Theory of Computation 9
Non-Deterministic Membership Testing

Frank Stephan

Department of Computer Science

Department of Mathematics

National University of Singapore

fstephan@comp.nus.edu.sg

Repetition 1

For a language $\mathbf{L}$ and a word $\mathbf{w}$ of length $\mathbf{n}$, one wants to decide whether $\mathbf{w} \in \mathbf{L}$. The following will be shown:

Regular language: Done by a finite automaton, time $\mathbf{O(n)}$.

Linear language: Special case of Cocke, Kasami and Younger's algorithm, time $\mathbf{O(n^2)}$.

Context-free language: Cocke, Kasami and Younger's algorithm, time $\mathbf{O(n^3)}$.

Context-sensitive language: Savitch's algorithm, space $\mathbf{O(n^2)}$, time $\mathbf{O(c^{n^2})}$ for some $\mathbf{c}$.

Today: Non-deterministic Linear Time Algorithms for Context-Free Languages.

Repetition 2

Let (N, Σ, P, S) be in Chomsky Normal Form and $a_1 a_2 \ldots a_n$ be the input word.

1. Initialisation: For all k,

$$E_{k,k} \quad = \quad \{A \in N : A \rightarrow a_k \text{ is a rule}\}.$$

2. Loop: Go through all pairs (i, j) such that they are processed in increasing order of $j - i$ and let

$$\begin{aligned} E_{i,j} \quad = \quad & \{A : \exists \text{ rule } A \rightarrow BC \, \exists k \\ & [i \leq k < j \text{ and } B \in E_{i,k} \text{ and } C \in E_{k+1,j}]\}. \end{aligned}$$

3. Decision: Word is generated by the grammar iff $S \in E_{1,n}$.

Set $E_{i,j}$ contains all non-terminals generating $a_i \ldots a_j$.
Time $O(n^3)$: $O(n^2)$ values $E_{i,j}$ with $O(n)$ choices of k.

Repetition 3

Linear Grammars

Each rules if of the form $\mathbf{A \to u}$ or $\mathbf{A \to vBw}$ with non-terminals $\mathbf{A, B}$ and terminal words $\mathbf{u, v, w}$.

Faster Membership Test

Variant of Cocke Kasami Younger algorithm does it in $\mathbf{O(n^2)}$.

Combination of Context-Free Grammars

A regular combination of grammars $\mathbf{L_1, \ldots, L_n}$ is formed from these by taking unions, intersections, concatenations, set difference, Kleene star and Kleene plus, if needed, repeatedly.

All grammars which are a regular combination of some context-free grammars have an $\mathbf{O(n^3)}$ membership test.

Some regular combinations of linear grammars have an $\mathbf{O(n^2)}$ membership test, for example $\mathbf{L^*}$ for linear $\mathbf{L}$.

Repetition 4

Algorithm 8.16

Context-senstive grammar (N, Σ, P, S), input word w.

Recursive Call: Function $Check(u, v, t)$

```
    Begin If $u = v$ or $u \Rightarrow v$ Then Return$(1)$;
    If $t \leq 1$ and $u \neq v$ and $u \not\Rightarrow v$ Then Return$(0)$;
    Let $t' = t/2$; Let $r' = 0$;
    For all $u' \in (N \cup \Sigma)^*$ with $|u| \leq |u'| \leq |v|$ Do
    Begin If $Check(u, u', t') = 1$ and $Check(u', v, t') = 1$
    Then $r' = 1$ End; Return$(r')$ End.
```

Decision: If $Check(S, w, k^n) = 1$ Then $w \in L$ Else $w \notin L$.

Space Complexity, per call $O(n)$, in total $O(n^2)$;

Value of t: $k^n/2^h$ in depth h of recursion $(k = |\Sigma| + |N| + 1)$;

Number of nested calls: $O(\log(k^n)) = O(\log(k) \cdot n)$.

Runtime: $O(c^{n^2})$ for any $c > (2k)^{\log(k)}$.

Repetition 5

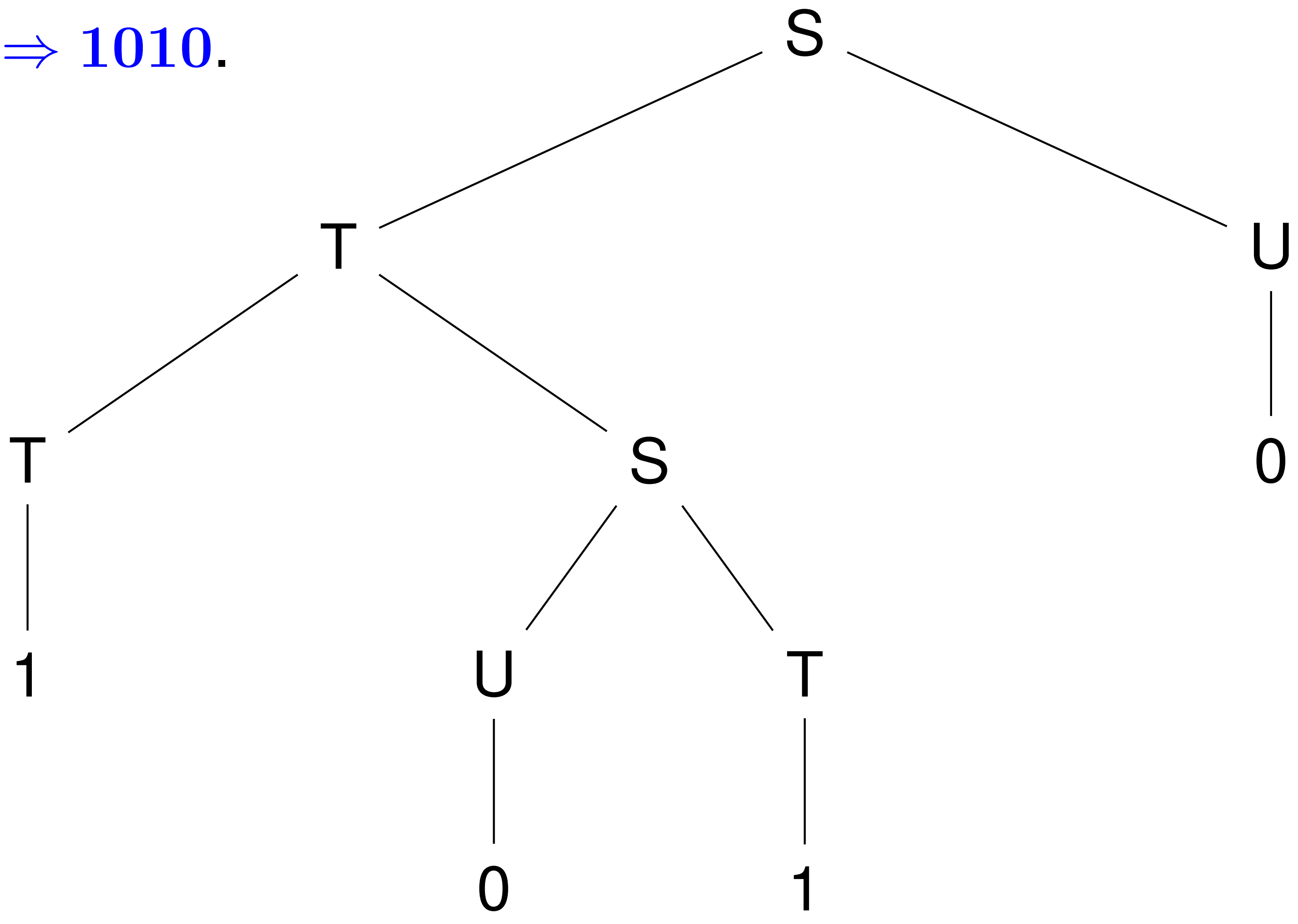
Definition [Dahlhaus and Warmuth 1986]
A grammar (N, Σ, P, S) is growing context-sensitive iff $|l| < |r|$ for all rules $l \rightarrow r$ in the grammar.

