Regular Grammars and Closure Properties of Regular Languages

CS154
Chris Pollett
Outline

• Regular Grammars
• Closure Properties of Regular Languages
Grammars

- We now consider a different way to look at the regular languages based on grammars.

- A **grammar** is defined as a 4-tuple $G=(V, T, S, P)$ where $V$ is a finite set of **variables**, $T$ is a finite set of **terminal** symbols, $S \in V$ is called the start variable, and $P$ is a finite set of **productions** of the form $v \rightarrow w$ where $v$ is in $(V \cup T)^+$ and $w$ is in $(V \cup T)^*$.

- For example, let $G=(\{<sentence>, <noun>, <verb>\}, \{dog, cat, walks, eats\}, <sentence>, P)$ where $P$ is
  
  - $<sentence> \rightarrow <noun> <verb>
  - $<sentence> \rightarrow <noun> <verb> <noun>
  - $<noun> \rightarrow dog \mid cat$ /* we are using $\mid$ to abbreviate two line $<noun> \rightarrow$
    
    - dog and $<noun> \rightarrow cat */
  - $<verb> \rightarrow walks \mid eats$
  - $<noun> <verb> \rightarrow <sentence>

- Beginning with the start variable, a grammar can **yield** or **generate** a string over the alphabet of terminals via a finite sequence of substitutions:
  
  - $<sentence> \Rightarrow <noun> <verb> \Rightarrow dog <verb> \Rightarrow dog walks$

  We write $<sentence> \Rightarrow* dog walks$ to indicate from the string $<sentence>$ we can get dog walks via a finite sequence of substitutions.

- We write $L(G)$ for the set of strings generated by a grammar.
Regular Grammars

- A grammar $G=(V, T, S, P)$ is called **right-linear** if all its productions are of the form $A \rightarrow xB$ or $A \rightarrow x$ for some $A, B$ in $V$ and $x$ in $T^*$.
- A grammar is called **left-linear** if all its productions are of the form $A \rightarrow Bx$ or $A \rightarrow x$ for some $A, B$ in $V$ and $x$ in $T^*$.
- A grammar is called **regular** if it is either left or right linear.
- For example, $G=(\{S\}, \{a,b\}, S, P)$ where $P$ contains $S \rightarrow abS \mid \lambda$ is right linear. It generates the strings in $(ab)^*$.
- For example, $G=(\{S, A\}, \{a,b\}, S, P)$ where $P$ contains $S \rightarrow Sab \mid A$, $A \rightarrow Aba \mid ba$ is left linear. It generates the strings in $(ba)^+(ab)^*$.
- The set of rules $S \rightarrow A$, $A \rightarrow aB \mid \lambda$, $A \rightarrow Ab$ are all linear (so could belong to a **linear grammar**). The second rule is right linear, and the third is left linear, so these rules together could not belong to either a right linear or left linear grammar.
Equivalence with Regular Languages

- Need to show every language generated by a regular grammar is regular and vice-versa.
- In class we will only look at right linear grammars, but a similar argument can be made for left linear grammars. To begin:

**Theorem.** Let $G=(V, T, S, P)$ be a right linear grammar. Then $L(G)$ is a regular language.

**Proof.** Let $V=\{V_0, \ldots, V_n\}$. Assume $S=V_0$. The alphabet of our NFA will be the set of terminals. The set of states of our NFA will consist of $V_0, \ldots, V_n$ together with some auxiliary states and the state $f$ which will be the unique accepting state. The start state will be $V_0$. The transition function $\delta$ will be based on the productions of $G$. A production $V_i \rightarrow a_1 \ldots a_m V_j$ will map to the sequence of states and transitions:

\[
V_i \rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_{m-1} \rightarrow a_m \rightarrow V_j
\]

where the unlabelled states are auxiliary states. A production of the form $V_i \rightarrow a_1 \ldots a_m$ is mapped to a set of transitions:

\[
V_i \rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_{m-1} \rightarrow a_m \rightarrow F
\]

Given this description of the NFA, one can observe that $V_0 \Rightarrow^* w$ if and only if $\delta^*(V_0, w) = f$ and so if and only if $w$ is accepted by the NFA.
Regular implies Regular Grammar

**Theorem.** If $L$ is a regular language, then it is generated by some regular grammar.

**Proof.** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for $L$. Assume $Q=\{q_0, \ldots, q_n\}$ and $\Sigma=\{a_0, \ldots, a_m\}$. Let $G=(V,\Sigma, S, P)$ be the grammar with $V=\{q_0, \ldots, q_n\}$ and $S= q_0$ and where for each transition $\delta(q_i, a_j) = q_k$ we have the production $q_i \rightarrow a_j q_k$ and if $q_k$ is in $F$ we also have the production $q_k \rightarrow \lambda$. It is not hard to see that $w$ is accepted by $M$ iff it is generated by this right linear grammar.
Closure Properties

• Last day we argued that the regular languages are closed under union, concatenation and $\ast$.
• Today, we will look at some further closure properties.
• To begin…

**Theorem.** The regular languages are closed under complement.

**Proof.** A regular language $L$ is accepted by some DFA $M=(Q, \Sigma, \delta, q_0, F)$. Let $M´=(Q, \Sigma, \delta, q_0, Q-F)$. This machine will accept precisely those strings in $\Sigma^*$ which are not accepted by $M$. i.e., $\bar{L}$.
Direct Product Construction for DFAs

**Theorem.** If $A_1$ and $A_2$ two regular languages, so is their intersection $A_1 \cap A_2$.

**Proof:** Let $M_1=(Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2=(Q_2, \Sigma, \delta_2, q_2, F_2)$ be the DFAs recognizing $A_1$ and $A_2$. We would like make a new DFA, $M$, which simultaneously simulates both $M_1$ and $M_2$ and accepts a string $w$ if both $M_1$ and $M_2$ accepts. To simulate both machines at the same time we use a so-called cartesian product construction. Let $Q = Q_1 \times Q_2$. $M$’s alphabet is $\Sigma$ like that of $M_1$ and $M_2$. Define $\delta((q, q'), a) = (\delta_1(q,a), \delta_2(q',a))$. Let the start state be $(q_1, q_2)$. Finally, let $F = (F_1 \times F_2)$.

**Corollary.** If $A_1$ and $A_2$ two regular languages, so is $A_1 - A_2$.

**Proof.** Notice $A_1 - A_2 = A_1 \cap \overline{A}_2$. 
Closure under Reversals

**Theorem.** If $L$ is regular then so is $L^R = \{w^R \mid w \text{ is in } L\}$. Here $w^R$ is $w$ written backwards.

**Proof.** Let $N = (Q, \Sigma, \delta, s, F)$ be an NFA for $L$. Recall from our proof that $L(N)$ can be generated by a regular expression, that we can assume $N$ has only one accept state $f$. Let $N'$ be the NFA obtained from $N$ by making $f$ the start state, $s$ the only accept state, and for each transition $\delta(q, a) = q'$ having instead the transition $\delta(q', a) = q$. This machine will recognize a string iff the reverse was in $N$. 