Turing Machines and The Chomsky Hierarchy

November 24, 2011

Grammars

- Grammar $G = (\Sigma, N, S, R)$
  - Σ : set of terminal symbols
  - N : set of non-terminal symbols
  - $S \in N$: start symbol
  - $R \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$: finite set of rules

Grammars

- Grammar $G = (\Sigma, N, S, R)$
  - Σ : set of terminal symbols
  - N : set of non-terminal symbols
  - $S \in N$: start symbol
  - $R \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$: finite set of rules
- Relation $\rightarrow \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$

Grammars

- Grammar $G = (\Sigma, N, S, R)$
  - Σ : set of terminal symbols
  - N : set of non-terminal symbols
  - $S \in N$: start symbol
  - $R \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$: finite set of rules
- Relation $\rightarrow \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$
- Relation $\rightarrow^* \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$ is the reflexive-transitive closure of $\rightarrow$

Languages generated by grammars

Lemma

The class of languages generated by grammars is the class of recursively enumerable languages

Languages generated by grammars

Lemma

The class of languages generated by grammars is the class of recursively enumerable languages

- We can enumerate all possible derivations of strings from S
  1. $L := [S]$
  2. Pop out the first element of L (call it x)
  3. if $x \in \Sigma^*$ then *print* x
     else for each derivation rule applicable on x add the result of the application of the rule as the last element of L
  4. if $L \neq [\,]$ go to step 2

Languages generated by grammars

- Grammar derivations can be used to simulate the moves of a Turing Machine, where the string being manipulated represents the Turing Machine's configuration
  - we define non-terminal symbols R, L
  - For all $((q, \sigma), (q', \sigma', \rightarrow)) \in \delta$ we create the derivations $S\sigma, q, \sigma_i \rightarrow R\sigma'\sigma_i, q$, for all σ_i
  - For all $((q, \sigma), (q', \sigma', \leftarrow)) \in \delta$ we create the derivation $S\sigma, q, \rightarrow L, q', \sigma'$
  - For all $((q, \sigma), (q', \sigma', -)) \in \delta$ we create the derivation $S\sigma, q \rightarrow S\sigma', q'$
  - We create the derivations $\sigma_i L \rightarrow S\sigma_i$ for all σ_i
  - We create the derivations $R\sigma_i \rightarrow \sigma_i S$ for all σ_i
  - For all $((q, \sigma), (yes, \sigma', m))$ we create the derivation $S\sigma, q, \rightarrow \sigma'$

$x \in L(G)$?

Lemma

Given grammar G and $x \in \Sigma^*$ it is undecideable whether $x \in L(G)$

$x \in L(G)$?

Lemma

Given grammar G and $x \in \Sigma^*$ it is undecideable whether $x \in L(G)$

- The *Halting Problem (HP)* is recursively enumerable so there is a grammar G so that $L(G) = HP$

$x \in L(G)$?

Lemma

Given grammar G and $x \in \Sigma^*$ it is undecideable whether $x \in L(G)$

- The *Halting Problem (HP)* is recursively enumerable so there is a grammar G so that $L(G) = HP$
- We could construct a non-deterministic Turing Machine M that simulates the rule applications of G ...

Context-sensitive Grammars

Definition

A context-sensitive grammar is a grammar for which whenever $(x, y) \in R$ *we have* $|x| \leq |y|$

Context-sensitive Grammars

Definition

A context-sensitive grammar is a grammar for which whenever $(x, y) \in R$ *we have* $|x| \leq |y|$

Example

There is a context-sensitive grammar that generates the language $L = \{xx : x \in \Sigma^*\}$

$$
\begin{aligned}
S &\rightarrow a_i a_i \\
S &\rightarrow a_i A_i \\
S &\rightarrow a_i S a_i \\
S &\rightarrow a_i S A_i \\
A_i a_j &\rightarrow a_j A_i \\
A_i \epsilon &\rightarrow a_i \epsilon
\end{aligned}
$$

where $a_i, a_j \in \Sigma$ and $A_i \in N \setminus \{S\}$

$x \in L(G)$?

Lemma

Given grammar G and $x \in \Sigma^*$ it is decideable whether $x \in L(G)$

$x \in L(G)$?

Lemma

Given grammar G and $x \in \Sigma^*$ it is decideable whether $x \in L(G)$

There is a non-deterministic algorithm (we can construct a non-deterministic Turing Machine) that decides this problem

$x \in L(G)$?

Lemma

Given grammar G and $x \in \Sigma^*$ it is decideable whether $x \in L(G)$

There is a non-deterministic algorithm (we can construct a non-deterministic Turing Machine) that decides this problem

1. $w := S$, $Store := \emptyset$
2. Choose $y \notin Store$ such that $w \rightarrow^1 y$
   if $|x| \leq |y|$ then *halt* with "no"
   else if $y = x$ then *halt* with "yes"
   else $Store := Store \cup \{w\}$, $w := y$, *repeat*

$x \in L(G)$?

