PDA to CFG Conversion

trip last time (3/12)

trip today (3/15)

CFG and PDA equivalence

- Theorem: **A language is context free if and only if some pushdown automaton recognizes it**

- Last lecture we saw one of the directions of this result: **If a language is context-free then some pushdown automaton recognizes it**
  - we saw how to convert a CFG G into an equivalent PDA P

- Today we will see the other direction: **If a pushdown automaton recognizes some language, then it is context free**
  - we will see how to convert a PDA P into an equivalent CFG G

Recap: CFG → PDA (Informal)

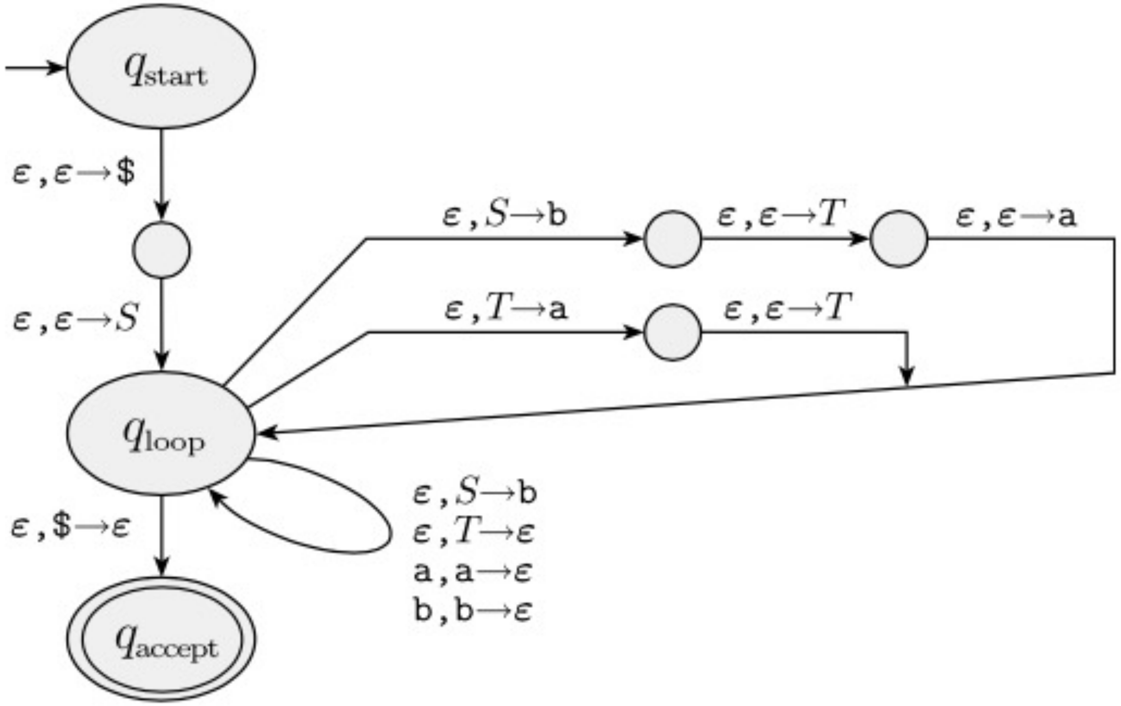
1. Place the marker symbol $ and the start variable on the stack
2. Repeat the following steps for ever
   a. If the top of stack is a variable symbol A, non-deterministically select one of the rules for A and substitute A by the string on the RHS of the rule.
   b. If the top of stack is a terminal 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 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.

