Context Free Languages

Mark Greenstreet, CpSc 421, Term 1, 2008/09

Lecture Outline

Context Free Languages

- $a^n b^n$ – One More Time
- Formal Definition
- More Examples

$\mathsf{a}^n\mathsf{b}^n$ – one more time

- Let $A = \mathsf{a}^n\mathsf{b}^n$. A is not regular.
- Here's an inductive definition of the language. A string, w, is in A iff
  - $w = \epsilon$, or
  - There is a string, $x \in A$ such that $w = \mathsf{a}x\mathsf{b}$.
- Can we formalize this approach?
  - Why formalize?

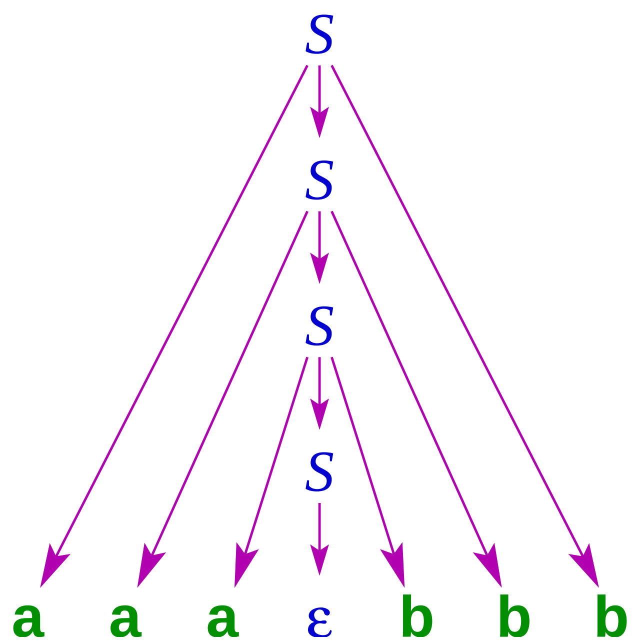
    We formalize the definition of languages so we can reason about properties that every language in some class has. That way, we don't have to prove properties individually.
  - Why not inductive definitions with English?

    Because it's not possible/practical to determine what can and cannot be said in English. How would you write an English sentence to state something that can't be said in English?

A Notation for Describing $\mathbf{a}^n\mathbf{b}^n$

- $S \rightarrow \epsilon \mid \text{a}\ S\ \text{b}$
- A string is in the language if we can derive it from S using these two rules.
- Example: aaabbb

$$\begin{aligned} S &\rightarrow \text{a}\ S\ \text{b} \\ &\rightarrow \text{aa}\ S\ \text{bb} \\ &\rightarrow \text{aaa}\ S\ \text{bbb} \\ &\rightarrow \text{aaa}\ \epsilon\ \text{bbb} = \text{aaabbb} \end{aligned}$$



$\#$0's $= \#$1's

- Let B be the language of strings that have an equal number of 0's and 1's.
- From the Sept. 5 notes,
  - $w = \epsilon$; or
  - There is a string x in B such that $w = 0x1$ or $w = 1x0$; or
  - There are strings x and y in B such that $w = xy$.
- Can we write this in our new notation?

$\#$`0`’s $=$ $\#$`1`’s

- Let B be the language of strings that have an equal number of `0`’s and `1`’s.
- From the Sept. 5 notes,
  - $w = \epsilon$; or
  - There is a string x in B such that $w = \texttt{0}x\texttt{1}$ or $w = \texttt{1}x\texttt{0}$; or
  - There are strings x and y in B such that $w = xy$.
- Can we write this in our new notation?

$$
\begin{array}{rcl}
B & \rightarrow & \epsilon \\
 & | & \texttt{0}\ B\ \texttt{1} \\
 & | & \texttt{1}\ B\ \texttt{0} \\
 & | & B\ B
\end{array}
$$

