91.304 Foundations of (Theoretical) Computer Science

Chapter 2 Lecture Notes (Section 2.2: Pushdown Automata)

Prof. Karen Daniels, Fall 2012

with acknowledgement to:

-Sipser *Introduction to the Theory of Computation* textbook and

-Dr. David Martin

Overview

- New computational model:
  - Pushdown Automata (like NFA, but add a stack)
    - Definition, Examples
- Equivalence with Context-Free Grammars
  - Theorem 2.20: A language is context-free iff some pushdown automaton recognizes it.
  - Lemma 2.21 ($\Longrightarrow$) If a language is context-free, then some pushdown automaton recognizes it.
  - Lemma 2.27 ($\Longleftarrow$) If a pushdown automaton recognizes some language, then it is context-free.

Pushdown Automata Definition

- Like NFA, but add a stack

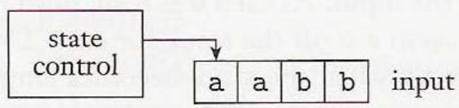


FIGURE 2.11
Schematic of a finite automaton

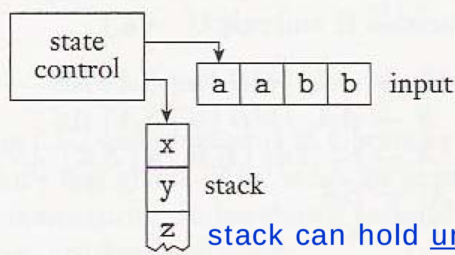


FIGURE 2.12
Schematic of a pushdown automaton

Source: Sipser Textbook

Pushdown Automata Definition

Formal Definition (6-tuple uses nondeterminism):

Nondeterministic PDA's are more powerful than deterministic ones. We focus on nondeterministic ones because they are as powerful as context-free grammars.

DEFINITION 2.13

A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet, *Each "thread" has its own stack.*
4. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

$\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$

Source: Sipser Textbook

Pushdown Automata Definition

- Formal Definition: Specification of F, δ

A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows. It accepts input w if w can be written as $w = w_1 w_2 \cdots w_m$, where each $w_i \in \Sigma_\varepsilon$ and sequences of states $r_0, r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ exist that satisfy the following three conditions. The strings s_i represent the sequence of stack contents that M has on the accepting branch of the computation.

1. $r_0 = q_0$ and $s_0 = \varepsilon$. This condition signifies that M starts out properly, in the start state and with an empty stack.
2. For $i = 0, \ldots, m-1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$. This condition states that M moves properly according to the state, stack, and next input symbol.
3. $r_m \in F$. This condition states that an accept state occurs at the input end.

Source: Sipser Textbook

Pushdown Automata Examples

EXAMPLE 2.14

The following is the formal description of the PDA (page 110) that recognizes the language $\{0^n1^n \mid n \geq 0\}$. Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0,1\}$,

$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, and

δ is given by the following table, wherein blank entries signify $\emptyset$.

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

\$ for empty stack test

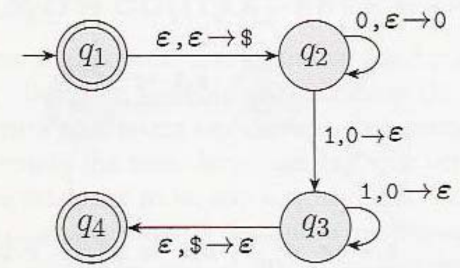


FIGURE 2.15
State diagram for the PDA M_1 that recognizes $\{0^n1^n \mid n \geq 0\}$

not regular!