Theorem [Dahlhaus and Warmuth 1986]
Given a growing context-senstive grammar there is a polynomial time algorithm which decides membership of the language generated by this growing grammar.

The language $\{0^n 1^{2^n} : n > 0\}$ has a growing context-sensitive grammar but not a context-free one.

Leftmost Derivation

The grammar $(\{S, T, U\}, \{0, 1\}, \{S \to SS|TU|UT,$
$U \to 0|US|SU,\ T \to 1|TS|ST\}, S)$ has on 1010 the left-most derivation $S \Rightarrow TU \Rightarrow TSU \Rightarrow 1SU \Rightarrow 1UTU \Rightarrow 10TU \Rightarrow 101U \Rightarrow 1010$.



Idea for Algorithm

Chomsky Normal Form

Every rule either produces an additional non-terminal or transforms a non-terminal into terminal.
Left-most derivation for word of length n needs $2n - 1$ steps.

Method to Check Derivation

Start with the symbol S on the stack and keep the non-terminals of the left-most derivation on the stack while the terminals, whenever generated, are compared with the input word to be checked.

Each cycle, pull top symbol of stack for one step.
If new non-terminals are made, push them back to stack.
If a terminal is made, compare it with the next input symbol.
If they agree then continue into next cycle else reject.

Accept when the whole input is compared and stack empty.

Formal Definition 9.2

Pushdown Automaton $(\mathbf{Q}, \mathbf{\Sigma}, \mathbf{N}, \delta, \mathbf{s}, \mathbf{S}, \mathbf{F})$
$\mathbf{Q}$ are states with start state $\mathbf{s}$ and accepting states $\mathbf{F}$;
$\mathbf{N}$ are stack symbols with start symbol $\mathbf{S}$;
$\mathbf{\Sigma}$ is terminal alphabet;
δ gives choices what to do in cycle; δ maps $(\mathbf{state}, \mathbf{current\ input}, \mathbf{top\ stack\ symbol})$ to choices for $(\mathbf{new\ state}, \mathbf{new\ top\ of\ stack})$ where the input is from $\{\varepsilon\} \cup \mathbf{\Sigma}$ and the new top of stack is from $\mathbf{N}^*$.

In each cycle, the pushdown automaton follows an option of δ what it can do.

A run is successful iff all input gets processed and an accepting state gets reached (acceptance by state); acceptance by empty stack requires in addition that the stack is empty.

Example 9.3 of PDA

Pushdown Automaton for Grammar from Page 7.

- States $\mathbf{Q} = \{\mathbf{s}\}$ and $\mathbf{F} = \{\mathbf{s}\}$;
- Terminals $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$;
- Stack Symbols $\mathbf{N} = \{\mathbf{S}, \mathbf{T}, \mathbf{U}\}$ with start symbol $\mathbf{S}$;
- $\delta(\mathbf{s}, \varepsilon, \mathbf{S}) = \{(\mathbf{s}, \mathbf{SS}), (\mathbf{s}, \mathbf{TU}), (\mathbf{s}, \mathbf{UT})\}$;
  $\delta(\mathbf{s}, \varepsilon, \mathbf{T}) = \{(\mathbf{s}, \mathbf{TS}), (\mathbf{s}, \mathbf{ST})\}$;
  $\delta(\mathbf{s}, \varepsilon, \mathbf{U}) = \{(\mathbf{s}, \mathbf{US}), (\mathbf{s}, \mathbf{SU})\}$;
  $\delta(\mathbf{s}, \mathbf{0}, \mathbf{U}) = \{(\mathbf{s}, \varepsilon)\}$;
  $\delta(\mathbf{s}, \mathbf{1}, \mathbf{T}) = \{(\mathbf{s}, \varepsilon)\}$;
  $\delta(\mathbf{s}, \mathbf{v}, \mathbf{A}) = \emptyset$ for all other choices of $(\mathbf{v}, \mathbf{A})$.