$\#$**0’s** $<$ $\#$**1’s**

- Let C be the language of strings that have an fewer `0`’s than `1`’s.
- String w is in C iff
  - There are strings x and y in B such that $w = x\texttt{1}y$, where B is the set of all strings with an equal number of ones and zeros as defined in the problem statement.
  - There are strings x and y in C such that $w = xy$.
- In our new notation, this is:

$\#\mathtt{0}$'s $< \#\mathtt{1}$'s

- Let C be the language of strings that have an fewer 0's than 1's.
- String w is in C iff
  - There are strings x and y in B such that $w = x\mathtt{1}y$, where B is the set of all strings with an equal number of ones and zeros as defined in the problem statement.
  - There are strings x and y in C such that $w = xy$.
- In our new notation, this is:

$$
\begin{array}{rcl|l}
C & \rightarrow & B\ \mathtt{1}\ B & C\ C \\
B & \rightarrow & \epsilon & B\ B \\
 & | & \mathtt{0}\ B\ \mathtt{1} & \mathtt{1}\ B\ \mathtt{0}
\end{array}
$$

Formalizing Our Notation

- A context-free grammar (CFG) is a 4-tuple, (V, Σ, R, S) where
  - V is a finite set of variables.
  - Σ is a finite set of terminals. $\Sigma \cap V = \emptyset$.
  - R is a finite set of rules.
    - Each rule is a tuple of the form (v, s) where $v \in V$ is a variable and $s \in (V \cup \Sigma)^*$ is as string of variables and/or terminals (possilby empty).
    - The interpretation is that any occurrence of v can be replaced with s.
    - We will write $v \longrightarrow s$ to indicate the tuple (v, s).
  - $S \in V$ is the start variable.

Derivations

CFGs give a set of rules for deriving strings of symbols and terminals.

- A single step derivation:
  - If $w = uAv$ with $w, u, v \in (V \cup \Sigma)^*$ and $A \in V$,
  - and $A \rightarrow x \in R$,
  - Then $w \Rightarrow uxv$,
  - and we say that w yields uxv.
- A multi-step derivation
  - We say that w derives x iff
    - We can find strings $v_0, v_1, \ldots, v_m$ such that
    - $v_0 = w$; and $v_m = x$; and
    - $v_{i-1} \Rightarrow v_i$ for all $i \in 1 \ldots m$.
  - We write $w \stackrel{*}{\Rightarrow} x$ if w derives x.

The Language Generated by a CFG

- Let $G = (V, \Sigma, R, S)$ be a CFG.
- The language generated by G is $L(G)$ where

$$L(G) \quad = \quad \{s \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} s\}$$

- Note that if $S \stackrel{*}{\Rightarrow} w$, w is a string in $(V \cup \Sigma)^*$.
  In other words, a derivation can, in general, produce a mixture of variables and terminals.
- However, $L(G)$ only includes strings with no variables –
  all variables must have been expanded into strings of terminals.

Regular Languages are Context Free

Proof: by induction on the definition of regular expressions.

- Let α be a regular expression with alphabet Σ.
- Case $\alpha = \emptyset$: Let $G = (\{S\}, \Sigma, \emptyset, S)$.
  With no productions, S cannot generate any string of terminals. Thus, $L(G) = \emptyset$.
- Case $\alpha = \epsilon$: Let $G = (\{S\}, \Sigma, \{S \to \epsilon\}, S)$.
  Only one derivation is possible: the single step derivation that yields ϵ. Thus, $L(G) = \{\epsilon\}$.
- Case $\alpha = \mathsf{c}$, for some $\mathsf{c} \in \Sigma$: Let $G = (\{S\}, \Sigma, \{S \to \mathsf{c}\}, S)$.
  $L(G) = \{\mathsf{c}\}$ by an argument like that for the previous case.
- Case $\alpha = \beta \cup \gamma$:
  Let $G_\beta = (V_\beta, \Sigma, R_\beta, S_\beta)$ and $G_\gamma = (V_\gamma, \Sigma, R_\gamma, S_\gamma)$ be CFGs that generate $L(\beta)$ and $L(\gamma)$ respectively. We assume that V_β and V_γ are dijoint.
  Let $G = (\{S\} \cup V_\beta \cup V_\gamma, \Sigma, \{S \to S_\beta, S \to S_\gamma\} \cup R_\beta \cup R_\gamma, S)$
  where $S \notin V_\beta \cup V_\gamma$.
  $L(G) = L(G_\beta) \cup L(G_\gamma) = L(\beta) \cup L(\gamma) = L(\beta \cup \gamma)$

  Proof details are on slide 18.