Example

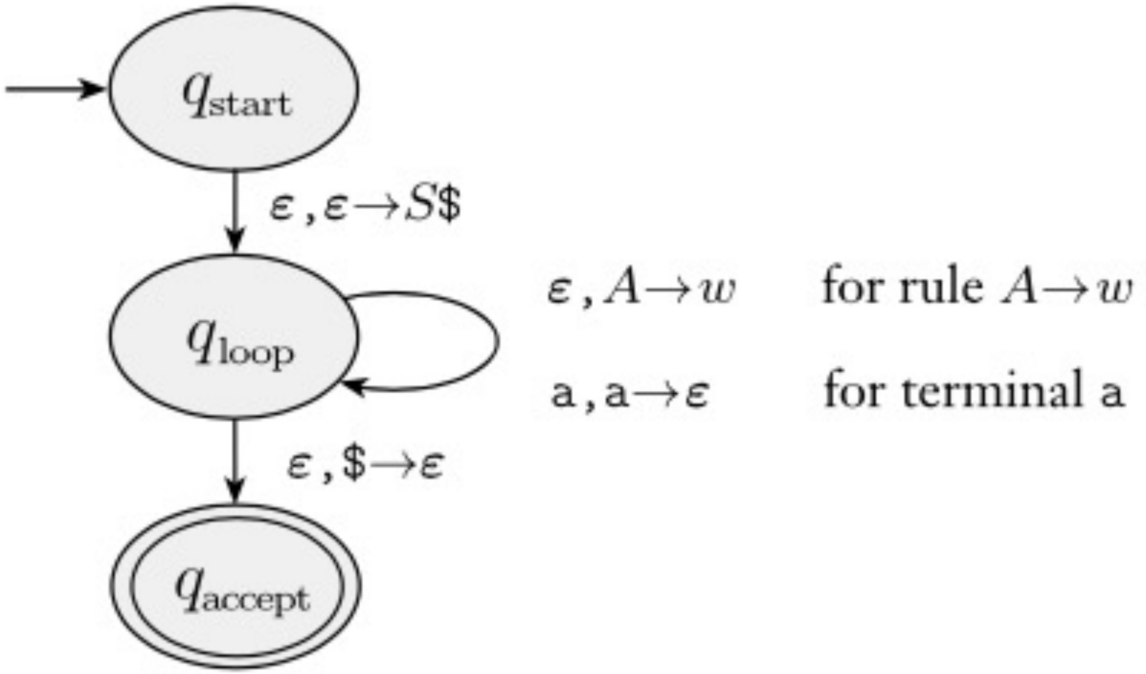
Convert the following CFG into a PDA

S ⟶ aTb | b

T ⟶ Ta | ε

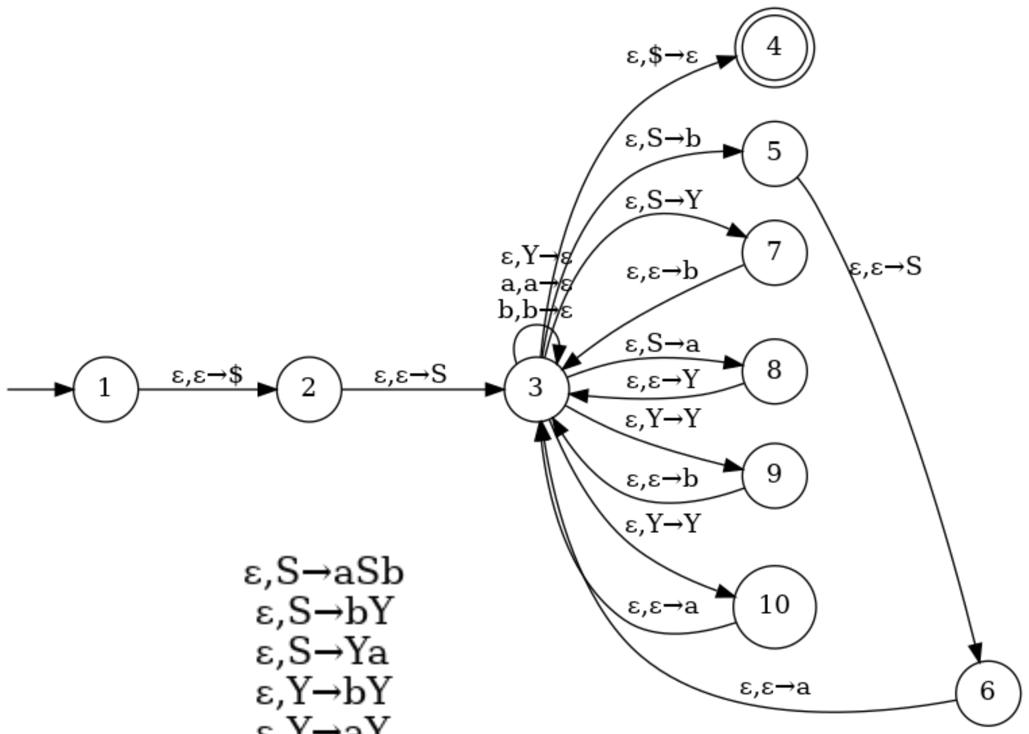


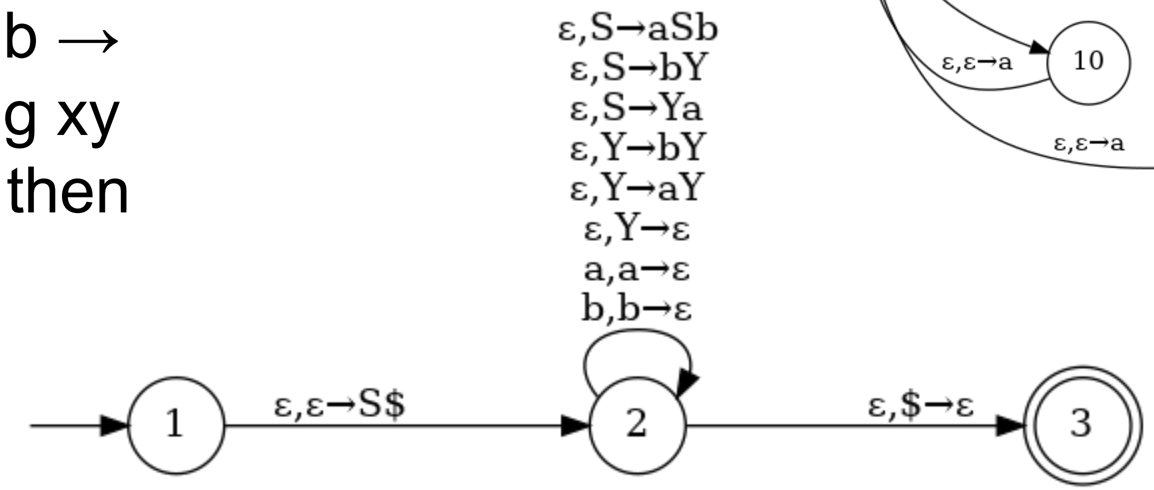
Recap: Diagram with Shorthand



Shorthand vs Full Diagram Example

- From our example of converting this grammar to a PDA
  - $S \to aSb \mid bY \mid Ya$
  - $Y \to bY \mid aY \mid \varepsilon$
- Additional correction from previous lecture: syntax of a,b → xy should be used for pushing xy onto stack, not for pushing x then y (don't reverse when using shorthand)





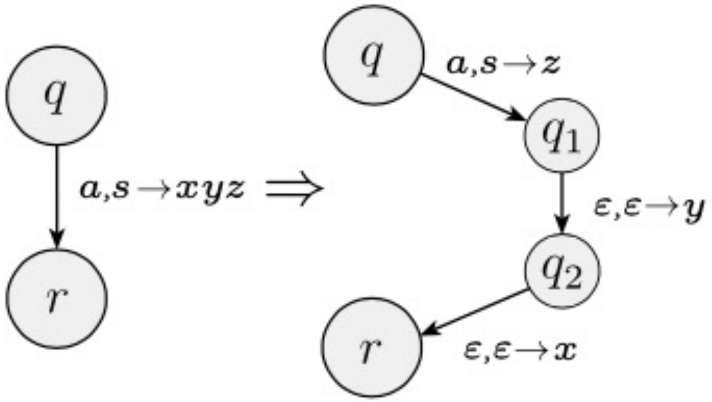
Recap: Shorthand Syntax

We use the notation $(r, u) \in \delta(q, a, s)$ to mean that when q is the state of the automaton, a is the next input symbol, and s is the symbol on the top of the stack, the PDA may

read the a and pop the s,

push the string u onto the stack and

go to the state r



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

Converting PDA to CFG

- Goal: given a PDA P, we want to construct a CFG G that generates all the strings that P accepts
  - In other words, G should generate a string if that string causes the PDA to go from its start state to an accept state

- Strategy: we design a grammar that does somewhat more
  - For **each pair** of states p and q in P, the grammar will have a variable A_{pq}
  - A_{pq} generates all the strings that can take P
    - ***from p with empty stack***
    - ***to q with empty stack***

Let us simplify our task

Modify P slightly to give it the following three features:

