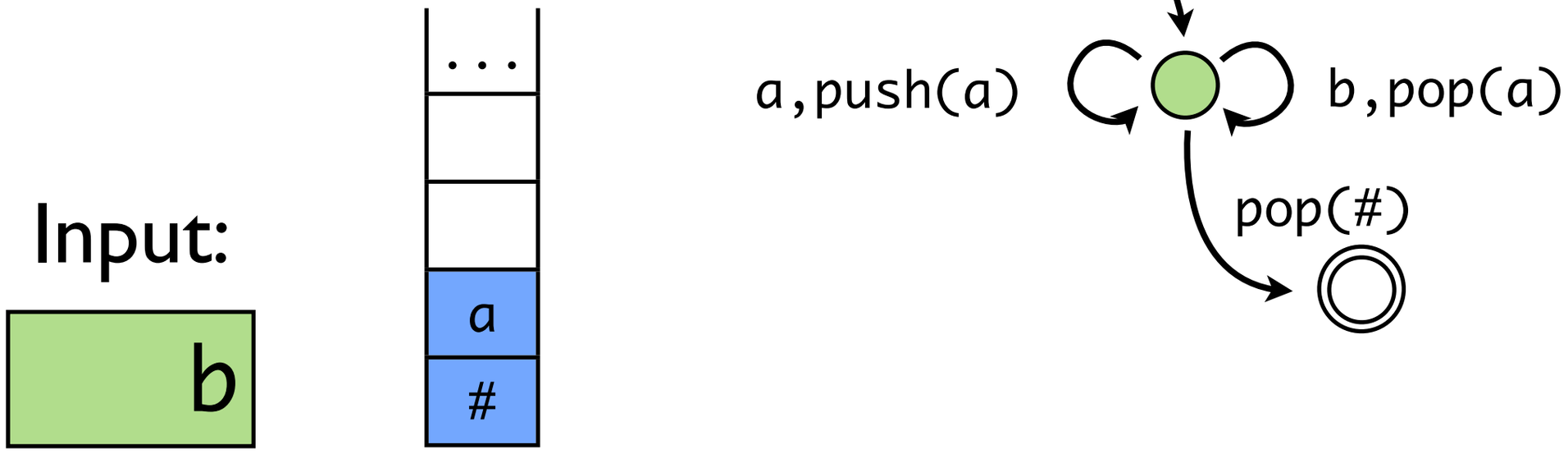
Pushdown Automata

- Finite control
- Unbounded stack



Pushdown Automata

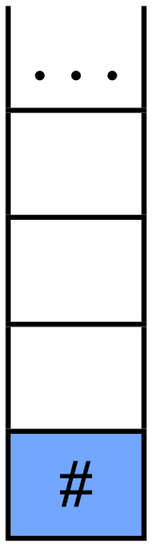
- Finite control
- Unbounded stack

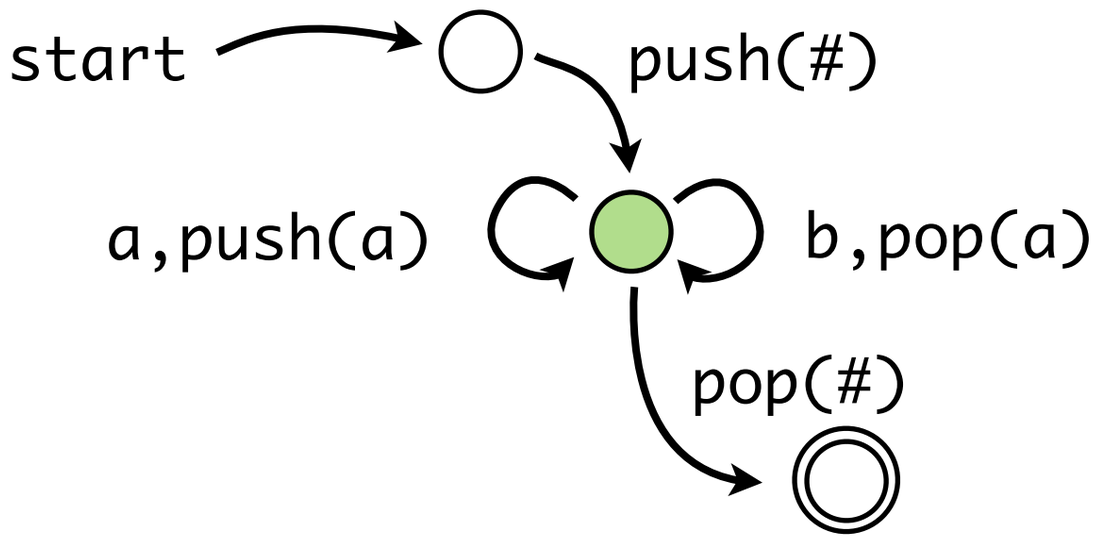


Pushdown Automata

- Finite control
- Unbounded stack

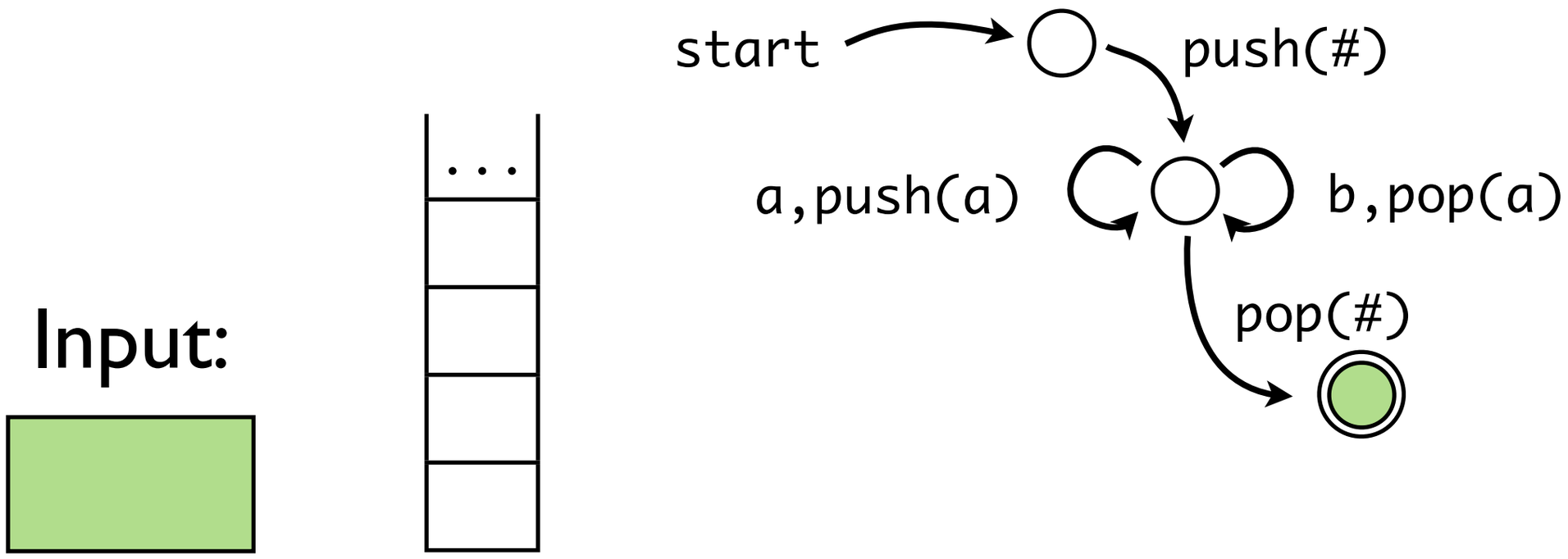
Input:





Pushdown Automata

- Finite control