Regular Languages...(cont)

- Case $\alpha = \beta \cdot \gamma$:
  Let G_β and G_γ be CFGs that generate $L(\beta)$ and $L(\gamma)$ as above.
  Let $G = (\{S\} \cup V_\beta \cup V_\gamma, \Sigma, \{S \to S_\beta\, S_\gamma\} \cup R_\beta \cup R_\gamma, S)$
  where $S \notin V_\beta \cup V_\gamma$.
  $L(G) = L(G_\beta) \cdot L(G_\gamma) = L(\beta) \cdot L(\gamma) = L(\beta \cdot \gamma)$
  Proof details are on slide 22.
- Case $\alpha = \beta^*$: Let G_β generate $L(\beta)$ as above.
  Let $G = (\{S\} \cup V_\beta, \Sigma, \{S \to S\, S_\beta, S \to \epsilon\} \cup R_\beta, S)$
  $L(G) = L(G_\beta)^* = L(\beta^*)$. Proof details are on slide 24.

Regular Languages are Context Free

Proof by building a CFG that simulates a DFA.

- Let A be a regular language.
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes A.
  Assume that Q and Σ are disjoint.
- Define R as shown below:

$$
\begin{array}{rcl}
R & = & \{(q \rightarrow cp) \mid c \in \Sigma \text{ and } \delta(q,c) = p\} \\
 & \cup & \{(q \rightarrow \epsilon) \mid q \in F\}
\end{array}
$$

- Let $G = (Q, \Sigma, R, q_0)$ be a CFG. $L(G) = A$.
- Proof:
  - We can prove by induction that $\delta(q_0, w) = q$ iff G generates wq (see slide 26).
  - If G generates wq and $q \in F$, then G generates w.
    Thus, $L(G) \supseteq A$.
  - For the other direction, we prove by induction on the derivation that if $q_0 \overset{*}{\Rightarrow} w$, then either $w = wq$ where $\delta(q_0, w) = q$, or $w \in A$.
    Thus, $L(G) \subseteq A$.

Arithmetic Expressions

$G = (V, \Sigma, R, \mathit{Expr})$, where

$$
\begin{array}{rcl}
V & = & \{\mathit{Expr}, \mathit{ExprList}, \mathit{NonEmptyExprList}\} \\
\Sigma & = & \{\texttt{INTEGER}, \texttt{IDENTIFIER}, \texttt{PLUS}, \texttt{MINUS}, \\
& & \quad \texttt{TIMES}, \texttt{DIVIDE}, \texttt{EXP}, \\
& & \quad \texttt{LPAREN}, \texttt{RPAREN}, \texttt{COMMA}\}
\end{array}
$$

$$
\begin{array}{rcll}
\mathit{Expr} & \rightarrow & \texttt{INTEGER} & \mid \texttt{IDENTIFIER} \\
& \mid & \mathit{Expr}\ \texttt{PLUS}\ \mathit{Expr} & \mid \mathit{Expr}\ \texttt{MINUS}\ \mathit{Expr} \\
& \mid & \mathit{Expr}\ \texttt{TIMES}\ \mathit{Expr} & \mid \mathit{Expr}\ \texttt{DIVIDE}\ \mathit{Expr} \\
& \mid & \mathit{Expr}\ \texttt{EXP}\ \mathit{Expr} & \mid \texttt{LPAREN}\ \mathit{Expr}\ \texttt{RPAREN} \\
& \mid & \texttt{IDENTIFIER}\ \texttt{LPAREN}\ \mathit{ExprList}\ \texttt{RPAREN} &
\end{array}
$$

$$
\mathit{ExprList} \rightarrow \epsilon \mid \mathit{NonEmptyExprList}
$$

$$
\begin{array}{l}
\mathit{NonEmptyExprList} \rightarrow \mathit{Expr} \\
\quad \mid \mathit{NonEmptyExprList}\ \texttt{COMMA}\ \mathit{Expr}\ .
\end{array}
$$