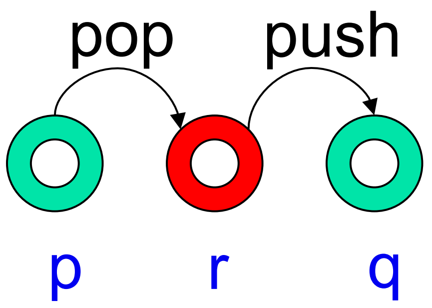
1. It has a **single accept state**, q_{accept}
2. It **empties** its stack before **accepting**
3. Each **transition** either **pushes** a symbol onto a stack or **pops** a symbol off the stack, but it **does not do both** at the same time

Let us simplify our task

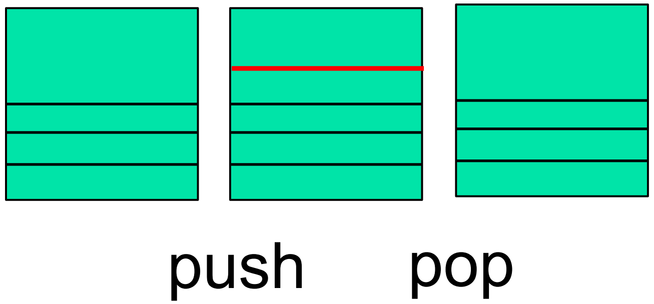
Giving P features 1 and 2 is easy.

To give it feature 3

- replace each transition that **simultaneously pops and pushes** with a *two-transition sequence that goes through a new state*



- replace each transition that **neither pops nor pushes** with *a two-transition sequence that pushes and then pops an arbitrary stack symbol*



Designing G

- To design G so that A_{pq} generates all strings that take P from p to q, starting and ending with an **empty stack**, we must understand how P operates on these strings.
  - For any such string x, P's **first** move on x mush be a *push*
  - the **last** move on x must be a *pop*

Two possibilities

Two possibilities occur during P's computation on x

1) The symbol popped at the end **is** the symbol pushed at the beginning
2) The symbol popped at the end **is not** the symbol pushed at the beginning

If (1) the stack could be empty only at the beginning and end of P's computation on x

If (2) the initially pushed symbol must get popped at some point before the end of x and thus the stack becomes empty at this point

The **former** possibility is simulated by the rule $\mathbf{A_{pq} \longrightarrow aA_{rs}b}$, where

a is the input read at the first move,

b is the input read at the last move,

r is the state following p, and

s is the state preceding q

The **latter** possibility is simulated by the rule $\mathbf{A_{pq} \longrightarrow A_{pr}\, A_{rq}}$, where

r is the state when when the stack becomes empty

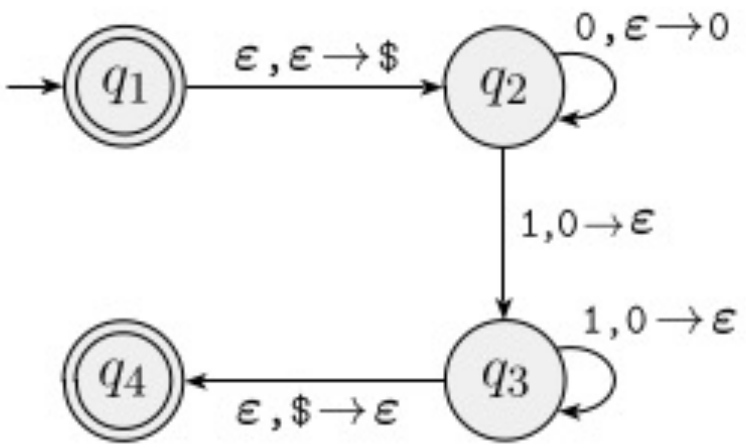
Summary

- Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ and construct G
- The variables of G are $\{A_{pq} \mid p, q \in Q\}$
- The start variable is $A_{q\_0,\ q\_accept}$
- We describe G's rules in three parts
  1. For each $p, q, r, s \in Q$; $u \in \Gamma$; and $a, b \in \Sigma_\varepsilon$; if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε), put the rule $A_{pq} \longrightarrow aA_{rs}b$ in G
  2. For each $p, q, r \in Q$, put the rule $A_{pq} \longrightarrow A_{pr}A_{rq}$ in G
  3. Finally, for each $p \in Q$, put the rule $A_{pp} \longrightarrow \varepsilon$ in G

- We can prove that this construction works by demonstrating that A_{pq} generates x if and only if x can bring P from p with empty stack to q with empty stack.