- Unbounded stack



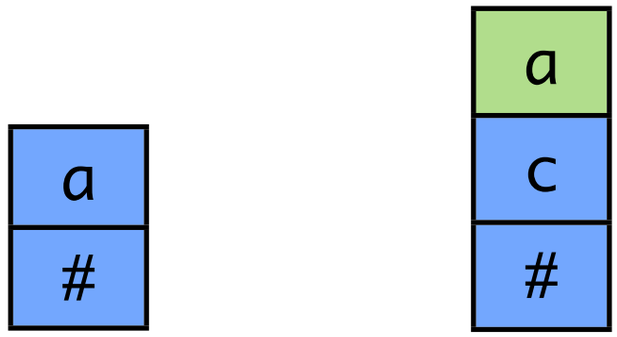
Pushdown Automata

- Finite control
- Unbounded stack
- Nondet. PDA accept exactly the context-free languages

Pushdown Automata

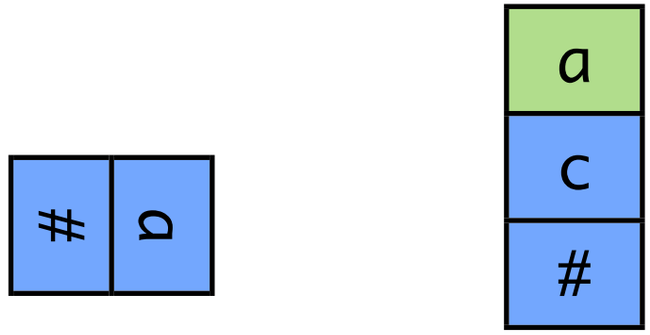
- Finite control
- Unbounded stack
- Nondet. PDA accept exactly the context-free languages
- What happens if we have several stacks?