Here a sample processing of the word $\mathbf{0011}$:

current input	start	ε	$\mathbf{0}$	ε	ε	$\mathbf{0}$	$\mathbf{1}$	$\mathbf{1}$
new stack	$\mathbf{S}$	$\mathbf{UT}$	$\mathbf{T}$	$\mathbf{ST}$	$\mathbf{UTT}$	$\mathbf{TT}$	$\mathbf{T}$	ε

Algorithm 9.4 CTF $\rightarrow$ PDA by ES

Let $(\mathbf{N}, \mathbf{\Sigma}, \mathbf{P}, \mathbf{S})$ be a grammar in Chomsky Normal Form generating a language $\mathbf{L}$. Then one can construct a pushdown automaton recognising the same language $\mathbf{L}$ by empty stack as follows:

- $\mathbf{Q} = \{\mathbf{s}\}$ and $\mathbf{F} = \{\mathbf{s}\}$;
- $\mathbf{\Sigma}$ and $\mathbf{N}$ are taken over from the grammar; furthermore, $\mathbf{S}$ is again the start symbol;
- For every non-terminal $\mathbf{A} \in \mathbf{N}$, one defines that $\delta(\mathbf{s}, \varepsilon, \mathbf{A}) = \{(\mathbf{s}, \mathbf{BC}) : \mathbf{A} \rightarrow \mathbf{BC}$ is in $\mathbf{P}\} \cup \{(\mathbf{s}, \varepsilon) : \mathbf{S} \rightarrow \varepsilon$ is in $\mathbf{P}$ and $\mathbf{A} = \mathbf{S}\}$,
  for $\mathbf{a} \in \mathbf{\Sigma}$, if $\mathbf{A} \rightarrow \mathbf{a}$ is in $\mathbf{P}$ then $\delta(\mathbf{s}, \mathbf{a}, \mathbf{A}) = \{(\mathbf{s}, \varepsilon)\}$ else $\delta(\mathbf{s}, \mathbf{a}, \mathbf{A}) = \emptyset$.

Every context-free language can be recognised by a pushdown automaton with empty stack acceptance condition.

Verification I

Let $\mathbf{v} \in \mathbf{\Sigma}^*$, $\mathbf{w} \in \mathbf{N}^*$ and $\mathbf{vw} \neq \varepsilon$.

One proves by induction the following:
The grammar $(\mathbf{N}, \mathbf{\Sigma}, \mathbf{P}, \mathbf{S})$ can derive $\mathbf{vw}$ in $\mathbf{n}$ steps iff the pushdown automaton can go from $\mathbf{S}$ in the stack to $\mathbf{w}$ in the stack in $\mathbf{n}$ steps while processing input $\mathbf{v}$. Clear for $\mathbf{n} = \mathbf{0}$.

Case $\mathbf{w} = \mathbf{vBC\tilde{w}}$, rule $\mathbf{A} \rightarrow \mathbf{BC}$ in grammar and grammar derives $\mathbf{S} \Rightarrow^* \mathbf{vA\tilde{w}} \Rightarrow \mathbf{vBC\tilde{w}}$ in $\mathbf{n} + \mathbf{1}$ steps. By hypothesis PDA has after $\mathbf{n}$ steps processed $\mathbf{v}$ and stack $\mathbf{A\tilde{w}}$. One more step gets $\mathbf{BC\tilde{w}}$ in stack.

Case $\mathbf{v} = \mathbf{\tilde{v}a}$ and grammar derives $\mathbf{S} \Rightarrow^* \mathbf{\tilde{v}Aw} \Rightarrow \mathbf{\tilde{v}aw}$ in $\mathbf{n} + \mathbf{1}$ steps. Then PDA processes input $\mathbf{\tilde{v}}$ in $\mathbf{n}$ steps and has stack $\mathbf{Aw}$. It can in next step read input $\mathbf{a}$ and consume stack symbol $\mathbf{A}$ in order to get $\mathbf{v} = \mathbf{\tilde{v}a}$ processed and stack to be $\mathbf{w}$.

Verification II

Other direction: Case PDA processes input $\mathbf{v}$ with stack $\mathbf{A\tilde{w}}$ in $\mathbf{n}$ steps and then replaces $\mathbf{A}$ by $\mathbf{BC}$. The grammar can derive $\mathbf{vA\tilde{w}}$ in $\mathbf{n}$ steps by induction hypothesis and then apply rule $\mathbf{A \to BC}$ to get $\mathbf{vBC\tilde{w}}$.

Case PDA processes input $\mathbf{\tilde{v}}$ with stack $\mathbf{Aw}$ in $\mathbf{n}$ steps and then reads input $\mathbf{a}$ and removes $\mathbf{A}$ in one step. By induction hypothesis, grammar reaches $\mathbf{\tilde{v}Aw}$ in $\mathbf{n}$ steps; it has rule $\mathbf{A \to a}$ to reach $\mathbf{\tilde{v}aw}$ in one further step.

Summary

The grammar can derive a nonempty word in $\mathbf{vw \in \Sigma^* \cdot N^*}$ iff the pushdown automaton can while processing input $\mathbf{v}$ reach a situation where $\mathbf{w}$ is in the stack. In particular, the language of words in $\mathbf{\Sigma^+}$ generated by the grammar is the same as the language recognised by the PDA.

Exercises

Construct pushdown automata recognising by empty stack for the below languages.

Exercise 9.5

The language is $\{0^n 1^m 2^k : n + m + 1 = k\}$.

Exercise 9.6

The language is $\{0^n 1^m 2^k : n + m < k\}$.

Exercise 9.7

The language is $\{0^n 1^m 2^k : n \neq m$ and $k > 0\}$.

Algorithm 9.8 CTF $\rightarrow$ PDA by STATE

Assume that $(\mathbf{N}, \mathbf{\Sigma}, \mathbf{P}, \mathbf{S})$ is a context-free language in Chomsky Normal Form. The pushdown automaton $(\{\mathbf{s}, \mathbf{t}\}, \mathbf{\Sigma}, \mathbf{N} \cup \mathbf{N}', \delta, \mathbf{s}, \mathbf{S}', \{\mathbf{t}\})$ accepts by state where $\mathbf{N}' = \{\mathbf{A}' : \mathbf{A} \in \mathbf{N}\}$ is a “primed copy” of $\mathbf{N}$ and for every non-terminal $\mathbf{A} \in \mathbf{N}$ and the corresponding $\mathbf{A}' \in \mathbf{N}'$, δ is defined as follows:

> $\delta(\mathbf{s}, \varepsilon, \mathbf{A}) = \{(\mathbf{s}, \mathbf{BC}) : \mathbf{A} \rightarrow \mathbf{BC}$ is a rule in $\mathbf{P}\}$;
> $\delta(\mathbf{s}, \varepsilon, \mathbf{A}') = \{(\mathbf{s}, \mathbf{BC}') : \mathbf{A} \rightarrow \mathbf{BC}$ is a rule in $\mathbf{P}\}$
> $\cup \{(\mathbf{t}, \varepsilon) : \mathbf{A}' = \mathbf{S}'$ and $\mathbf{S} \rightarrow \varepsilon$ is a rule in $\mathbf{P}\}$;
> for all terminals $\mathbf{a}$, if the rule $\mathbf{A} \rightarrow \mathbf{a}$ is in $\mathbf{P}$
> then $\delta(\mathbf{s}, \mathbf{a}, \mathbf{A}) = \{(\mathbf{s}, \varepsilon)\}$ and $\delta(\mathbf{s}, \mathbf{a}, \mathbf{A}') = \{(\mathbf{t}, \varepsilon)\}$
> else $\delta(\mathbf{s}, \mathbf{a}, \mathbf{A}) = \emptyset$ and $\delta(\mathbf{s}, \mathbf{a}, \mathbf{A}') = \emptyset$;
> $\delta(\mathbf{t}, \mathbf{v}, \mathbf{A}), \delta(\mathbf{t}, \mathbf{v}, \mathbf{A}')$ are $\emptyset$ for all $\mathbf{v}$.