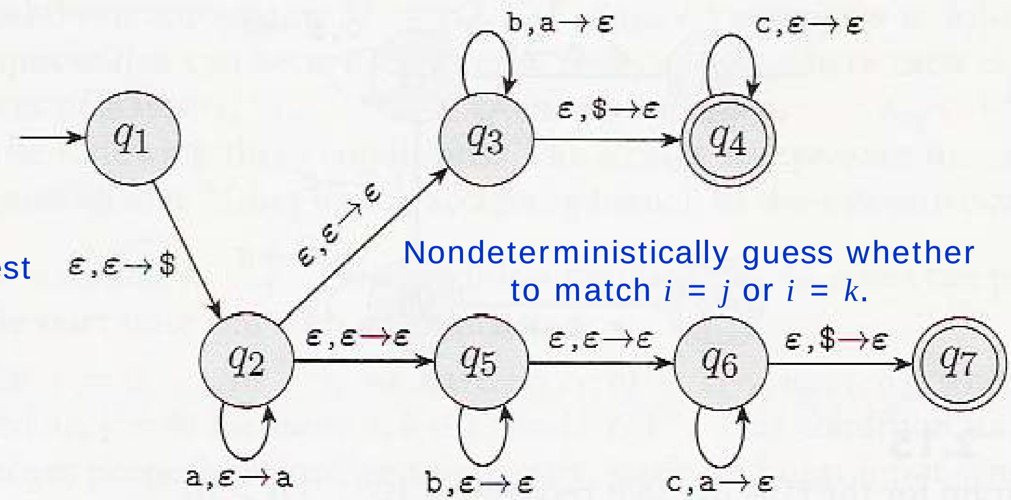
a,*b* → *c* means: when machine is reading *a* from input, it replaces *b* (from top of stack) with *c*.

Source: Sipser Textbook

Pushdown Automata Examples

- Example 2.16

$ for empty stack test



Nondeterministically guess whether to match $i = j$ or $i = k$.

FIGURE 2.17
State diagram for PDA M_2 that recognizes $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

$a,b \rightarrow c$ means: when machine is reading a from input, it replaces b (from top of stack) with c.

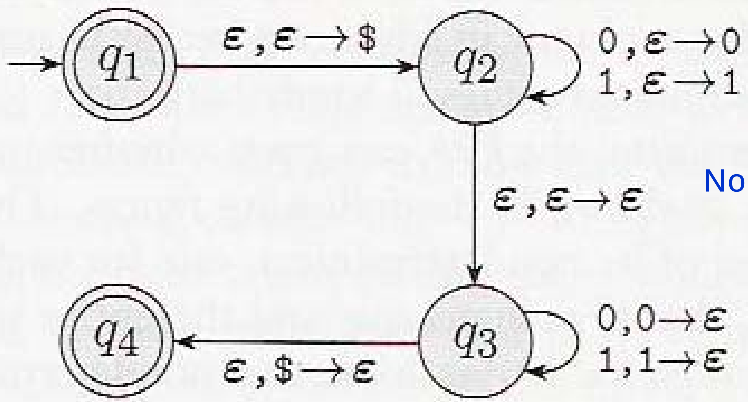
Nondeterminism is *essential* for recognizing this language with a PDA!

Source: Sipser Textbook

Pushdown Automata Examples

- Example 2.18

$ for empty stack test



Nondeterministically guess end of w.

FIGURE 2.19
State diagram for the PDA M_3 that recognizes $\{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$

$a,b \rightarrow c$ means: when machine is reading a from input, it replaces b (from top of stack) with c.

Source: Sipser Textbook

Equivalence with Context-Free Grammars

(for nondeterministic PDAs)

- **Theorem 2.20**: A language is context-free iff some pushdown automaton recognizes it.
- **Lemma 2.21** ($\Rightarrow$) If a language is context-free, then some pushdown automaton recognizes it.
- **Lemma 2.27** ($\Leftarrow$) If a pushdown automaton recognizes some language, then it is context-free.

Source: Sipser Textbook

Equivalence with Context-Free Grammars

- Lemma 2.21 ($\Rightarrow$) If a language is context-free, then some pushdown automaton recognizes it.
  - Proof Idea: Produce a pushdown automaton P from the context-free grammar G for the context-free language.
    - If G generates w, then P accepts its input w by checking if there's a derivation for w.
      - Each step of derivation yields an intermediate string.
        - Keep only part of this string on the stack.
        - (see next slide for illustration)
      - Nondeterminism guesses sequence of correct substitutions for a derivation.