Pushdown Automata



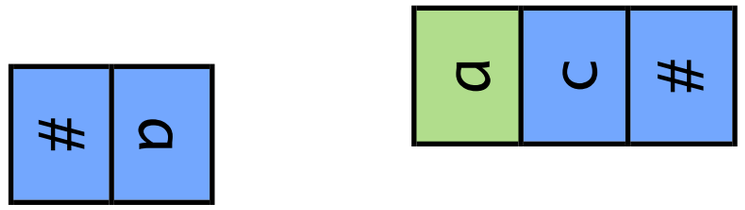
2 stacks

Pushdown Automata



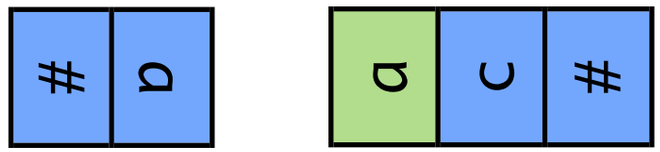
2 stacks

Pushdown Automata



2 stacks

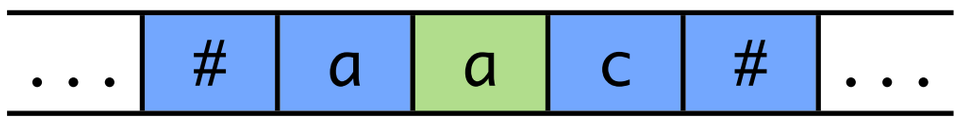
Pushdown Automata



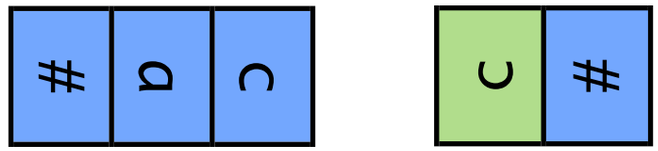
2 stacks

Pushdown Automata

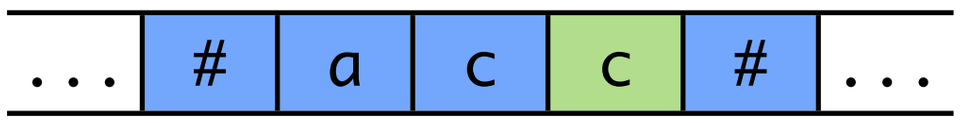
2 stacks



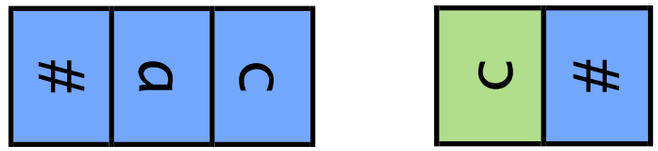
Pushdown Automata



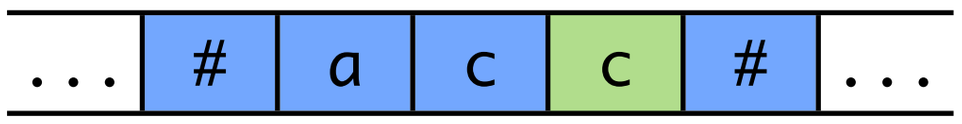