Arithmetic Terminals

Regular Expressions:

$$\begin{aligned}
\texttt{INTEGER} &\equiv \texttt{DIGIT}\,\texttt{DIGIT}^* \\
\texttt{DIGIT} &\equiv \texttt{0} \cup \texttt{1} \cup \texttt{2} \cup \texttt{3} \cup \texttt{4} \cup \texttt{5} \cup \texttt{6} \cup \texttt{7} \cup \texttt{8} \cup \texttt{9} \\
\texttt{IDENTIFIER} &\equiv \texttt{ISTART}\,\texttt{ITAIL}^* \\
\texttt{ISTART} &\equiv \texttt{A} \cup \texttt{B} \cup \ldots \cup \texttt{Z} \cup \texttt{a} \cup \texttt{b} \cup \ldots \cup \texttt{z} \\
\texttt{ITAIL} &\equiv \texttt{ISTART} \cup \texttt{DIGIT}
\end{aligned}$$

PLUS	≡	+	MINUS	≡	-
TIMES	≡	*	DIVIDE	≡	/
EXP	≡	∧	COMMA	≡	,
LPAREN	≡	(	RPAREN	≡	)

Arithmetic Example

$2 + 3 * 4$

$$
\begin{aligned}
Expr &\Rightarrow Expr\ \texttt{PLUS}\ Expr \\
&\Rightarrow \texttt{INTEGER}\ \texttt{PLUS}\ Expr \\
&\Rightarrow \texttt{INTEGER}\ \texttt{PLUS}\ Expr\ \texttt{TIMES}\ Expr \\
&\Rightarrow \texttt{INTEGER}\ \texttt{PLUS}\ \texttt{INTEGER}\ \texttt{TIMES}\ Expr \\
&\Rightarrow \texttt{INTEGER}\ \texttt{PLUS}\ \texttt{INTEGER}\ \texttt{TIMES}\ \texttt{INTEGER}
\end{aligned}
$$

The Grammar of Java

See

- http://www.daimi.au.dk/dRegAut/JavaBNF.html, or
- http://www.cui.unige.ch/db-research/Enseignement/analyseinfo/JAVA/BNFindex.html

The coming week

Reading:

September 24 (Today): Introduction to Context Free Languages – *Sipser* 2.1.
Lecture will cover through “Designing Context-Free Grammars” (i.e. pages 99-105).

September 26 (Friday): Chomsky Normal Form
The rest of *Sipser* 2.1 (i.e. pages 105–109).

September 29 (Monday): Push-Down Automata – *Sipser* 2.2. Lectue will cover through page 114: “Examples of Push-Down Automata.”

October 1 (A week from today): Equivalence of CFGs and PDAs
The rest of *Sipser* 2.2.

Homework:

September 26 (Friday): Homework 2 due. Homework 3 goes out (due Oct. 3).
The due date for homework 3 will be strict – no late assignments will be accepted.

Midterm: Oct. 8

Proof for $\alpha = \beta \cup \gamma$

$L(G) \subseteq L(\beta \cup \gamma)$

- Let $s \in L(G)$. Thus, $S \stackrel{*}{\Rightarrow} w$.
- Because $S \neq w$, the derivation must have at least one step, in our notation $S \Rightarrow u \stackrel{*}{\Rightarrow} w$.
- u must be either S_β or S_γ from the definition of R (there are no other rules for S).
- If $u = S_\beta$, then $w \in L(G_\beta)$ by the construction of G. The induction hypothesis yields $L(G_\beta) = L(\beta)$, and thus $u \in L(\beta \cup \gamma)$ by the definition of the languages generated by regular expressions.
- If $u = S_\gamma$, the argument is similar.