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.21 ($\Rightarrow$)

- Proof Idea (again): Produce a pushdown automaton P from the context-free grammar G for the context-free language.
  - Each step of derivation yields an intermediate string.
    - Storing entire intermediate string on stack makes may not allow PDA to find variables in intermediate string to make substitutions.
    - Fix: Essentially keep only part of this string on the stack, starting with 1st variable. (terminals temporarily pushed onto stack, then matched with input and popped off)

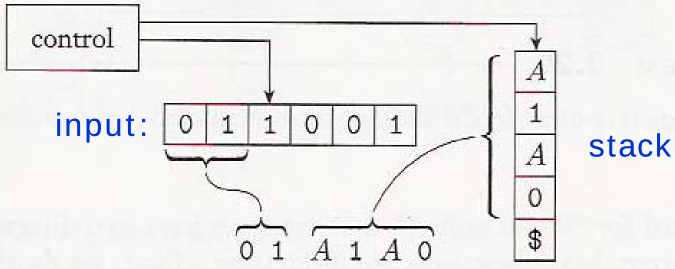


FIGURE 2.22
P representing the intermediate string $01A1A0$

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.21 (⇒)

- Proof Idea (again): Produce a pushdown automaton P from the context-free grammar G for the context-free language.

The following is an informal description of P.

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

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.21 (⇒)

- Proof Idea (again): Produce a pushdown automaton P from the context-free grammar G for the context-free language.
  - Substituting string $u = u_1 \cdots u_l$ on right-hand side of a rule.
    - $(r,u) \in \delta(q,a,s)$ means when P is in state q, a is next input symbol, and s is symbol on top of stack, P reads a, pops s, pushes u onto stack and goes to state r.

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

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

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

$\vdots$

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

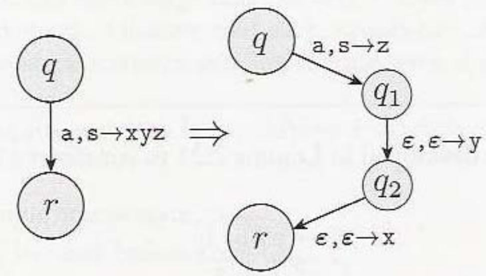


FIGURE 2.23
Implementing the shorthand $(r, xyz) \in \delta(q, a, s)$

(note reverse order)

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.21 ($\Rightarrow$)

- Proof Idea (again): Produce a pushdown automaton P from the context-free grammar G for the context-free language.

Recall informal description of P:

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

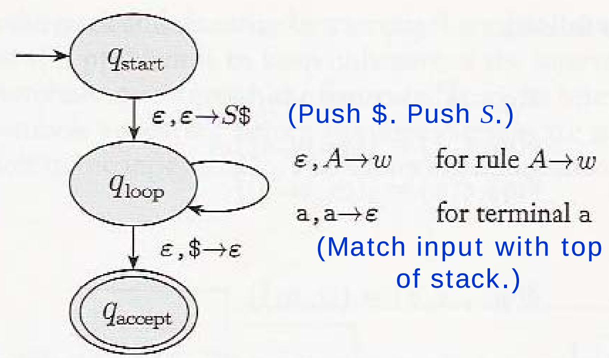


FIGURE 2.24
State diagram of P

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.21 ($\Rightarrow$)

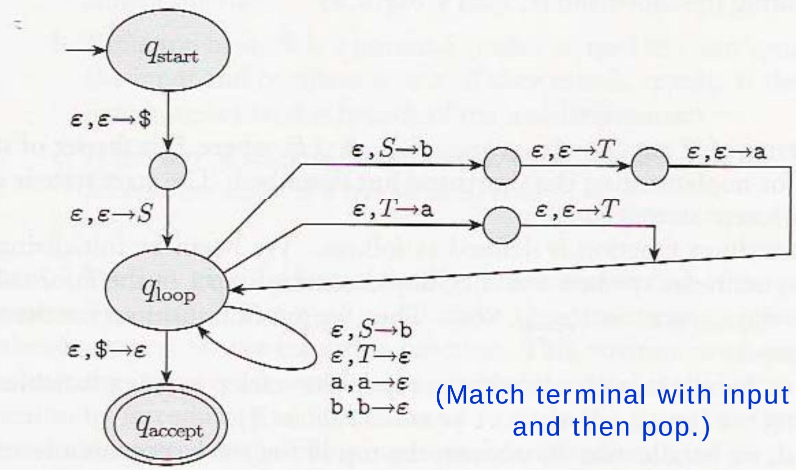
- Proof Idea (again): Produce a pushdown automaton P from the context-free grammar G for the context-free language.
- Example:

EXAMPLE 2.25

We use the procedure developed in Lemma 2.21 to construct a PDA P_1 from the following CFG G.

$$S \to \mathtt{a}T\mathtt{b} \mid \mathtt{b}$$
$$T \to T\mathtt{a} \mid \varepsilon$$

The transition function is shown in the following diagram.



