CSCI 340: Computational Models

Minsky's Theorem

Chapter 21

Department of Computer Science

The Two-Stack PDA

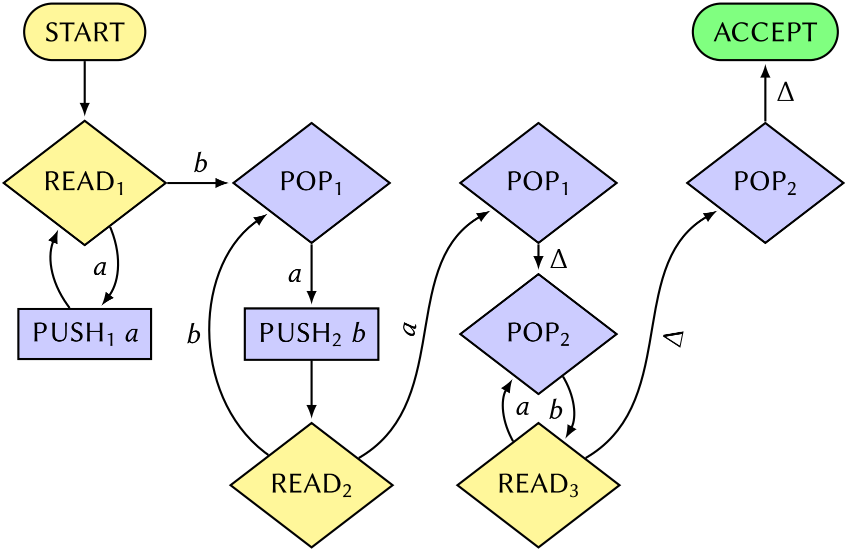
- Turing machines never seemed like a natural extension — comparing FAs to PDAs
- There is no such extension between PDAs and TMs
- *Insight:* the addition of a PUSHDOWN STACK made a considerable improvement in the power of an FA
- *Idea:* What would happen if we add **another** PUSHDOWN STACK to a PDA? or 3? or 70?

Two-Pushdown Stack Machine — 2PDA

Definition

- A **two-pushdown stack machine**, denoted 2PDA, is like a PDA except that it has two PUSHDOWN STACKS
  1. STACK_1
  2. STACK_2
- When we push a character, we must indicate which stack we are PUSHing onto. We do this by renaming PUSH to PUSH_1 and introduce a PUSH_2 state.
- When we pop a character from a stack, we need to indicate which stack we are POPing from. We do this by renaming POP to POP_1 and introduce a POP_2 state.
- We also will insist that 2PDAs are **deterministic**

2PDA Example



Just Another TM

Theorem

2PDA = TM

In other words, any language accepted by a 2PDA can be accepted by some TM and any language accepted by a TM can be accepted by some 2PDA.

Proof.

Part 1 — Modeling a 2PDA on a TM

- A 2PDA has three locations where it can store information:
    1. INPUT TAPE
    2. STACK_1
    3. STACK_2
- A TM has one location where it can store information: the **TAPE**
- Model the **TAPE** to store INPUT TAPE, STACK_1, and STACK_2

(continued...)

Just Another TM

Proof.

- Assume # and $ are symbols not part of Σ or Γ
- Store on the **TAPE** the following:

$$\text{INPUT TAPE} \quad \# \quad \text{STACK}_1 \quad \$ \quad \text{STACK}_2$$

- *Always* have the TAPE HEAD point at the # after any operation
- Simulating READ
  1. Move the TAPE HEAD to the left and find the rightmost “front” Δ
  2. Move one to the right to find the next *input letter*
  3. If this character is #, the input has been exhausted
  4. Otherwise, change this character into Δ
  5. Branch according to what was read. In each branch, move down to the #, then start simulating the next state

(continued...)

Just Another TM

Proof.

- Simulating POP_1 and POP_2
  - Move to \$ if POP_2; otherwise stay at #
  - Move to the right. If \$ is read, then $STACK_1$ is empty
  - Else, we are removing the current character from our stack.
  - Branch to a unique path based on the character read
  - Call the DELETE subprogram
  - Rewind back to # and start simulating the next state
- Simulating $PUSH_1$ and $PUSH_2$
  - Move to \$ if $PUSH_2$; otherwise stay at #
  - Call the INSERT subprogram
  - Rewind back to # and start simulating the next state
- When the 2PDA branches to ACCEPT, enter HALT

(continued...)

Just Another TM

Proof.

Part 2 — Modeling a TM on a 2PDA

- Or... how about we don't do that
- Instead, why don't we model a Post Machine on a 2PDA?
  1. Transfer all of the PM STORE to STACK_1 (use STACK_2 as buffer to maintain order)
  2. Emulate ADD X by moving everything from STACK_1 to STACK_2, PUSHing X onto STACK_1, then POP everything from STACK_2 back to STACK_1
  3. Emulate READ by just calling POP_1
  4. REJECTs can be discarded or kept the same
  5. ACCEPTs remain exactly the same
- *Key Insight:* STACK_2 is only used to initialized STACK_1 and to simulate ADD

We have now shown 2PDA $\subseteq$ TM and TM $\subseteq$ 2PDA □

nPDAs

Theorem

Any language accepted by a PDA with n STACKs (where n is 2 or more), called an nPDA, can also be accepted by some TM. In power we have:

$$nPDA = TM \quad \text{if } n \geq 2$$

Proof.

- Use similar representation of 2PDAs on a TM by introducing new separators: $\#_1, \#_2, \dots, \#_n$
- Relevant PUSH and POP operations will function on the TM
- Therefore, nPDA = TM
- 2PDA was already determined to be as powerful as TM
- 2PDA = nPDA □

FA = TG = NFA < DPDA < PDA < 2PDA = nPDA = PM = TM

An Aside — Structure of the Book

Part 1

Regular Expressions, Finite Automata, Transition Graphs, Kleene's Theorem, Finite Automata with Output, Regular Languages, Nonregular Languages (Pumping Lemma), Decidability

All of these are equivalent to a 0PDA

Part 2

Context-Free Grammars, Grammatical Format, Pushdown Automata, CFG=PDA, Non-Context-Free Languages (Pumping Lemma), Context-Free Languages, Decidability

All of these are equivalent to a 1PDA

Part 3

Turing Machines, Post Machines, Minsky's Theorem...

All of these are equivalent to a 2PDA

Homework 11a

1. [4pts each] VERYEQUAL is the language ($\Sigma = \{a\ b\ c\}$) as all strings that have as many total a's as total b's as total c's
   - Draw a TM that accepts VERYEQUAL
   - Draw a PM that accepts VERYEQUAL
   - Draw a 3PDA that accepts VERYEQUAL
   - Draw a 2PDA that accepts VERYEQUAL
2. [4pts] Draw a 2PDA that accepts EVEN-EVEN and keeps at most two letters in its STACKs