$L(G) \supseteq L(\beta \cup \gamma)$ (see next slide).

Proof for $\alpha = \beta \cup \gamma$ (cont.)

$L(G) \supseteq L(\beta \cup \gamma)$

- If $w \in L(\beta \cup \gamma)$, then either $w \in L(\beta)$ or $w \in L(\gamma)$. We'll assume $w \in L(\beta)$; the other case is equivalent.
- $w \in L(G_\beta)$ by the induction hypothesis.
- $S_\beta \overset{*}{\Rightarrow} w$ by the definition of $L(G_\beta)$
- $S \Rightarrow S_\beta \overset{*}{\Rightarrow} w$ by the definition R.
- $w \in L(G)$ by the definition of $L(G)$.

Thus, $L(G) = L(\beta \cup \gamma)$ as claimed.

A lemma for concatenation

- Let $G = (V, \Sigma, R, S)$ be a CFG and let $x_1, x_2 \in (V \cup \Sigma)^*$ be strings of varibles and/or terminals.
- $x_1\, x_2 \stackrel{*}{\Rightarrow} w$ iff there are strings $w_1, w_2 \in (V \cup \Sigma)^*$ such that $x_1 \stackrel{*}{\Rightarrow} w_1$ and $x_2 \stackrel{*}{\Rightarrow} w_2$.
- Proof: If you think this is obvious, feel free to skip to slide 22.
  - If $x_1\, x_2 \stackrel{*}{\Rightarrow} w$, then $\exists w_1, w_2$ s.t. $x_1 \stackrel{*}{\Rightarrow} w_1$, $x_2 \stackrel{*}{\Rightarrow} w_2$, and $w = w_1 w_2$.
    By induction on the length of the derivation.
    - If the derivation has zero steps, then $w = x_1\, x_2$, and the result holds trivially (i.e. $x \stackrel{*}{\Rightarrow} x$ for any string $x \in (V \cup \Sigma)^*$).
    - If the derivation has $k+1$ steps, then $x_1\, x_2 \stackrel{k}{\Rightarrow} u \Rightarrow w$.
      - By the induction hypothesis, we can write $u = u_1 u_2$ where $x_1 \stackrel{*}{\Rightarrow} u_1$ and $x_2 \stackrel{*}{\Rightarrow} u_2$.
      - Furthermore, we can write $u = yVz$ where $V \to t$ and $w = ytz$.
      - If $|yV| \leq |u_1|$, then we write $u_1 = yVz_1$ and note that $z = z_1 u_2$. Then $u_1 \Rightarrow ytz_1$, and $u_2 \stackrel{0}{\Rightarrow} u_2$, and $w = (ytz_1) \cdot u_2$; showing the claim.
        If $|yV| > |u_1|$, then a similar argument applies where the last step is applied to u_2.
  - If $x_1 \stackrel{*}{\Rightarrow} w_1$ and $x_2 \stackrel{*}{\Rightarrow} w_2$, then $x_1 x_2 \stackrel{*}{\Rightarrow} w_1 w_2$. See next slide.

A lemma for concatenation (cont.)

- Proof (cont.)

  If $x_1 \stackrel{*}{\Rightarrow} w_1$ and $x_2 \stackrel{*}{\Rightarrow} w_2$, then $x_1 x_2 \stackrel{*}{\Rightarrow} w_1 w_2$.

  We show that $x_1 x_2 \stackrel{*}{\Rightarrow} w_1 x_2 \stackrel{*}{\Rightarrow} w_1 w_2$.

  We show that $x_1 x_2 \stackrel{*}{\Rightarrow} w_1 x_2$ by induction on the length of the derivation for $x_1 \stackrel{*}{\Rightarrow} w_1$.