Ideas of Verification

Last symbol on stack is of type $\mathbf{A'}$, all others are of type $\mathbf{A}$.

Pushdown automaton goes into the accepting state $\mathbf{t}$ only when processing some $\mathbf{A'}$ and then stops doing anything; this simulates acceptance by empty stack without having it as a rule of the pushdown automaton.

One shows the following two statements:

$\mathbf{S} \Rightarrow^* \mathbf{vwA}$ with $\mathbf{v} \in \Sigma^*$, $\mathbf{w} \in \mathbf{N}^*$ and $\mathbf{A} \in \mathbf{N}$ iff the pushdown automaton can, on input $\mathbf{v}$ reach the stack content $\mathbf{wA'}$ and is in state $\mathbf{s}$.

The pushdown automaton can process input $\mathbf{v}$ and reach empty stack iff it can process $\mathbf{v}$ and reach the state $\mathbf{t}$.

Exercises

Construct pushdown automata accepting by state for the following languages.

Exercise 9.9
Construct a pushdown automaton accepting by state for the language $\{0^n1^m2^k : n+m > k\}$.

Exercise 9.10
Construct a pushdown automaton accepting by state for the language $\{0^n1^m2^k : n \neq k$ or $m \neq k\}$.

Exercise 9.11
Construct a pushdown automaton accepting by state for the language $\{w \in \{0,1\}^* : w$ is not a palindrome$\}$.

PDA by STATE $\rightarrow$ PDA by ES

Theorem 9.12

If $\mathbf{L}$ can be recognised by a pushdown automaton accepting by final state then it can also be recognised by a pushdown automaton accepting by empty stack.

Construction

Given a pushdown automaton $(\mathbf{Q}, \mathbf{\Sigma}, \mathbf{N}, \delta, \mathbf{s}, \mathbf{S}, \mathbf{F})$ for $\mathbf{L}$, one constructs a new automaton $(\mathbf{Q} \cup \{\mathbf{t}\}, \mathbf{N}, \delta', \mathbf{s}, \mathbf{S}, \mathbf{F} \cup \{\mathbf{t}\})$ as follows (where $\mathbf{t}$ is a new state outside $\mathbf{Q}$):

- For all $\mathbf{q} \in \mathbf{Q}$, $\mathbf{A} \in \mathbf{N}$ and $\mathbf{v}$, $\delta'(\mathbf{q}, \mathbf{v}, \mathbf{A}) = \delta(\mathbf{q}, \mathbf{v}, \mathbf{A}) \cup \{(\mathbf{t}, \varepsilon) : \mathbf{v} = \varepsilon$ and $\mathbf{q} \in \mathbf{F}\}$;
- For all $\mathbf{A} \in \mathbf{N}$ and $\mathbf{v} \neq \varepsilon$, $\delta'(\mathbf{t}, \varepsilon, \mathbf{A}) = \{(\mathbf{t}, \varepsilon)\}$ and $\delta'(\mathbf{t}, \mathbf{v}, \mathbf{A}) = \emptyset$.

Idea: Simulate old PDA, but from accepting state can transit to $\mathbf{t}$ and remove all remaining stack symbols.

Algorithm 9.13 PDA by ES to CTF

Given a pushdown automaton $(\mathbf{Q}, \mathbf{\Sigma}, \mathbf{N}, \delta, \mathbf{s}, \mathbf{S}, \mathbf{F})$ for $\mathbf{L}$, let the grammar $((\mathbf{Q} \times \mathbf{N} \times \mathbf{Q}) \cup \{\mathbf{S}'\}, \mathbf{\Sigma}, \mathbf{P}, \mathbf{S}')$ be defined by putting the following rules into $\mathbf{P}$:

- For all $\mathbf{p} \in \mathbf{F}$, put all rules $\mathbf{S}' \to (\mathbf{s}, \mathbf{S}, \mathbf{p})$ into $\mathbf{P}$;
- For all $\mathbf{q}, \mathbf{r} \in \mathbf{Q}$, $\mathbf{A} \in \mathbf{N}$, $\mathbf{v} \in \mathbf{\Sigma}^*$, $(\mathbf{p_1}, \mathbf{w}) \in \delta(\mathbf{q}, \mathbf{v}, \mathbf{A})$ with $\mathbf{w} = \mathbf{B_1 B_2 \ldots B_n}$ and $\mathbf{n} > \mathbf{0}$, $\mathbf{p_2}, \ldots, \mathbf{p_n} \in \mathbf{Q}$, put the rule

  $$(\mathbf{q}, \mathbf{A}, \mathbf{r}) \to \mathbf{v}(\mathbf{p_1}, \mathbf{B_1}, \mathbf{p_2})(\mathbf{p_2}, \mathbf{B_2}, \mathbf{p_3}) \ldots (\mathbf{p_n}, \mathbf{B_n}, \mathbf{r})$$

  into $\mathbf{P}$;
- For each $\mathbf{q} \in \mathbf{Q}$, $\mathbf{A} \in \mathbf{N}$, $\mathbf{v} \in \mathbf{\Sigma}^*$ and $(\mathbf{p}, \varepsilon) \in \delta(\mathbf{q}, \mathbf{v}, \mathbf{A})$, put the rule $(\mathbf{q}, \mathbf{A}, \mathbf{p}) \to \mathbf{v}$ into $\mathbf{P}$.