2 stacks



Pushdown Automata



2 stacks



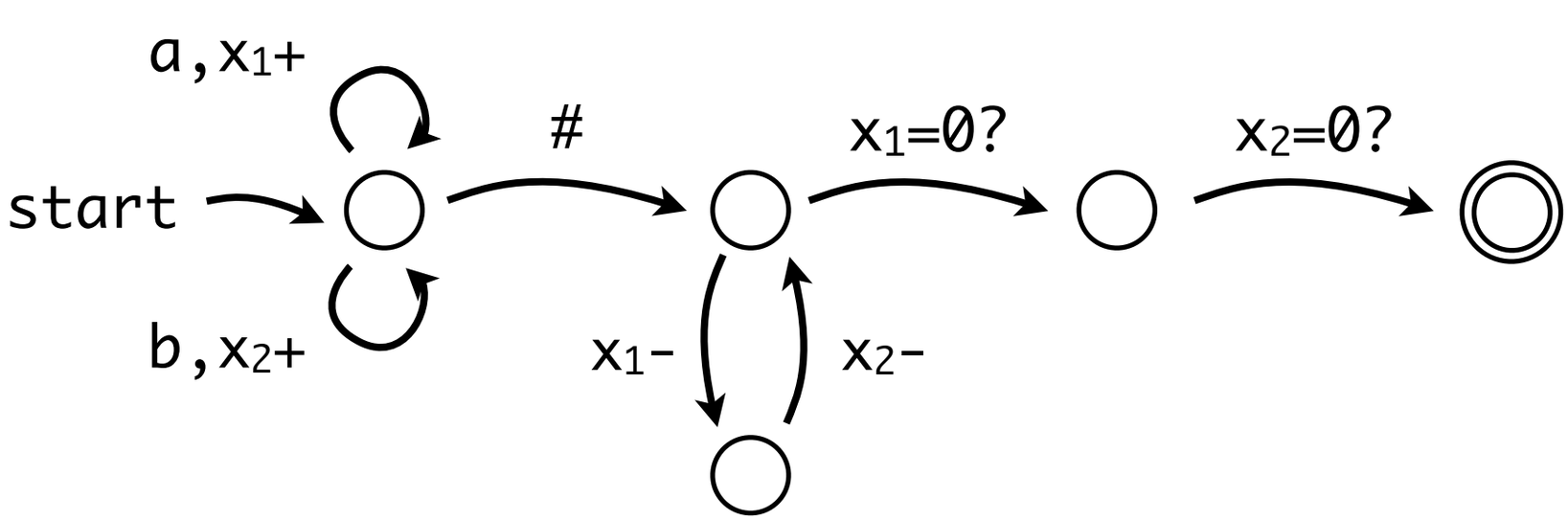
2-stack PDA are Turing-powerful!

Overview

- Turing Machines
- Linear Bounded Automata
- Pushdown Automata
- Counter Machines

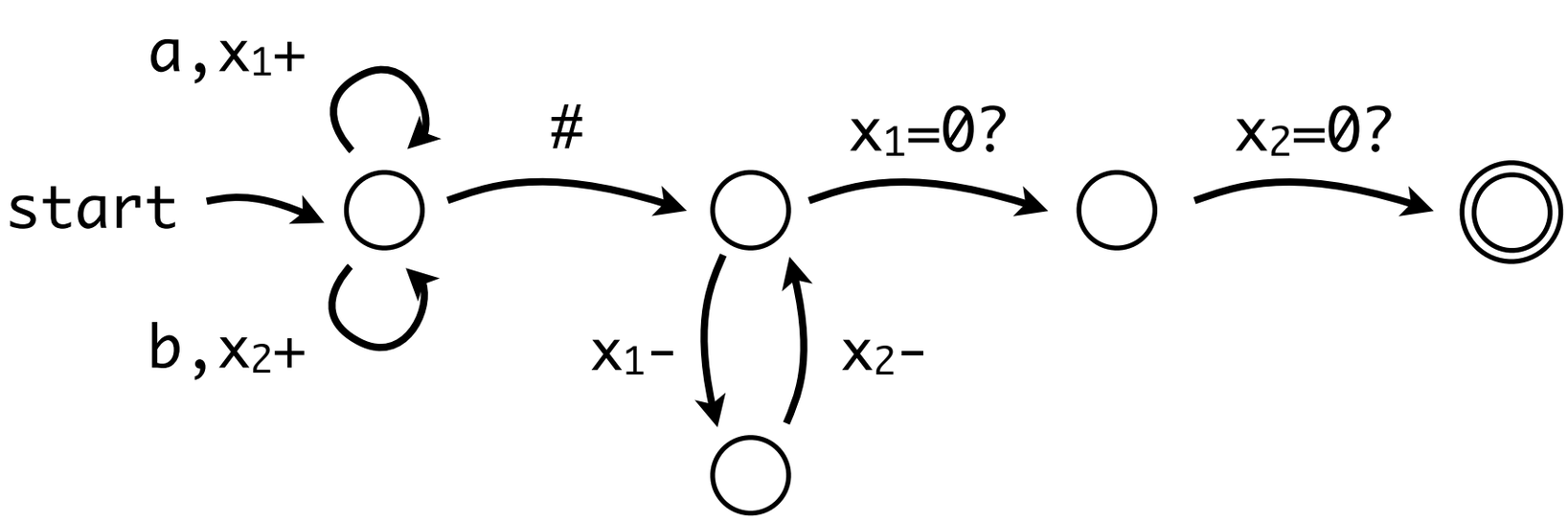
Counter Machines

- Finite control
- Counters storing natural numbers



Counter Machines

- Finite control
- Counters storing natural numbers



- How powerful is this model?