$S \to aTb$

$T \to Ta$

(Match terminal with input and then pop.)

FIGURE 2.26
State diagram of P_1

Additional example: board work

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.27 ($\Leftarrow$)

- Lemma 2.27 ($\Leftarrow$) If a pushdown automaton P recognizes some language, then it is context-free.
  - Proof Idea:
    - Design grammar G that does more:
      - Create variable A_{pq} for each pair of states p and q in P.
        - A_{pq} generates all strings taking P from p with *empty stack* to q with *empty stack* (overkill!)
      - To support this, first modify P so that:
        - It has a single accept state q_{accept}.
        - It empties its stack before accepting.
        - Each transition *either* pushes a symbol onto the stack or pops one off the stack (not simultaneous).
        - How can we implement these 3 features? (example)

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.27 ($\Leftarrow$)

- Lemma 2.27 ($\Leftarrow$) If a pushdown automaton P recognizes some language, then it is context-free.
  - Proof Idea (continued): Design grammar G that does more (continued):
    - Understand how P operates on strings (e.g. string x):
      - First move must be a *push* (why?)
      - Last move must be a *pop* (why?)
      - Intermediate moves: 2 cases
        - Case 1: Symbol popped at end is symbol pushed at beginning. $A_{pq} \rightarrow aA_{rs}b$
        - Case 2: Otherwise, symbol pushed at start is popped somewhere in between. $A_{pq} \rightarrow A_{pr}A_{rq}$

See figures in later slides.

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.27 ($\Leftarrow$)

- Lemma 2.27 ($\Leftarrow$) If a pushdown automaton P recognizes some language, then it is context-free.
  - Recall: $(r,u) \in \delta(q,a,s)$ means when P is in state q, a is next input symbol, and s is symbol on top of stack, P reads a, pops s, pushes u onto stack and goes to state r.

PROOF Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$ and construct G. The variables of G are $\{A_{pq} | \, p, q \in Q\}$. The start variable is $A_{q_0, q_{\text{accept}}}$. Now we describe G's rules.

- For each $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $\delta(p, a, \varepsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ε), put the rule $A_{pq} \to aA_{rs}b$ in G.
- For each $p, q, r \in Q$, put the rule $A_{pq} \to A_{pr}A_{rq}$ in G.
- Finally, for each $p \in Q$, put the rule $A_{pp} \to \varepsilon$ in G.

Continue example...

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.27 ($\Leftarrow$)

- Lemma 2.27 ($\Leftarrow$) If a pushdown automaton P recognizes some language, then it is context-free.

- For each $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $\delta(p, a, \varepsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ε), put the rule $A_{pq} \to aA_{rs}b$ in G.

