More on Grammars and Their Languages

Robert M. Keller
Harvey Mudd College
27 April 2005

Notation

- Recall that $x \Rightarrow y$ means that there are strings u, v, v', w such that
  - $x = uvw$,
  - $y = uv'w$,
  - $v \rightarrow v'$ is a production
- Define $\Rightarrow^*$ to be the reflexive transitive closure of $\Rightarrow$:
  - $x \Rightarrow^* y$ means ($x = y$ or $\exists z$ ($x \Rightarrow^* z$ and $z \Rightarrow y$)).

What we’re assuming as background

- A language is defined to be regular if it is denoted by some regular expression.

- It has been shown that regular languages are equivalent to languages accepted by non–deterministic finite-state acceptors (NFA’s).

- Every NFA can be converted to a DFA that accepts the same language.

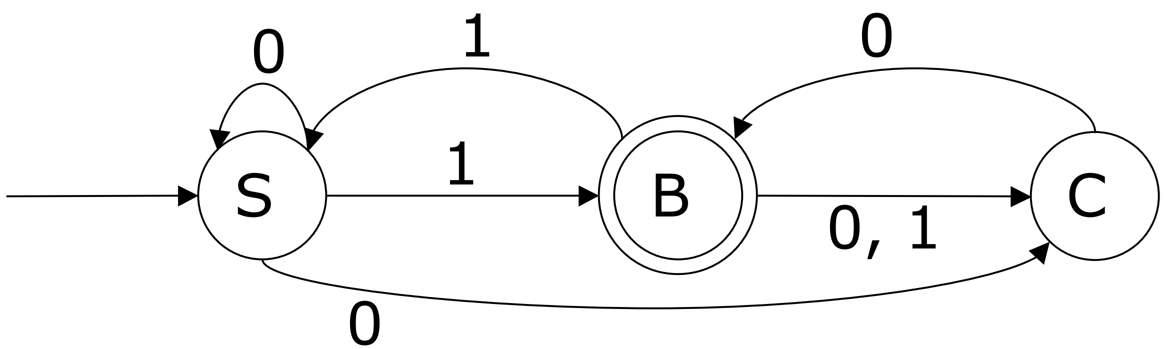
- Summarization: **Kleene’s Theorem** (1956):
  A language is regular iff it is accepted by a DFA.

A language is regular iff it is generated by some type 3 grammar.

- Type 3 productions are of one of two types:
  - $B \rightarrow \sigma C$, where $B \in A$, $\sigma \in \Sigma$
  - $B \rightarrow \Lambda$
- To prove this result, identify the states of a NFA with auxiliaries in the grammar. Assume a single start state and no Λ-transitions (WLOG).
  - $B \rightarrow \sigma C$ is a production if state B goes to state C via symbol σ.
  - $B \rightarrow \Lambda$ is a production iff B is an **accepting** state in the NFA.
- The language generated by the grammar is the language generated by the NFA.
- The only way to get rid of the auxiliary in the derived string is to use the production $B \rightarrow \Lambda$, which corresponds to the NFA being in an accepting state.

Example: NFA vs. Grammar

NFA:



Grammar:

- Start symbol is S
- Productions:

$S \rightarrow 0S$	$B \rightarrow 1S$	$C \rightarrow 0B$
$S \rightarrow 0C$	$B \rightarrow 0C$	
$S \rightarrow 1B$	$B \rightarrow 1C$	
	$B \rightarrow \Lambda$	

- Sample derivation: $S \Rightarrow 0S \Rightarrow 00C \Rightarrow 000B \Rightarrow 000$

Pumping Lemma for Regular Languages

- For any regular language L:

$$
\begin{aligned}
&(\exists n \in N)\ (\forall x \in L)\\
&\quad (|x| \geq n) \rightarrow\\
&\qquad ((\exists u\ \exists v\ \exists w)\\
&\qquad\quad x = uvw\\
&\qquad\quad \wedge\ v \neq \Lambda\\
&\qquad\quad \wedge\ |uv| \leq n\\
&\qquad\quad \wedge\ (\forall m \in N)\ uv^m w \in L)
\end{aligned}
$$

Note that m = 0 is included.