Simulating a Stack

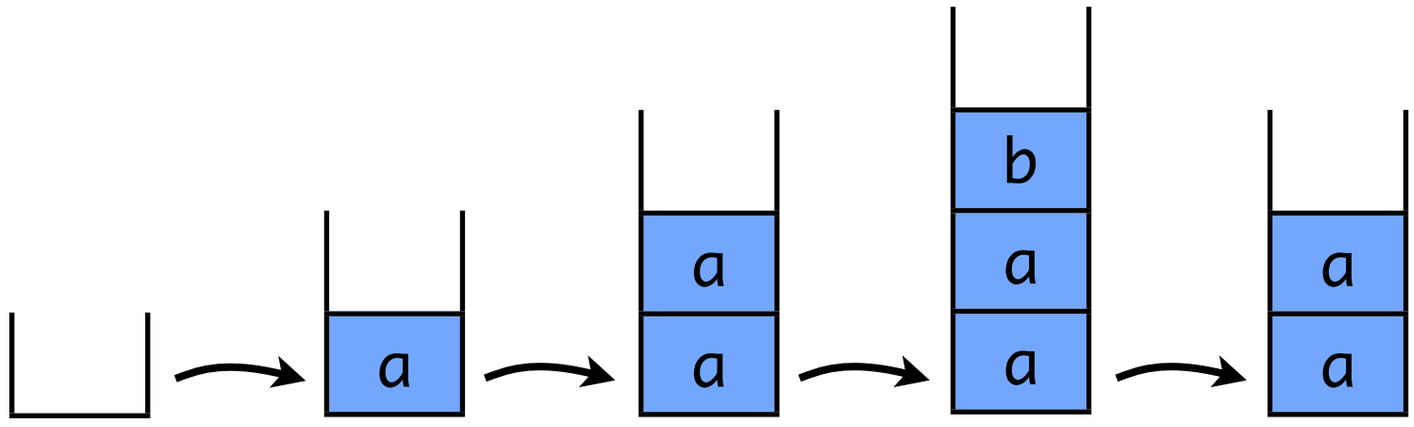
- A 2-CM can simulate a 1-stack PDA:
  - Encode stack as natural number
  - Implement push and pop for each stack symbol as operations on this number

Simulating a Stack

Example: $\Sigma = \{a, b\}$

Push *a*	Double x_1 and add 1
Push *b*	Double x_1 and add 2
Pop *a*	Check if x_1-1 ≡ 0 mod 2
Pop *b*	Check if x_1-2 ≡ 0 mod 2

Stack:



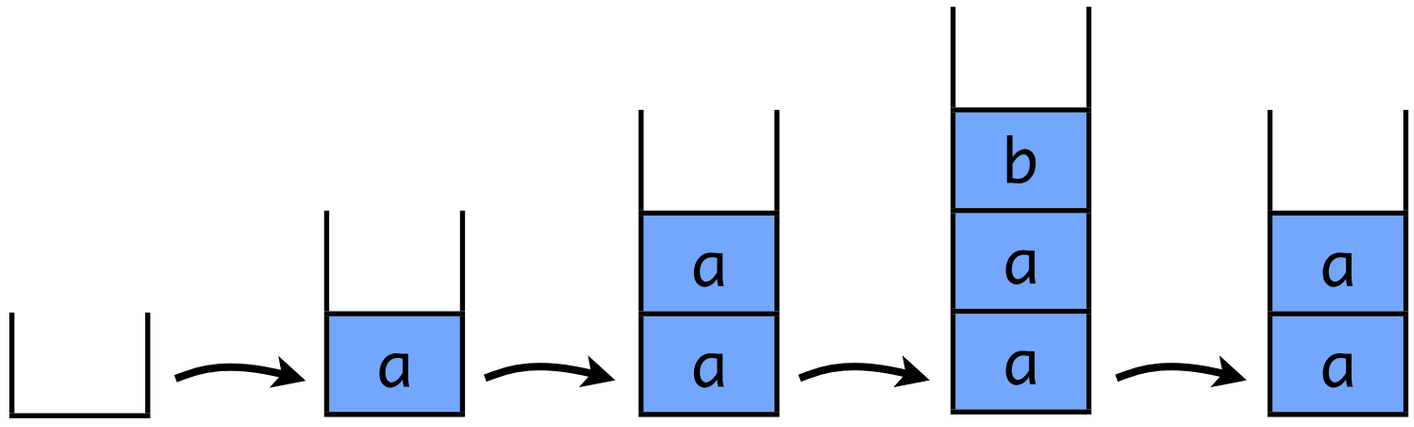
Counter: 0

Simulating a Stack

Example: $\Sigma = \{a, b\}$

Push *a*	Double x_1 and add 1
Push *b*	Double x_1 and add 2
Pop *a*	Check if x_1-1 ≡ 0 mod 2
Pop *b*	Check if x_1-2 ≡ 0 mod 2

Stack:

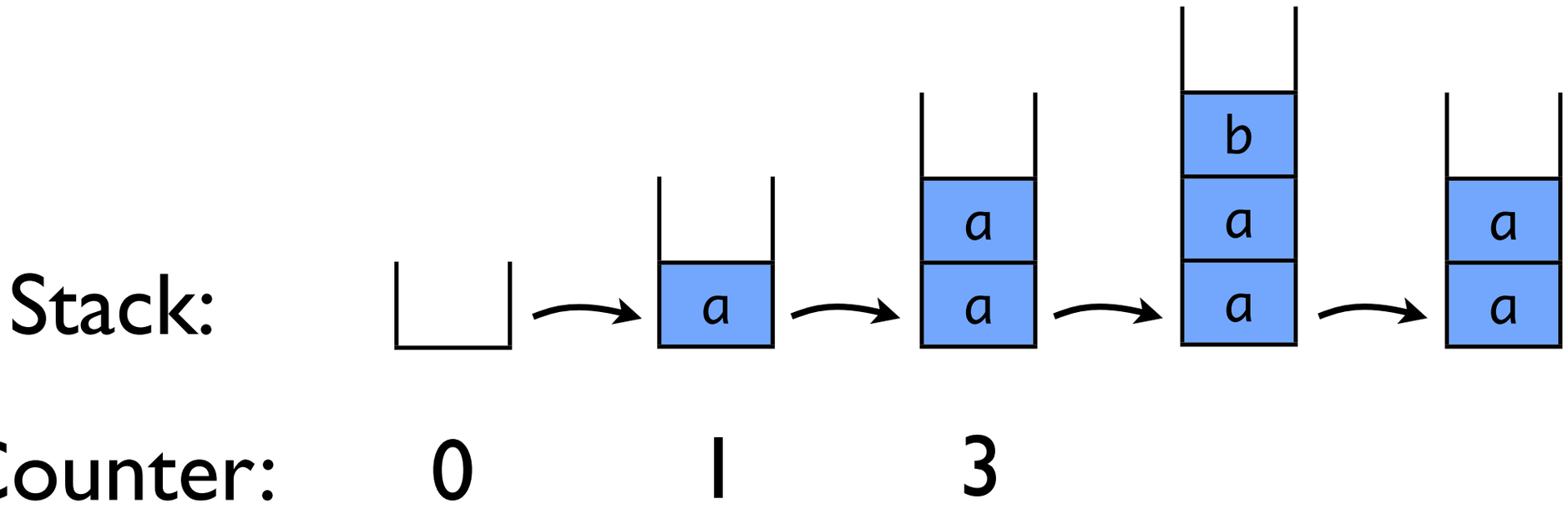


Counter: 0 1

Simulating a Stack

Example: $\Sigma = \{a, b\}$

Push *a*	Double x_1 and add 1
Push *b*	Double x_1 and add 2
Pop *a*	Check if x_1-1 ≡ 0 mod 2
Pop *b*	Check if x_1-2 ≡ 0 mod 2

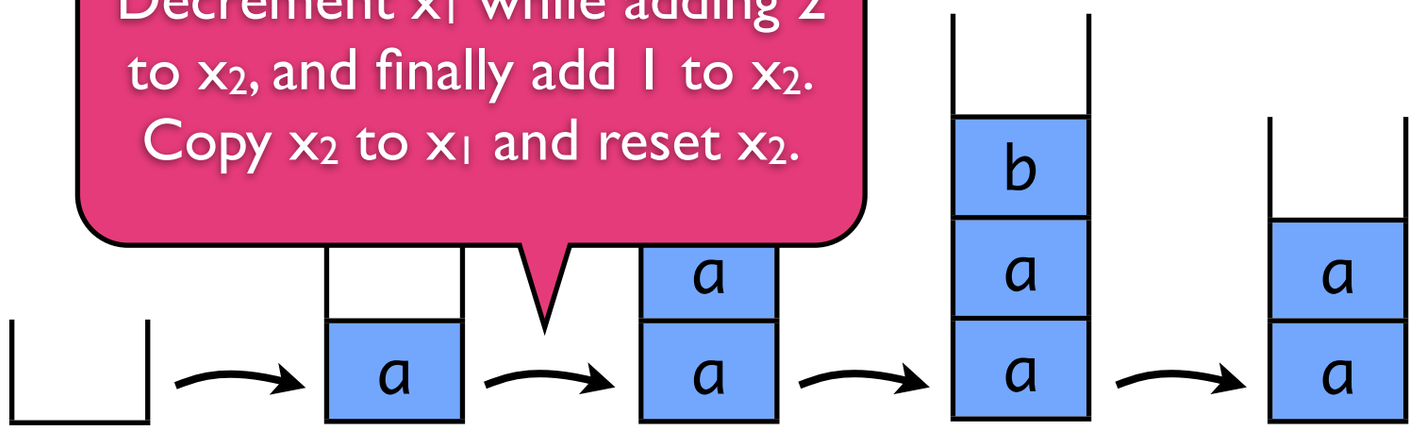


Simulating a Stack

Example: $\Sigma = \{a, b\}$

Push *a*	Double x_1 and add 1
Push *b*	Double x_1 and add 2
Pop *a*	Check if x_1-1 ≡ 0 mod 2
Pop *b*	Check if x_1-2 ≡ 0 mod 2

Decrement x_1 while adding 2 to x_2, and finally add 1 to x_2. Copy x_2 to x_1 and reset x_2.