Lemma

Given grammar G and $x \in \Sigma^*$ it is decideable whether $x \in L(G)$

There is a non-deterministic algorithm (we can construct a non-deterministic Turing Machine) that decides this problem

1. $w := S$, $Store := \emptyset$
2. Choose $y \notin Store$ such that $w \longrightarrow^1 y$
   if $|x| \leq |y|$ then *halt* with "no"
   else if $y = x$ then *halt* with "yes"
   else $Store := Store \cup \{w\}$, $w := y$, *repeat*

Due to the restriction $(x, y) \Longrightarrow |x| \leq |y|$ we need at most $\sum_{i=0}^{|x|} |\Sigma|^i$ steps to surpass the length of x

Languages generated by context-sensitive grammars

Lemma

The class of languages generated by context-sensitive grammars is precisely $\mathbf{NSPACE}\,(n)$

Languages generated by context-sensitive grammars

Lemma

The class of languages generated by context-sensitive grammars is precisely **NSPACE** (n)

- There is a non-deterministic algorithm that decides $L(G)$ with additional space n
  1. Choose $x_1, x_2, \ldots, x_k$ so that
     $S \rightarrow x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_k$ and
     $|x_i| \leq |x|$
  2. if $x_k \equiv x$ then *halt* with "yes" else *halt* with "no"

Languages generated by context-sensitive grammars

Lemma

The class of languages generated by context-sensitive grammars is precisely $\mathbf{NSPACE}(n)$

- There is a non-deterministic algorithm that decides $L(G)$ with additional space n
  1. Choose $x_1, x_2, \ldots, x_k$ so that
     $S \to x_1 \to x_2 \to \cdots \to x_k$ and
     $|x_i| \leq |x|$
  2. if $x_k \equiv x$ then *halt* with "yes" else *halt* with "no"
- If a non-deterministic Turing Machine using additional space n decides L then there is a context-sensitive G such that $L = L(G)$
  - The string representation of the machine's configuration has length $n + 3$ at most
  - We can use $\sqcup$ (blank) as a terminal symbol of G
  - We can design the grammar rules so that the string being manipulated has always length $n + 3$ (padding with $\sqcup$)
  - But this is a context-sensitive grammar...

Context-free Grammars

Definition

A grammar is context-free if, for all rules $(x, y) \in R$, $x \in N$

Context-free Grammars

Definition

A grammar is context-free if, for all rules $(x, y) \in R$, $x \in N$

Example

There is a context-free grammar that generates the language of balanced parentheses

$$S \rightarrow SS$$
$$S \rightarrow (S)$$
$$S \rightarrow ()$$
$$S \rightarrow \epsilon$$

$x \in L(G)$?

Lemma

Given grammar G and $x \in \Sigma^*$ it is in **P** *to decide whether $x \in L(G)$.*

$x \in L(G)$?

Lemma

Given grammar G and $x \in \Sigma^*$ it is in **P** *to decide whether $x \in L(G)$.*

Papadimitriou gives a dynamic-programming algorithm which solves this problem in polynomial time

Right-linear Context-free Grammars

Definition

A context-free grammar is right-linear if $R \subseteq N \times (\Sigma N \cup \{\epsilon\})$

Right-linear Context-free Grammars

Definition

A context-free grammar is right-linear if $R \subseteq N \times (\Sigma N \cup \{\epsilon\})$

Example

There is right-linear context-free languages that generates all strings of 1, 0 *that end in* 101

$$S \to 0S$$
$$S \to 1S$$
$$S \to 1A$$
$$A \to 0B$$
$$B \to 1C$$
$$C \to \epsilon$$

Languages generated by right-linear context-free grammars

Lemma

The class of languages generated by right-linear context-free grammars are precisely the regular languages

Languages generated by right-linear context-free grammars

Lemma

The class of languages generated by right-linear context-free grammars are precisely the regular languages

- For every such grammar G we can contruct a NFA that accepts $L(G)$
  1. For every non-terminal symbol of G we create a new state for the NFA
  2. For every rule $A \to aB$ we create a transition $A_s \xrightarrow{a} B_s$
  3. For every rule $A \to \epsilon$ we define state A_s to be an accepting state

Languages generated by right-linear context-free grammars

Lemma

The class of languages generated by right-linear context-free grammars are precisely the regular languages

- For every such grammar G we can contruct a NFA that accepts $L(G)$
  1. For every non-terminal symbol of G we create a new state for the NFA
  2. For every rule $A \rightarrow aB$ we create a transition $A_s \xrightarrow{a} B_s$
  3. For every rule $A \rightarrow \epsilon$ we define state A_s to be an accepting state
- For every language L accepted by a DFA we can contruct a right-linear contex-free grammar G such that $L = L(G)$
  1. For every transition $q \xrightarrow{a} p$ we create a rule $Q \rightarrow aP$ (we add Q, P in N if they are not already there)
  2. For every accepting state q we create a rule $Q \rightarrow \epsilon$ (we add Q in N if it is not already there)