Ideas of Verification

The main construction guideline is that

> $(\mathbf{q}, \mathbf{A}, \mathbf{r}) \Rightarrow^* \mathbf{v}$ iff the pushdown automaton can process input $\mathbf{v}$ while using up the non-terminal $\mathbf{A}$ without touching the stack behind $\mathbf{A}$ and transiting from state $\mathbf{q}$ to state $\mathbf{r}$.

The rule

$$(\mathbf{q}, \mathbf{A}, \mathbf{r}) \rightarrow \mathbf{v}(\mathbf{p_1}, \mathbf{B_1}, \mathbf{p_2})(\mathbf{p_2}, \mathbf{B_2}, \mathbf{p_3}) \ldots (\mathbf{p_n}, \mathbf{B_n}, \mathbf{r})$$

considers any combination of intermediate states $\mathbf{p_2}, \mathbf{p_3}, \ldots, \mathbf{p_n}$ which do not lead to a terminating derivation if guessed wrongly, but lead to a correct termination if guessed correctly. $\mathbf{p_1}$ is computed by the transition function. The rule $(\mathbf{q}, \mathbf{A}, \mathbf{r}) \rightarrow \mathbf{v}$ is only put into the grammar if the δ updates from $\mathbf{q}$ to $\mathbf{r}$ by using up $\mathbf{A}$ and reading $\mathbf{v}$.

Example: Pushdown Automaton

Example 9.14
The following pushdown automaton accepts by empty stack a language.

- $\mathbf{Q} = \{\mathbf{s}, \mathbf{t}\}$, $\mathbf{F} = \{\mathbf{t}\}$, start state is $\mathbf{s}$;
- $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$;
- $\mathbf{N} = \{\mathbf{S}, \mathbf{U}, \mathbf{T}\}$, start symbol is $\mathbf{S}$;
- $\delta(\mathbf{s}, \mathbf{0}, \mathbf{S}) = \{(\mathbf{s}, \mathbf{SU}), (\mathbf{t}, \mathbf{U}), (\mathbf{t}, \varepsilon)\}$;
  $\delta(\mathbf{s}, \mathbf{1}, \mathbf{S}) = \{(\mathbf{s}, \mathbf{ST}), (\mathbf{t}, \mathbf{T}), (\mathbf{t}, \varepsilon)\}$;
  $\delta(\mathbf{t}, \mathbf{0}, \mathbf{U}) = \{(\mathbf{t}, \varepsilon)\}$; $\delta(\mathbf{t}, \mathbf{1}, \mathbf{U}) = \emptyset$;
  $\delta(\mathbf{t}, \mathbf{1}, \mathbf{T}) = \{(\mathbf{t}, \varepsilon)\}$; $\delta(\mathbf{t}, \mathbf{0}, \mathbf{T}) = \emptyset$;
  $\delta(\mathbf{q}, \varepsilon, \mathbf{A}) = \emptyset$ for all $\mathbf{q} \in \mathbf{Q}$ and $\mathbf{A} \in \mathbf{N}$.

Grammars for PDA

The corresponding pushdown automaton has the states $\{\mathbf{s},\mathbf{t}\} \times \{\mathbf{S},\mathbf{U},\mathbf{T}\} \times \{\mathbf{s},\mathbf{t}\} \cup \{\mathbf{S}'\}$; one can omit most states and get the following grammar:

- Non-terminals: $\mathbf{S}', (\mathbf{s},\mathbf{S},\mathbf{t}), (\mathbf{t},\mathbf{U},\mathbf{t}), (\mathbf{t},\mathbf{T},\mathbf{t})$;
- Terminals: $\mathbf{0},\mathbf{1}$;
- $\mathbf{S}' \to (\mathbf{s},\mathbf{S},\mathbf{t})$;
  $(\mathbf{s},\mathbf{S},\mathbf{t}) \to \mathbf{0}(\mathbf{s},\mathbf{S},\mathbf{t})(\mathbf{t},\mathbf{U},\mathbf{t})|\mathbf{0}(\mathbf{t},\mathbf{U},\mathbf{t})|\mathbf{0}$;
  $(\mathbf{s},\mathbf{S},\mathbf{t}) \to \mathbf{1}(\mathbf{s},\mathbf{S},\mathbf{t})(\mathbf{t},\mathbf{T},\mathbf{t})|\mathbf{1}(\mathbf{t},\mathbf{T},\mathbf{t})|\mathbf{1}$;
  $(\mathbf{t},\mathbf{U},\mathbf{t}) \to \mathbf{0}$;
  $(\mathbf{t},\mathbf{T},\mathbf{t}) \to \mathbf{1}$;
- Start symbol $\mathbf{S}'$.

Improved grammar:
$(\{\mathbf{S}\},\{\mathbf{0},\mathbf{1}\},\{\mathbf{S} \to \mathbf{0S0}|\mathbf{1S1}|\mathbf{00}|\mathbf{11}|\mathbf{0}|\mathbf{1}\},\mathbf{S})$.

Algorithm 9.15 GNF to PDA

Greibach Normal Form grammar $(\mathbf{N}, \mathbf{\Sigma}, \mathbf{P}, \mathbf{S})$ to PDA $(\mathbf{Q}, \mathbf{\Sigma}, \mathbf{N}', \delta, \mathbf{s}, \mathbf{S}, \mathbf{F})$ by accepting state:

- The set of states is $\{\mathbf{s}, \mathbf{t}, \mathbf{u}\}$; start state $\mathbf{s}$;
  if ε is in the language then $\mathbf{F} = \{\mathbf{s}, \mathbf{u}\}$ else $\mathbf{F} = \{\mathbf{u}\}$;
- Let $\mathbf{N}' = \{\mathbf{A}, \mathbf{A}' : \mathbf{A} \in \mathbf{N}\}$ and $\mathbf{S}'$ be the start symbol;
- The terminal alphabet is $\mathbf{\Sigma}$ as for the grammar;
- For all symbols $\mathbf{a} \in \mathbf{\Sigma}$ and $\mathbf{A} \in \mathbf{N}$,
  $\delta(\mathbf{s}, \mathbf{a}, \mathbf{S}') = \{(\mathbf{t}, \mathbf{B_1 B_2 \ldots B_n'}) : \mathbf{S} \rightarrow \mathbf{a B_1 B_2 \ldots B_n}$ is a rule in $\mathbf{P}$ with $\mathbf{n} > \mathbf{0}\} \cup \{(\mathbf{u}, \varepsilon) : \mathbf{S} \rightarrow \mathbf{a}$ is a rule in $\mathbf{P}\}$;
  $\delta(\mathbf{t}, \mathbf{a}, \mathbf{A}) = \{(\mathbf{t}, \mathbf{B_1 B_2 \ldots B_n}) : \mathbf{A} \rightarrow \mathbf{a B_1 B_2 \ldots B_n}$ is a rule in $\mathbf{P}$ with $\mathbf{n} \geq \mathbf{0}\}$;
  $\delta(\mathbf{t}, \mathbf{a}, \mathbf{A}') = \{(\mathbf{t}, \mathbf{B_1 B_2 \ldots B_n'}) : \mathbf{A} \rightarrow \mathbf{a B_1 B_2 \ldots B_n}$ is a rule in $\mathbf{P}$ with $\mathbf{n} > \mathbf{0}\} \cup \{(\mathbf{u}, \varepsilon) : \mathbf{A} \rightarrow \mathbf{a}$ is a rule in $\mathbf{P}\}$;
  $\delta(\mathbf{q}, \mathbf{v}, \mathbf{A}), \delta(\mathbf{q}, \mathbf{v}, \mathbf{A}')$ are $\emptyset$ for all states $\mathbf{q}$, $\mathbf{A} \in \mathbf{N}$ and $\mathbf{v}$ where not defined before.

Greibach Normal Form

The pushdown automaton from the previous slide was complicated in order to accomodate ε in the case that it is in the language. If it is not, an easier way is to do it by acceptance by empty stack.

Exercise 9.16
Given a grammar $(\mathbf{N}, \mathbf{\Sigma}, \mathbf{P}, \mathbf{S})$ in Greibach Normal Form for a language $\mathbf{L}$ not containing ε, explain how to define δ for a pushdown automaton $(\{\mathbf{s}\}, \mathbf{\Sigma}, \mathbf{N}, \delta, \mathbf{s}, \mathbf{S}, \{\mathbf{s}\})$ accepting by empty stack the same language $\mathbf{L}$. This pushdown automaton should process one input symbol in every step. Explain the key ideas of the construction and its verification.

Digit Sum at Average 1

Example 9.17: Deterministic PDA

- $\mathbf{Q} = \{\mathbf{s}\}$; $\mathbf{F} = \{\mathbf{s}\}$; start state $\mathbf{s}$;
- $\mathbf{N} = \{\mathbf{S}\}$; start symbol $\mathbf{S}$;
- $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\}$;
- $\delta(\mathbf{s}, \mathbf{0}, \mathbf{S}) = \{(\mathbf{s}, \varepsilon)\}$;
  $\delta(\mathbf{s}, \mathbf{1}, \mathbf{S}) = \{(\mathbf{s}, \mathbf{S})\}$;
  $\delta(\mathbf{s}, \mathbf{2}, \mathbf{S}) = \{(\mathbf{s}, \mathbf{SS})\}$;
  $\delta(\mathbf{s}, \mathbf{3}, \mathbf{S}) = \{(\mathbf{s}, \mathbf{SSS})\}$;
  $\delta(\mathbf{s}, \varepsilon, \mathbf{S}) = \emptyset$;
- Acceptance mode is by empty stack.

The PDA recognises $\{\mathbf{w} : \mathbf{digitsum}(\mathbf{w}) < |\mathbf{w}|$ and all proper prefixes $\mathbf{v}$ of $\mathbf{w}$ satisfy $\mathbf{digitsum}(\mathbf{v}) \geq |\mathbf{v}|\}$ deterministically.

Deterministic Context-Free

Exercise 9.18
Make context-free grammars in Chomsky Normal Form and Greibach Normal Form for the language of the PDA from Example 9.17.

Definition 9.19
A deterministic pushdown automaton is given as $(\mathbf{Q}, \mathbf{\Sigma}, \mathbf{N}, \delta, \mathbf{s}, \mathbf{S}, \mathbf{F})$ and has the acceptance mode by state with the additional constraint that for every $\mathbf{A} \in \mathbf{N}$ and every $\mathbf{a} \in \mathbf{\Sigma}$ and every $\mathbf{q} \in \mathbf{Q}$, only one of the sets $\delta(\mathbf{q}, \varepsilon, \mathbf{A}), \delta(\mathbf{q}, \mathbf{a}, \mathbf{A})$ can be non-empty and the non-empty one contains exactly one pair $(\mathbf{p}, \mathbf{w})$.

The languages recognised by a deterministic pushdown automaton are called deterministic context-free languages.

Closure Under Complement

Given a deterministic PDA $(\mathbf{Q}, \mathbf{\Sigma}, \mathbf{N}, \delta, \mathbf{s}, \mathbf{S}, \mathbf{F})$ with acceptance by state, construct the new PDA as follows:

- $\mathbf{Q}' = \mathbf{Q} \cup \{\mathbf{t}, \mathbf{u}\}$ for new state $\mathbf{t}, \mathbf{u}$; $\mathbf{F}' = \{\mathbf{u}\} \cup \mathbf{Q} - \mathbf{F}$; start state $\mathbf{t}$;
- $\mathbf{\Sigma}$ is same, new start symbol $\mathbf{S}'$; $\mathbf{N}' = \mathbf{N} \cup \{\mathbf{S}'\}$;
- The new transition function δ' is as follows, where $\mathbf{v} \in \mathbf{\Sigma} \cup \{\varepsilon\}$, $\mathbf{a} \in \mathbf{\Sigma}$, $\mathbf{q} \in \mathbf{Q}$, $\mathbf{A} \in \mathbf{N}$:
  1. $\delta'(\mathbf{t}, \varepsilon, \mathbf{S}') = \{(\mathbf{s}, \mathbf{SS}')\}$;
  2. if $\delta(\mathbf{q}, \mathbf{v}, \mathbf{A}) \neq \emptyset$ then $\delta'(\mathbf{q}, \mathbf{v}, \mathbf{A}) = \delta(\mathbf{q}, \mathbf{v}, \mathbf{A})$;
  3. if $\delta(\mathbf{q}, \mathbf{a}, \mathbf{A})$ and $\delta(\mathbf{q}, \varepsilon, \mathbf{A})$ are both $\emptyset$ then $\delta'(\mathbf{q}, \mathbf{a}, \mathbf{A}) = (\mathbf{u}, \mathbf{S}')$;
  4. $\delta'(\mathbf{q}, \mathbf{a}, \mathbf{S}') = \{(\mathbf{u}, \mathbf{S}')\}$;
  5. $\delta'(\mathbf{u}, \mathbf{a}, \mathbf{S}') = \{(\mathbf{u}, \mathbf{S}')\}$;
  6. δ' is $\emptyset$ everywhere else.