  - If $x_1 \stackrel{0}{\Rightarrow} w_1$, then $w_1 = x_1$ and the result holds trivially.
  - If $x_1 \stackrel{k+1}{\Rightarrow} w_1$, then $x_1 \stackrel{k}{\Rightarrow} u \Rightarrow w_1$.
    - By the induction hypothesis, $x_1 x_2 \stackrel{*}{\Rightarrow} u x_2$.
    - Because $u \Rightarrow w_1$, we have $x_1 x_2 \stackrel{*}{\Rightarrow} w_1 x_2$ as claimed.

  The proof that $w_1 x_2 \stackrel{*}{\Rightarrow} w_1 w_2$ is similar.

  □

Proof for $\alpha = \beta \cdot \gamma$

$L(G) \subseteq L(\beta \cdot \gamma)$

- Let $w \in L(G)$. Thus, $S \stackrel{*}{\Rightarrow} w$.
- Because $S \neq w$, the derivation must have at least one step, in our notation $S \Rightarrow u \stackrel{*}{\Rightarrow} w$.
- u must be $S_\beta\, S_\gamma$ from the definition of R (there are no other rules for S).
- By the lemma from slide 20, there are strings w_β and w_γ such that $S_\beta \stackrel{*}{\Rightarrow} w_\beta$, $S_\gamma \stackrel{*}{\Rightarrow} w_\gamma$, and $w = w_\beta \cdot w_\gamma$.
- By the construction of G, $w_\beta \in L(G_\beta)$ and the induction hypothesis yields $w_\beta \in L(\beta)$. Likewise, $w_\gamma \in L(\gamma)$.
- Thus, $w = w_\beta \cdot w_\gamma \in L(\beta) \cdot L(\gamma) \in L(\beta \cdot \gamma)$ as required.

$L(G) \supseteq L(\beta \cdot \gamma)$ (see next slide).

Proof for $\alpha = \beta \cdot \gamma$ (cont.)

$L(G) \supseteq L(\beta \cdot \gamma)$

- If $w \in L(\beta \cdot \gamma)$, then there are strings w_β and w_γ such that $w_\beta \in L(\beta)$, $w_\gamma \in L(\gamma)$, and $w = w_\beta \cdot w_\gamma$.
- By the induction hypothesis, $S_\beta \stackrel{*}{\Rightarrow} w_\beta$ and S_γ derives w_γ.
- By the lemma from slide 20, $S_\beta S_\gamma \stackrel{*}{\Rightarrow} w_\beta w_\gamma$.
- From the construction of G, $S \rightarrow S_\beta\, S_\gamma$.
- Thus, $S \Rightarrow S_\beta, S_\gamma \stackrel{*}{\Rightarrow} w_1 w_2 = w$.
- $\therefore\ w \in L(G)$ as required.

Thus, $L(G) = L(\beta \cdot \gamma)$ as claimed.

Proof for $\alpha = \beta^*$

$L(G) \subseteq L(\beta^*)$

- Let $w \in L(G)$. Thus, $S \overset{*}{\Rightarrow} w$.
- Because $S \neq w$, the derivation must have at least one step, in our notation $S \Rightarrow u \overset{*}{\Rightarrow} w$.
- u must be either ϵ or $S\, S_\beta$ from the definition of R (there are no other rules for S).
- If $u = \epsilon$, then $w = \epsilon \in L(\beta^*)$
- Otherwise, $u = S\, S_\beta$, and by the lemma from slide 20, we can find w_1 and w_2 such that $S \overset{*}{\Rightarrow} w_1$, $S_\beta \overset{*}{\Rightarrow} w_2$ and $w = w_1 w_2$.
- $w_1 \in L(\beta^*)$ by induction on the derivation.
- $w_2 \in L(\beta)$ by the induction hypothesis for our induction on the definition of regular expressions.
- Thus, $w_1 w_2 \in L(\beta)$ by the definition of $L(\beta)$ as required.

$L(G) \supseteq L(\beta^*)$ (see next slide).