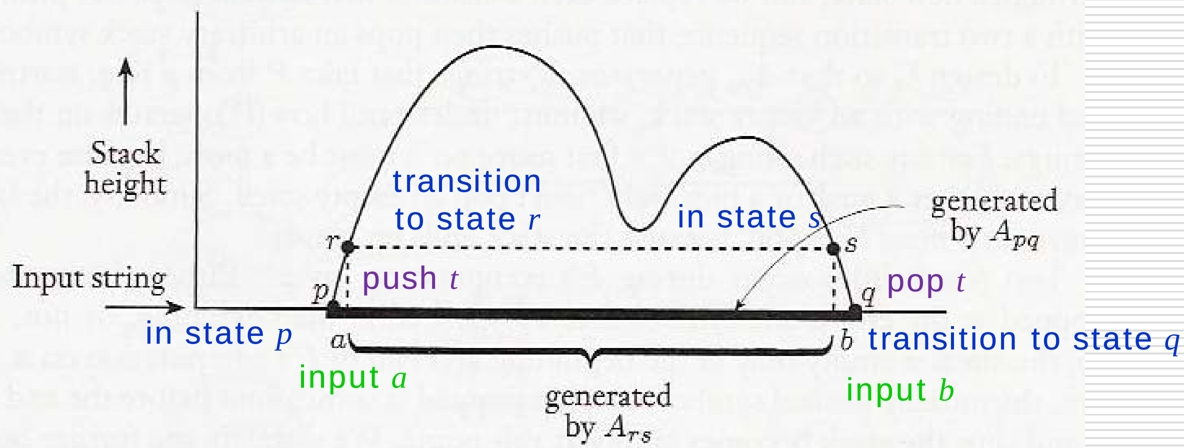


FIGURE 2.29
PDA computation corresponding to the rule $A_{pq} \to aA_{rs}b$

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.27 ($\Leftarrow$)

- Lemma 2.27 ($\Leftarrow$) If a pushdown automaton P recognizes some language, then it is context-free.

• For each $p, q, r \in Q$, put the rule $A_{pq} \to A_{pr}A_{rq}$ in G.

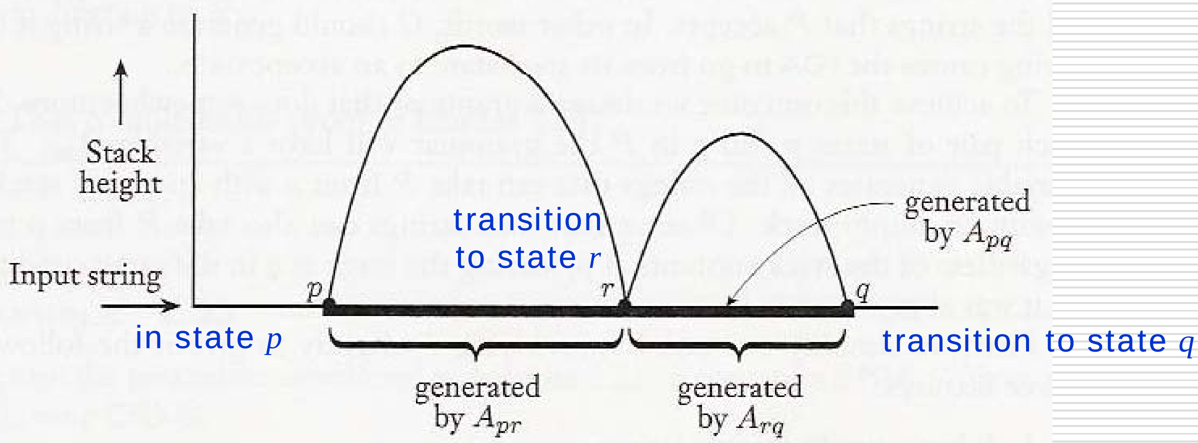


FIGURE 2.28
PDA computation corresponding to the rule $A_{pq} \to A_{pr}A_{rq}$

Source: Sipser Textbook

Equivalence with Context-Free Grammars: Lemma 2.27 ($\Leftarrow$)

- Lemma 2.27 ($\Leftarrow$) If a pushdown automaton P recognizes some language, then it is context-free.
  - Show construction (previous 3 slides) works by proving:
    - A_{pq} generates x iff x can bring P from state p with empty stack to state q with empty stack.
    - $\Rightarrow$ Claim 2.30: If A_{pq} generates x, then x can bring P from state p with empty stack to state q with empty stack.
      - Proof is by induction on number of steps in deriving x from A_{pq}.
      - (see textbook for details)
    - $\Leftarrow$ Claim 2.31: If x can bring P from state p with empty stack to state q with empty stack, then A_{pq} generates x.
      - Proof is by induction on number of steps in computation of P that goes from state p to state q with empty stacks on input x.
      - (see textbook for details)

Source: Sipser Textbook

A Consequence of Lemma 2.27

- Corollary 2.32: Every regular language is context free.
  - Proof Idea:
    - Every regular language is recognized by a finite automaton.
    - Every finite automaton is a pushdown automaton that ignores its stack.
    - Lemma 2.27 (rephrased): Every pushdown automaton can be associated with a context-free grammar.
    - Now apply transitivity.