Verification Ideas I

- It starts with state $\mathbf{t}$ and symbol $\mathbf{S'}$ and pushes $\mathbf{SS'}$ onto the stack before simulating the old automaton by instruction of type 1;
- It then simulates the old automaton using instructions of type 2 and it accepts iff the old automaton rejects;
- When the old automaton gets stuck by a missing instruction then the new automaton pushes $\mathbf{S'}$ and goes to state $\mathbf{u}$ by instruction of type 3;
- When the old automaton gets stuck by empty stack then this is indicated by $\mathbf{S'}$ being the symbol to be used and the new automaton pushes $\mathbf{S'}$ back onto the stack and goes to state $\mathbf{u}$ by instruction of type 4;

Verification Ideas II

- Once the automaton reaches state $\mathbf{u}$ and has $\mathbf{S}'$ on the top of the stack, it stays in this situation forever and accepts all subsequent inputs by instructions of type 5;
- The instruction set is completed by defining that δ' takes $\emptyset$ in the remaining cases in order to remain deterministic and to avoid choices in the transitions.

New Automaton gets never stuck. Doing complementation twice gives PDA for old language which never gets stuck.

Every deterministic context-free language is recognised by a deterministic pushdown automaton $(\mathbf{Q}, \mathbf{\Sigma}, \mathbf{N}, \delta, \mathbf{s}, \mathbf{S}, \mathbf{F})$ which has the acceptance mode by state with the additional constraint that for every $\mathbf{A} \in \mathbf{N}$ and every $\mathbf{a} \in \mathbf{\Sigma}$ and every $\mathbf{q} \in \mathbf{Q}$ there is exactly one pair $(\mathbf{p}, \mathbf{w}) \in \delta(\mathbf{q}, \varepsilon, \mathbf{A}) \cup \delta(\mathbf{q}, \mathbf{a}, \mathbf{A})$; this pair is also only in one of these two sets.

Combining with Regular Languages

Proposition 9.21
If $\mathbf{L}$ is recognised by a deterministic pushdown automaton $(\mathbf{Q}, \mathbf{\Sigma}, \mathbf{N}, \delta, \mathbf{s}, \mathbf{S}, \mathbf{F})$ which never gets stuck and $\mathbf{H}$ is recognised by a complete deterministic finite automaton $(\mathbf{Q}', \mathbf{\Sigma}, \delta', \mathbf{s}', \mathbf{F}')$ then $\mathbf{L} \cap \mathbf{H}$ is recognised by the deterministic pushdown automaton

$$(\mathbf{Q} \times \mathbf{Q}', \mathbf{\Sigma}, \mathbf{N}', \delta \times \delta', (\mathbf{s}, \mathbf{s}'), \mathbf{S}, \mathbf{F} \times \mathbf{F}')$$

and $\mathbf{L} \cup \mathbf{H}$ is recognised by the deterministic pushdown automaton

$$(\mathbf{Q} \times \mathbf{Q}', \mathbf{\Sigma}, \mathbf{N}', \delta \times \delta', (\mathbf{s}, \mathbf{s}'), \mathbf{S}, \mathbf{Q} \times \mathbf{F}' \cup \mathbf{F} \times \mathbf{Q}')$$

where $(\delta \times \delta')((\mathbf{q}, \mathbf{q}'), \mathbf{a}, \mathbf{A}) = \{(\mathbf{p}, \mathbf{p}'), \mathbf{w}) : (\mathbf{p}, \mathbf{w}) \in \delta(\mathbf{q}, \mathbf{a}, \mathbf{A})$ and $\mathbf{p}' = \delta'(\mathbf{q}', \mathbf{a})\}$ and $(\delta \times \delta')((\mathbf{q}, \mathbf{q}'), \varepsilon, \mathbf{A}) = \{(\mathbf{p}, \mathbf{q}'), \mathbf{w}) : (\mathbf{p}, \mathbf{w}) \in \delta(\mathbf{q}, \varepsilon, \mathbf{A})\}$.

Example 9.22

There is a deterministic pushdown-automaton which accepts iff two types of symbols appear in the same quantity, say $\mathbf{0}$ and $\mathbf{1}$ and which never gets stuck:

- $\mathbf{Q} = \{\mathbf{s}, \mathbf{t}\}$; $\mathbf{s}$ is start state and accepting;
- $\mathbf{0}, \mathbf{1} \in \mathbf{\Sigma}$;
- $\mathbf{N} = \{\mathbf{S}, \mathbf{T}, \mathbf{U}, \mathbf{V}, \mathbf{W}\}$ with start symbol $\mathbf{S}$;
- $\delta(\mathbf{q}, \mathbf{a}, \mathbf{A}) = \{(\mathbf{q}, \mathbf{A})\}$ for all $\mathbf{a} \in \mathbf{\Sigma} - \{\mathbf{0}, \mathbf{1}\}$ and $\mathbf{A} \in \mathbf{N}$;
  $\delta(\mathbf{s}, \mathbf{0}, \mathbf{S}) = \{(\mathbf{t}, \mathbf{US})\}$; $\delta(\mathbf{s}, \mathbf{1}, \mathbf{S}) = \{(\mathbf{t}, \mathbf{TS})\}$;
  $\delta(\mathbf{t}, \mathbf{1}, \mathbf{U}) = \{(\mathbf{s}, \varepsilon)\}$; $\delta(\mathbf{t}, \mathbf{0}, \mathbf{U}) = \{(\mathbf{t}, \mathbf{VU})\}$;
  $\delta(\mathbf{t}, \mathbf{1}, \mathbf{V}) = \{(\mathbf{t}, \varepsilon)\}$; $\delta(\mathbf{t}, \mathbf{0}, \mathbf{V}) = \{(\mathbf{t}, \mathbf{VV})\}$;
  $\delta(\mathbf{t}, \mathbf{0}, \mathbf{T}) = \{(\mathbf{s}, \varepsilon)\}$; $\delta(\mathbf{t}, \mathbf{1}, \mathbf{T}) = \{(\mathbf{t}, \mathbf{WT})\}$;
  $\delta(\mathbf{t}, \mathbf{0}, \mathbf{W}) = \{(\mathbf{t}, \varepsilon)\}$; $\delta(\mathbf{t}, \mathbf{1}, \mathbf{W}) = \{(\mathbf{t}, \mathbf{WW})\}$;
  δ takes value $\emptyset$ everywhere else.