A simple example

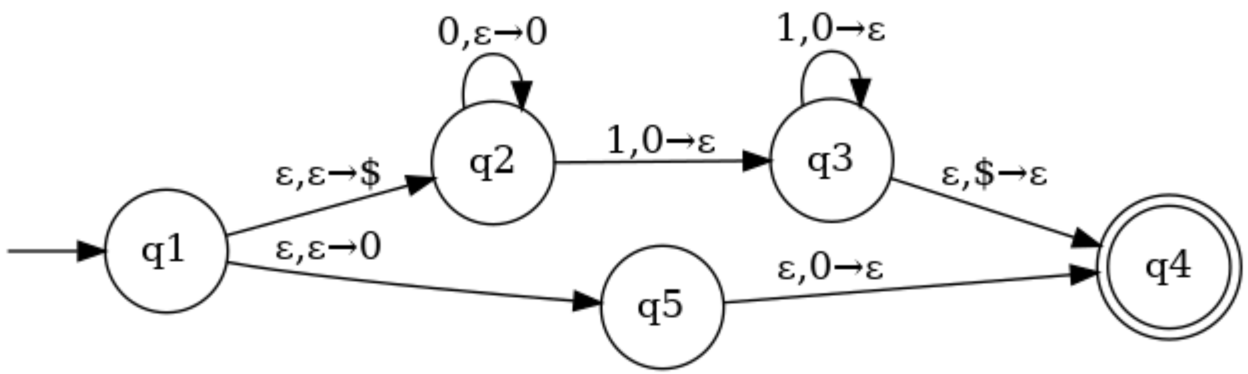
- Consider the first PDA introduced in the book
- Designed for the language { 0^n1^n | $n \geq 0$ }
- Requires very little transformation to our target format
  - All transitions already either push or pop one symbol
  - Needs to have just one accept state



A simple example (cont)

- First, make a transition from q1 to q4 through a dummy state (q5) to make P have the correct features.
- Next, create a start variable, A_{14}, representing the path from the start state (q1) to the accept state (q4).

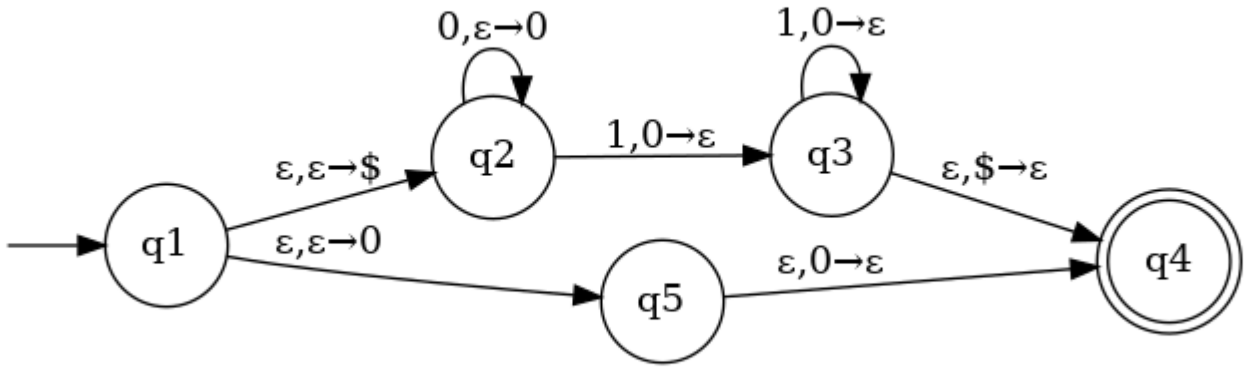
- $A_{14} \rightarrow$?