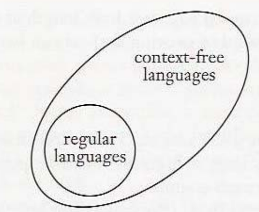
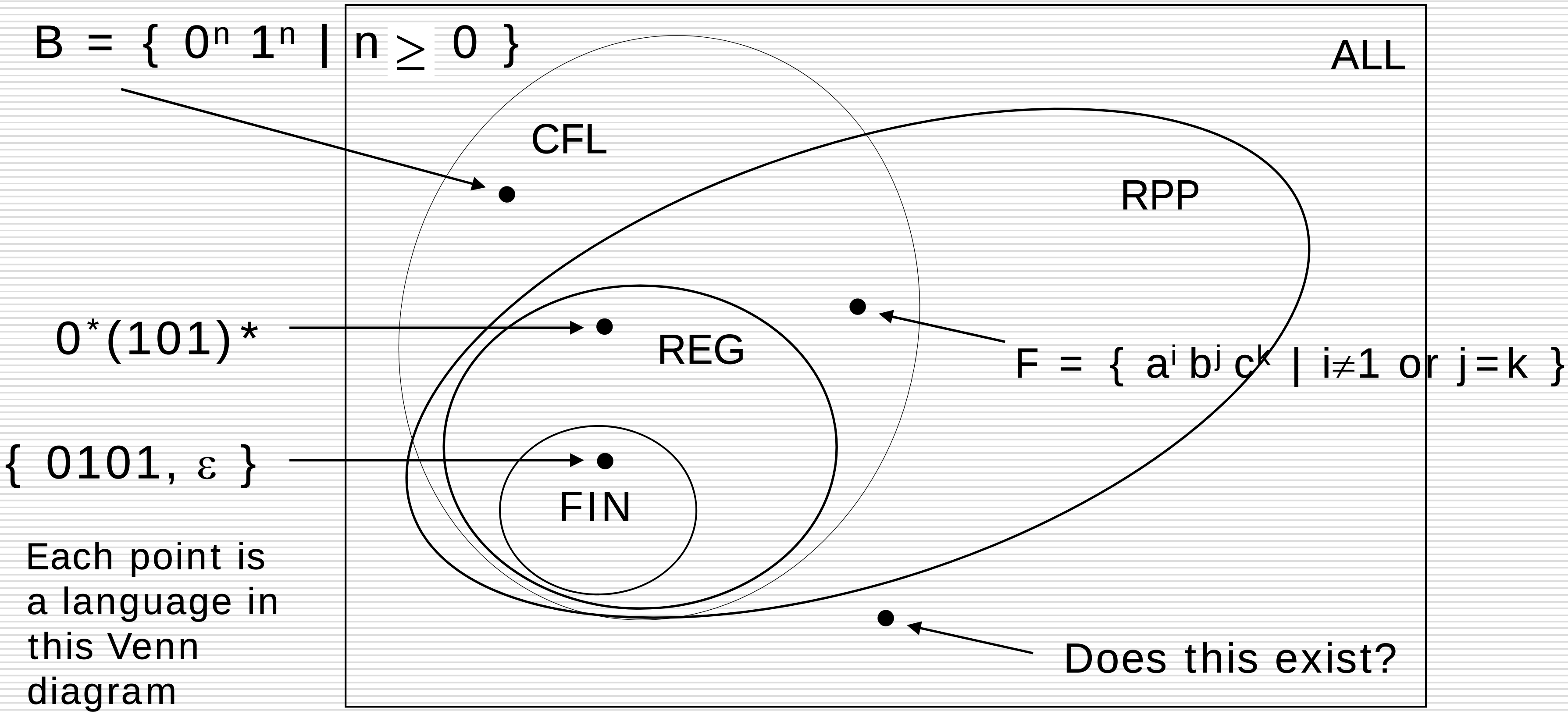


FIGURE 2.33
Relationship of the regular and context-free languages

Source: Sipser Textbook

Picture so far



Each point is a language in this Venn diagram

Source: Dr. David Martin