Proof for $\alpha = \beta \cdot \gamma$ (cont.)

$L(G) \supseteq L(\beta^*)$

- If $w \in L(\beta^*)$, then there is some $k \geq 0$ and strings $x_1, \ldots x_k$ such that $w = \prod_{i=1}^{k} x_i$ (with $\prod$ denoting concatenation). Our proof is by induction on k, and our induction hypothesis is $S \stackrel{*}{\Rightarrow} \prod_{i=1}^{k} x_i$.
- If $k = 0$, then $w = \epsilon \in L(G)$ because $S \to \epsilon$.
- If $k > 0$, then we note that $\prod_{i=0}^{k} x_i = \left(\prod_{i=0}^{k-1} x_i\right) x_k$.
- $S \stackrel{*}{\Rightarrow} \prod_{i=0}^{k-1} x_i$ by the induction hypothesis.
- $S_\beta \stackrel{*}{\Rightarrow} x_k$ by the induction hypothesis for our induction on the definition of regular expressions.
- Thus, $S \Rightarrow S\, S_\beta \stackrel{*}{\Rightarrow} \prod_{i=0}^{k} x_i = \left(\prod_{i=0}^{k-1} x_i\right) x_k$ by the definition of G and the lemma from slide 20.

Thus, $L(G) = L(\beta^*)$ as claimed.

DFA proof

Let M and G be a DFA and CFG as defined on slide 12.

Claim: $\delta(q_0, w) = q$ iff G generates wq.

If $\delta(q_0, w) = q$ then G generates wq – by induction on w.

- case $w = \epsilon$:
  $\delta(q_0, w) = \delta(q_0, \epsilon) = q_0$.
  $q_0 \overset{*}{\Rightarrow} q_0$ because any string derives itself in zero steps.
- case $w = x \cdot c$: $\delta(q_0, w) = \delta(\delta(q_0, x), c) = q$ by the definition of δ for strings.
  $q_0 \overset{*}{\Rightarrow} x\delta(q_0, x)$ by the induction hypothesis. Thus, $q_0 \overset{*}{\Rightarrow} wq$ as required.

If G generates wq then $\delta(q_0, w) = q$ – by induction on k, the length of the derivation.

- see the next slide.

DFA proof (cont.)

If G generates wq then $\delta(q_0, w) = q$ – by induction on k, the length of the derivation.

- case $k = 0$:
  $q_0 \stackrel{0}{\Rightarrow} q_0 = \epsilon q_0$, and $\delta(q_0, w) = \delta(q_0, \epsilon) = q_0$.
- case $k > 0$:

  By the induction hypothesis, there is some string $u \in \Sigma^*$ and some state $p \in Q$ such that $q_0 \stackrel{k-1}{\Rightarrow} up \Rightarrow wq$. Because p is the only variable in up, there must be a rule in R of the form $p \rightarrow x$ such that $ux = wq$. By the construction of R, x is of the form cq, and $\delta(p, c) = q$. Thus, $w = uc$ and $\delta(q_0, w) = q$ as required.

Remarks on the proofs

- I wrote these proofs to provide some more examples of proofs for the students in class who said that they would like to see more examples.
- While it seemed more intuitive to go from regular expressions to CFGs than to go from DFAs to CFGs, the latter proof turned out to be simpler.
- The basic ideas behined the regular expression to CFG proof were pretty simple. For each of the six ways to construct a regular expression, I showed a corresponding CFG. The tedium was that this created six lemmas that needed to be proven, and the last three needed some effort.
- In particular, the proofs get a bit more involved because they had nested inductions. The outer induction was over the definition of regular expressions. For some of the β^* case, there was in inner induction over the number of concatenated strings in the asteration.
- The DFA to CFG proof was comparativly simple. It involved creating a CFG that simulate the DFA. The string at each step of a derivation is the string of symbols that the DFA has read so far followed by the current state of the DFA.
- If the current state is accepting, the CFG can replace the state with ϵ and thus complete the derivation.