Proof of the Pumping Lemma (1)

- If L is regular, then there is a type 3 grammar G that generates L.
- Let n be the number of auxiliary symbols in G.
- If a string $x_1\, x_2\, x_3 \ldots x_r$ having length n or more is in the language, then the same auxiliary A was used at least twice in generating that string, with no prior or intervening uses of A

  $$S \Rightarrow x_1B_1 \Rightarrow x_1\, x_2B_2 \Rightarrow \ldots \Rightarrow x_1\, x_2 \ldots x_iA \Rightarrow \ldots$$
  $$\Rightarrow x_1\, x_2 \ldots x_ix_{i+1} \ldots x_jA \Rightarrow x_1\, x_2\, x_3 \ldots x_r$$

  (A, B_1 , B_2, … are distinct, so there can be at most n)

- Choose $u = x_1\, x_2 \ldots x_i$, $v = x_{i+1} \ldots x_j$, $v = x_{j+1} \ldots x_r$ and observe that the desired properties of these strings hold.

Proof of the Pumping Lemma (2)

$S \Rightarrow x_1 B_1 \Rightarrow x_1 x_2 B_2 \Rightarrow \ldots \Rightarrow x_1 x_2 \ldots x_i A \Rightarrow \ldots$
$\Rightarrow x_1 x_2 \ldots x_i x_{i+1} \ldots x_j A \Rightarrow x_1 x_2 x_3 \ldots x_r$

Chose $u = x_1 x_2 \ldots x_i$, $v = x_{i+1} \ldots x_j$, $v = x_{j+1} \ldots x_r$
Observe that

$$
\begin{aligned}
& x = uvw && = x_1 x_2 \ldots x_i x_{i+1} \ldots x_j x_{j+1} \ldots x_r \\
\wedge\ & v \neq \Lambda && = x_{i+1} \ldots x_j \\
\wedge\ & |uv| \leq n && (A\ B_1\ B_2 \ldots \text{distinct, at most } n) \\
\wedge\ & (\forall m \in N)\ uv^m w \in L && \text{since } A \Rightarrow^* x_{i+1} \ldots x_j A
\end{aligned}
$$

The distinctness property is a consequence of the pigeonhole principle.

Pigeonhole Principle

- If p pigeons are placed in h holes:
  if $h < p$, then some hole gets more than one pigeon.

- Contrapositive:
  - If no hole gets more than one pigeon, then $h \geq p$.

- In our case: The holes correspond to the n auxiliary symbols, while the pigeons correspond to the instances of those symbols in the derivation. The number of the latter is at least $n+1$ for a derivation of a string of length at least n.

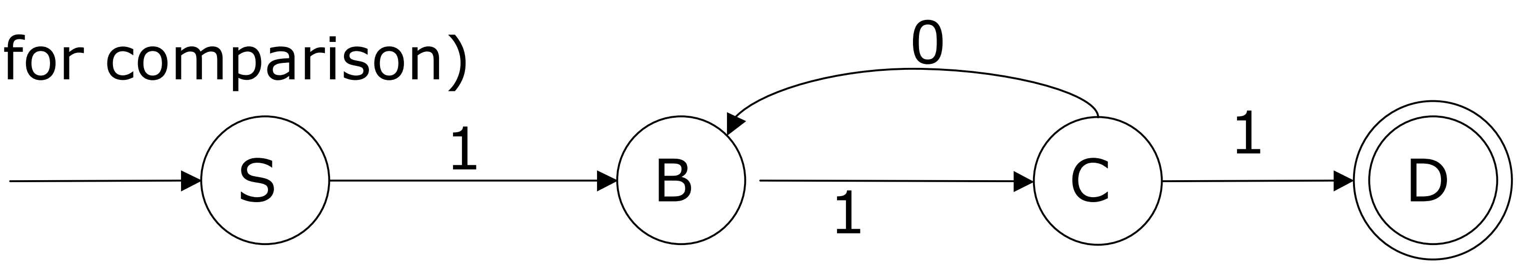
Pumping Lemma Example

Grammar:

- $S \rightarrow 1B$ $B \rightarrow 1C$ $C \rightarrow 0B$ $C \rightarrow 1D$ $D \rightarrow \Lambda$

- Here n = 4.
- Consider $11011 \in L$, which has length ≥ 4.
- Derivation is $S \Rightarrow 1B \Rightarrow 11C \Rightarrow 110B \Rightarrow 1101C \Rightarrow 11011D \Rightarrow 11011$
- B is the first repeated auxiliary, $B \Rightarrow^* 10B$.
- u = 1 v = 10 w = 11
- $(\forall m \in N)\ 1(10)^m 11 \in L$
- For example: $\{111, 11011, 1101011, 110101011, \ldots\} \subseteq L$.