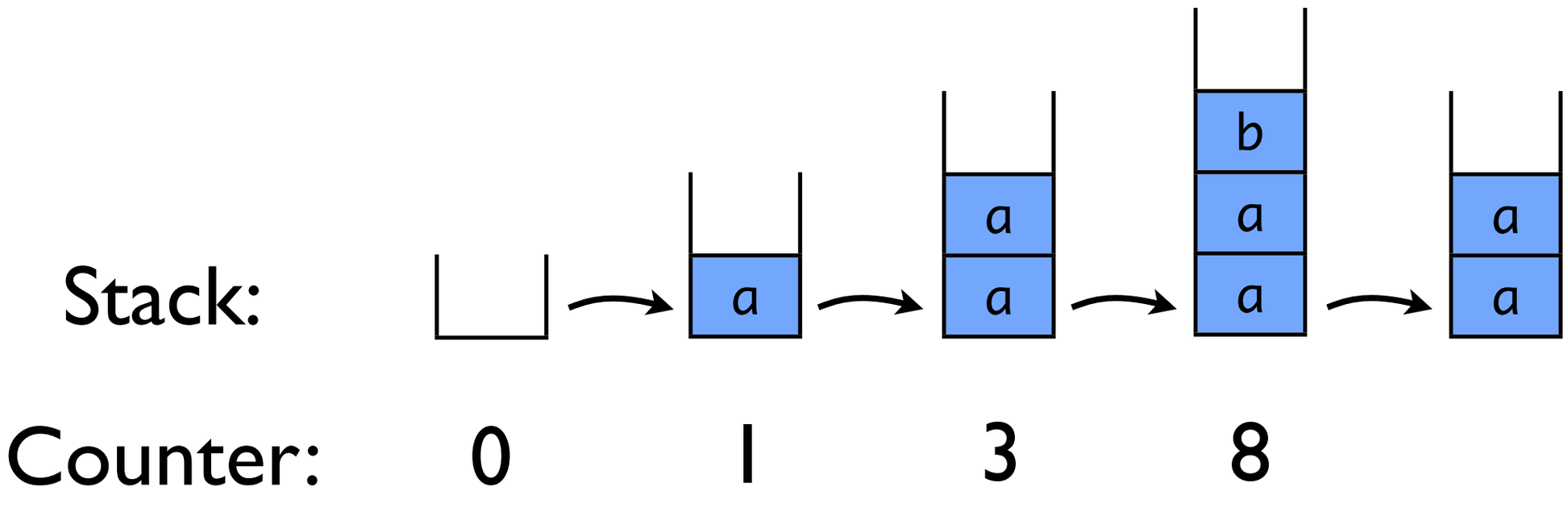


Counter: 0 1 3

Simulating a Stack

Example: $\Sigma = \{a, b\}$

Push *a*	Double x_1 and add 1
Push *b*	Double x_1 and add 2
Pop *a*	Check if x_1-1 ≡ 0 mod 2
Pop *b*	Check if x_1-2 ≡ 0 mod 2