Closure under Union and Intersection

Theorem 9.23
The deterministic context-free languages are closed neither under union and nor under intersection.

Proof
The language $\mathbf{L_{a,b} = \{w \in \{0,1,2\}^* : w}$ has as many $\mathbf{a}$ as $\mathbf{b\}}$ is deterministic context-free (where $\mathbf{(a,b) = (0,1),(1,2)}$).

So is $\mathbf{L_{1,2} \cap 0^*1^*2^*}$. However,

$$\mathbf{L_{0,1} \cap (L_{1,2} \cap 0^*1^*2^*) = \{0^n1^n2^n : n \in \mathbb{N}\}}$$

is neither context-free nor deterministic context-free.

As $\mathbf{L \cap H = \Sigma^* - ((\Sigma^* - L) \cup (\Sigma^* - H))}$ for all $\mathbf{L, H}$ and as deterministic context-free languages are closed under complement, the deterministic context-free languages are also not closed under union.

Other Closure Properties

Answer the questions and give reasons for the answer.

Exercise 9.24
Show that the language $\mathbf{L = \{0^n10^m : n \geq m\}}$ is deterministic context-free. What about $\mathbf{L^*}$?

Exercise 9.25
Assume that $\mathbf{L}$ is deterministic context-free and $\mathbf{H}$ is regular. Is it always true that $\mathbf{L \cdot H}$ is deterministic context-free?

Exercise 9.26
Assume that $\mathbf{L}$ is deterministic context-free and $\mathbf{H}$ is regular. Is it always true that $\mathbf{H \cdot L}$ is deterministic context-free?

Exercise 9.27
Is $\mathbf{L^{mi}}$ deterministic context-free whenever $\mathbf{L}$ is?

Derivatives

Exercise 9.28
Assume that $\mathbf{L}$ is recognised by a grammar in Greibach normal form such that for every $\mathbf{b} \in \mathbf{\Sigma}$ and every nonterminal $\mathbf{A} \in \mathbf{N}$ there is exactly one rule $\mathbf{A} \to \mathbf{bw}$ with $\mathbf{w} \in \mathbf{N}^*$ in the grammar. Show that there is a finite family of languages $\mathbf{H_1, \ldots, H_n}$ with $\mathbf{H_1} = \mathbf{L}$, $\mathbf{H_2} = \{\varepsilon\}$ and $\mathbf{H_3} = \emptyset$ such that for every $\mathbf{a} \in \mathbf{\Sigma}$ and every $\mathbf{H_k}$, the derivative $(\mathbf{H_k})_\mathbf{a}$ is either an $\mathbf{H}_\ell$ or a product of several of the $\mathbf{H}_\ell$. Note that $\emptyset_\mathbf{a} = \emptyset$ for all $\mathbf{a} \in \mathbf{\Sigma}$.

Exercise 9.29
Assume $\mathbf{L}$ is prefix-free and $\mathbf{L} \neq \{\varepsilon\}$ and $\mathbf{L}$ satisfies that every derivative of $\mathbf{L}$ is the product of some fixed languages $\mathbf{H_1, \ldots, H_n}$. Is then $\mathbf{L}$ is recognised by a grammar in Greibach normal form where for every $\mathbf{A} \in \mathbf{N}$ and $\mathbf{b} \in \mathbf{\Sigma}$ there is at most one rule in the grammar of form $\mathbf{A} \to \mathbf{bw}$?

Additional Exercises

Exercise 9.30

Consider the language $\mathbf{L}$ of all ternary words which have as many $\mathbf{0}$ as $\mathbf{1}$. Show that every derivative of $\mathbf{L}$ is the product of several items of $\mathbf{L}$, $\mathbf{L_0}$ and $\mathbf{L_1}$ but that there is no grammar in Greibach normal form for $\mathbf{L}$ which has for every $\mathbf{A} \in \mathbf{N}$ and $\mathbf{b} \in \mathbf{\Sigma}$ at most one rule of the form $\mathbf{A} \to \mathbf{bw}$ with $\mathbf{w} \in \mathbf{N}^*$.

Exercise 9.31

Consider the context-free language $\mathbf{L}$ over the alphabet $\{\mathbf{f}, (,), \mathbf{0}, \mathbf{1}, , \}$ with the last symbol being a comma. The rules of the grammar are $\mathbf{S} \to \mathbf{f(S, S)|0|1}$ and create all expressions of a binary function $\mathbf{f}$ from $\{\mathbf{0}, \mathbf{1}\}^\mathbf{2}$ to $\{\mathbf{0}, \mathbf{1}\}$. Construct for $\mathbf{L}$ a grammar in Greibach normal form where for each pair $(\mathbf{A}, \mathbf{b})$ there is at most one rule $\mathbf{A} \to \mathbf{bw}$ in the grammar.

Characterisation

Exercise 9.32

Prove the following rules of the derivative:

$(\mathbf{L} \cup \mathbf{H})_\mathbf{x} = \mathbf{L}_\mathbf{x} \cup \mathbf{H}_\mathbf{x}$;

If $\varepsilon \in \mathbf{L}$ and $\mathbf{a} \in \mathbf{\Sigma}$ then $(\mathbf{L} \cdot \mathbf{H})_\mathbf{a} = \mathbf{L}_\mathbf{a} \cdot \mathbf{H} \cup \mathbf{H}$ else $(\mathbf{L} \cdot \mathbf{H})_\mathbf{a} = \mathbf{L}_\mathbf{a} \cdot \mathbf{H}$.

Theorem 9.33

A language $\mathbf{L}$ is context-free iff there is a finite list of languages $\mathbf{H_1}, \mathbf{H_2}, \ldots, \mathbf{H_n}$ with $\mathbf{L} = \mathbf{H_1}$ such that for every word $\mathbf{x}$ and every $\mathbf{H_m}$, $(\mathbf{H_m})_\mathbf{x}$ is a finite union of finite products of some $\mathbf{H_k}$.

Exercise 9.34

Prove this theorem.