(NFA for comparison)



Use of the Pumping Lemma

- The main use is to show that certain languages are *not* regular.

- That is, the n that must exist for a regular language *cannot* exist for the language in question.

Example of Pumping Lemma Use (1)

- The language $L = \{0^k1^k \mid k \in N\}$ is not regular.

- Proof: If L were regular, then let n be the number that exists according to the pumping lemma.
- Let $x = 0^n1^n \in L$.
- Let u, v, w be such that $x = uvw$, $v \neq \Lambda$, $|uv| \leq n$ and $(\forall m \in N)\ uv^mw \in L$.
- Since $|uv| \leq n$, and $v \neq \Lambda$, v must consist of one or more 0's.
- But then $uw \in L$ would have fewer 0's than 1's, contradicting the definition of L.

Example of Pumping Lemma Use (2)

- The language $L = \{1^p \mid p \text{ is prime}\}$ is not regular.

- Proof: If L were regular, then let n be the number that exists according to the pumping lemma.
- Let $x = 1^p \in L$. , where $p > n$ (since there are infinitely-many primes).
- Let u, v, w be such that $x = uvw$, $v \neq \Lambda$, $|uv| \leq n$ and $(\forall m \in N)\ uv^m w \in L$, according to the pumping lemma.
- Let $q = |v| > 0$, $r = |uw|$, so rephrasing, $(\forall m \in N)\ 1^r 1^{mq} \in L$.
- Since (taking $m = 0$) $1^r = 1^r 1^{0q} \in L$, we know that $r > 1$.
- In particular, for $m = r$, we have $1^r 1^{rq} = 1^{r(q+1)} \in L$.
- But $r(q+1)$ cannot be prime, giving a contradiction.

There are languages of type 2 that are **not regular**.

- $\{0^n 1^n \mid n \in N\}$ is known to be non-regular.
- But the following type 2 grammar generates it:
  - $S \to 0S1$
  - $S \to \Lambda$

Abstract States for Any Language

Defining the factoring function /

- Let $L \subseteq \Sigma^*$ be any language.
- For any $x \in \Sigma^*$, define

$$L/x = \{w \in \Sigma^* \mid xw \in L\}$$

- Example: $L = \{x \in \{1\}^* \mid |x| \text{ is a multiple of } 3\}$. Then
  - $L/\Lambda = L$.
  - $L/1 = \{x \in \{1\}^* \mid |x| \bmod 3 = 2\}$.
  - $L/11 = \{x \in \{1\}^* \mid |x| \bmod 3 = 1\}$.
  - $L/111 = \{x \in \{1\}^* \mid |x| \bmod 3 = 0\} = L$.
- Although the number of elements of $\{1\}^*$ is infinite, the number of **distinct** sets of the form L/x is **finite** in this case.

Another Example

- Example: $L = \{0^n1^n \mid n \in N\}$. Then
  - $L/\Lambda = L$.
  - $L/0 = \{0^n1^{n+1} \mid n \in N\}$.
  - $L/1 = \varnothing$.
  - $L/00 = \{0^n1^{n+2} \mid n \in N\}$.
  - $L/01 = \{\Lambda\}$.
  - $L/11 = \varnothing$.
  - $L/10 = \varnothing$.
  - $L/000 = \{0^n1^{n+3} \mid n \in N\}$.
  - etc.
- In this case, the number of **distinct** sets of the form L/x is **infinite**.

Abstract States

- We call the **sets** L/x for a language L the **abstract states** of the language.

- **Myhill-Nerode Theorem Variant**: A language is regular iff its set of abstract states is finite.