A simple example (cont)

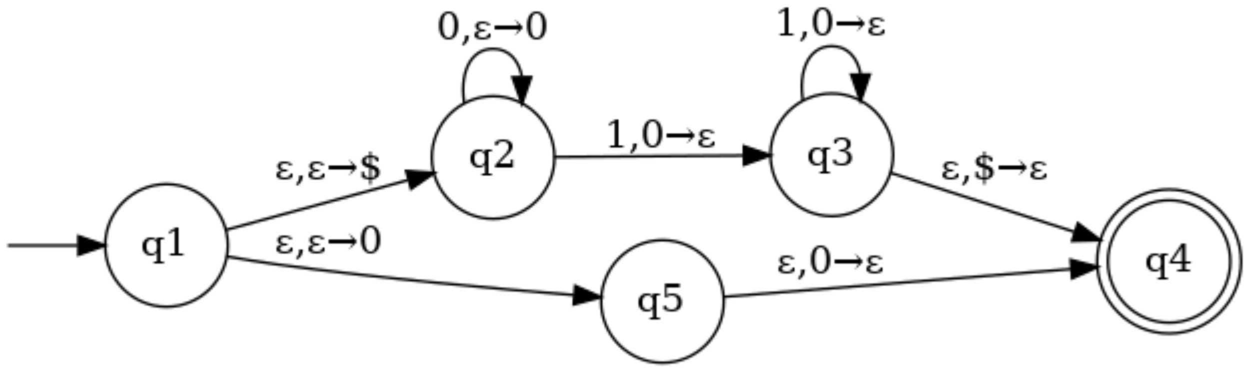
- There are two paths from q1 to q4, starting and ending with an empty stack. Both push and pop the same symbol.
- Accordingly, We use the formula $A_{pq} \rightarrow aA_{rs}b$, where r and s are the states after p and before q respectively, and a and b are the symbols read there.
- In both cases, we read nothing, so just use the variable.

- $A_{14} \rightarrow A_{23} \mid A_{55}$



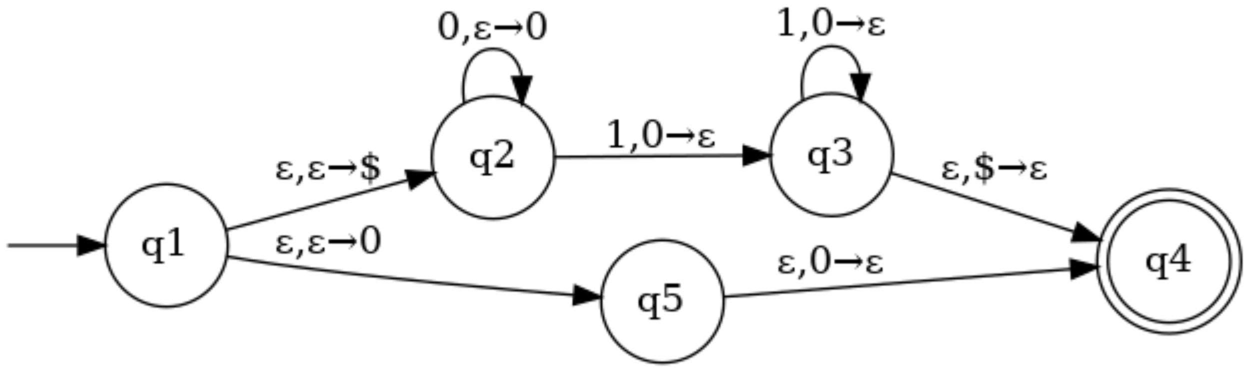
A simple example (cont)

- In the past slide, we generated a variable called A_{55} . This is representing a movement from state q5 to itself without a change in stack.
- In the grammar, all A_{pp} are ε. So we should add that rule.
- Processing A_{23} is more tricky
- $A_{14} \rightarrow A_{23} \mid A_{55}$
  $A_{55} \rightarrow \varepsilon$



A simple example (cont)

- Recall next our third rule for inserting rules in the grammar
- For each $p, q, r, s \in Q$; $u \in \Gamma$; and $a, b \in \Sigma_\varepsilon$; if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε), put the rule $A_{pq} \longrightarrow aA_{rs}b$ in G
- Consider that $(q2, 0) \in \delta(q2, 0, \varepsilon)$ and $(q3, \varepsilon) \in \delta(q3, 1, 0)$
- Thus we add $A_{23} \rightarrow 0A_{23}1$ to the grammar.
- We can similarly work out $A_{23} \rightarrow 0A_{22}1$



A simple example (cont)

- The final grammar is as follows

- $A_{14} \rightarrow A_{23} \mid A_{55}$
  $A_{23} \rightarrow 0A_{22}1 \mid 0A_{23}1$
  $A_{22} \rightarrow \varepsilon$
  $A_{55} \rightarrow \varepsilon$