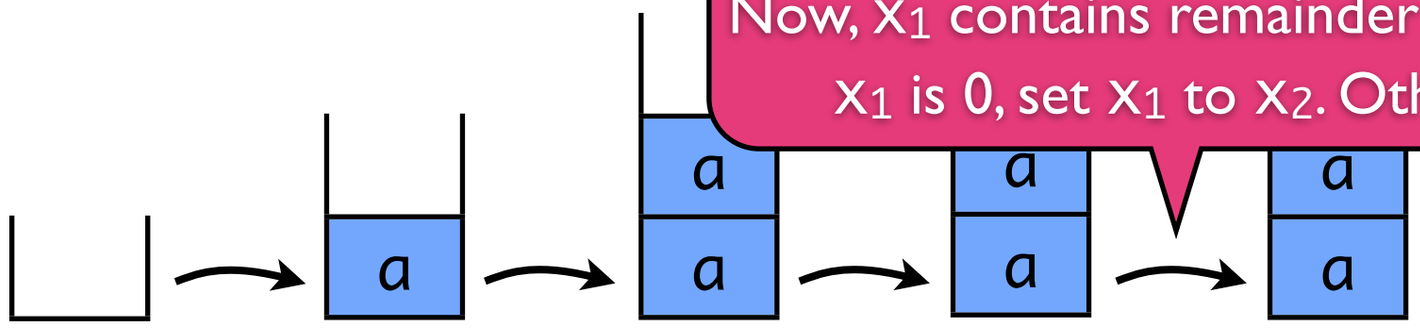


Simulating a Stack

Example: $\Sigma = \{a, b\}$

Push *a*	Double x_1 and add 1
Push *b*	Double x_1 and add 2
Pop *a*	Check if x_1-1 ≡ 0 mod 2
Pop *b*	Check if x_1-2 ≡ 0 mod 2

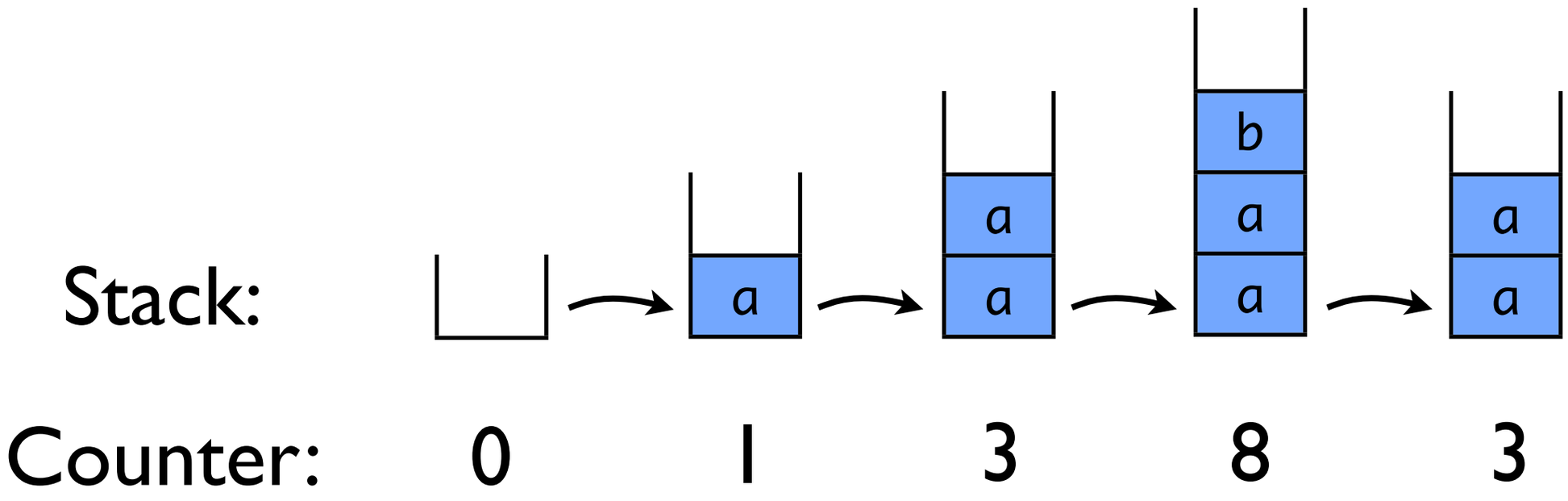
First, decrement by 1. Decrement x_1 by 2 while adding 1 to x_2, until no longer possible. Now, x_1 contains remainder and x_2 quotient. If x_1 is 0, set x_1 to x_2. Otherwise, abort.



Simulating a Stack

Example: $\Sigma = \{a, b\}$

Push *a*	Double x_1 and add 1
Push *b*	Double x_1 and add 2
Pop *a*	Check if x_1-1 ≡ 0 mod 2
Pop *b*	Check if x_1-2 ≡ 0 mod 2



Counter Machines

- A 2-CM can simulate a 1-stack PDA.
- A 4-CM can simulate a 2-stack PDA.
- So 4-CM are Turing powerful!

Counter Machines

- But there is more:
  - A 2-CM can simulate an n-CM for any n.
    - Encode k_1,k_2,k_3,k_4 as $2^{k1}3^{k2}5^{k3}7^{k4}$

Counter Machines

- But there is more:
  - A 2-CM can simulate an n-CM for any n.
    - Encode k_1,k_2,k_3,k_4 as $2^{k1}3^{k2}5^{k3}7^{k4}$
    - 2-CM are Turing powerful!

Counter Machines

- Easy way to prove undecidability: show that if your problem is decidable, you can solve some undecidable problem for 2-CM.
- Simulate 3 operations (for each counter):
  - Check for 0
  - Increment counter
  - Decrement counter

Thank You!