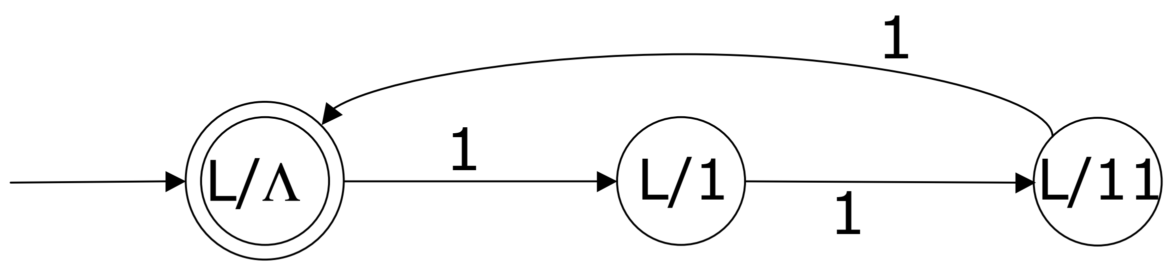
Proof of the Myhill-Nerode Theorem (1)

- ($\Leftarrow$) Suppose the set of abstract states of L is finite. Then we can define a (deterministic) finite-state acceptor M for L as follows:
- The states of M are the abstract states of L.
- The initial state is L/Λ.
- L/x is an accepting state iff $\Lambda \in L/x$.
- The next state function is defined by

  $f(L/x, \sigma) = L/(x\sigma)$.
- We still must show that f is **well-defined**; that is, the definition does not depend on which x in L/x is chosen for the definition. This is done in the following discussion.

Example: Myhill-Nerode

- $L = \{x \in \{1\}^* \mid |x| \text{ is a multiple of } 3\}$. Then
  - $L/\Lambda = L$.
  - $L/1 = \{x \in \{1\}^* \mid |x| \bmod 3 = 2\}$.
  - $L/11 = \{x \in \{1\}^* \mid |x| \bmod 3 = 1\}$.
  - $L/111 = \{x \in \{1\}^* \mid |x| \bmod 3 = 0\} = L$.
- The following DFA is constructed:



- In defining $f(L/\Lambda, 1) = L/1$, we get the same thing as for $f(L/111, 1)$, $f(L/111111, 1)$, etc. The next state is the same regardless.

Showing f is well-defined

- We claim that f, given by
  $f(L/x, \sigma) = L/(x\sigma)$
  is a well-defined function.

- How could it not be?
- L/x is just a set of strings, and as such, there could be **other** $y \neq x$ such that $L/x = L/y$.

- We need to show it doesn't matter whether we use x or y, i.e. that if $L/x = L/y$, then also $L/(x\sigma) = L/(y\sigma)$.

$L/x = L/y \rightarrow L/(x\sigma) = L/(y\sigma)$.

- Assume that $L/x = L/y$ and let $\sigma \in \Sigma$.
- $L/x = \{w \in \Sigma^* \mid xw \in L\} = \{w \in \Sigma^* \mid yw \in L\} = L/y$.
- Thus for any w, $xw \in L \leftrightarrow yw \in L$.
- In particular, for w of the form $\sigma w'$, where w' is arbitrary,
- $$x(\sigma w') \in L \leftrightarrow y(\sigma w') \in L.$$
- which is the same as saying $(x\sigma)w' \in L \leftrightarrow (y\sigma)w' \in L$.
- Thus $L/(x\sigma) = L/(y\sigma)$.

The Myhill-Nerode Equivalence Relation

- Let $L \subseteq \Sigma^*$ be any language.
- Define a binary relation $\equiv_L$ on Σ^* as follows:

$$x \equiv_L y \text{ iff } L/x = L/y$$

(The standard definition uses the equivalent:
$x \equiv_L y$ iff $((\forall w \in \Sigma^*)\ (xw \in L \leftrightarrow yw \in L))$.)

- Example: $L = \{x \in \{1\}^* \mid |x| \text{ is a multiple of } 3\}$:
- Here $\Lambda \equiv_L 111$, $111 \equiv_L 111111$, etc.
- Also, $1 \equiv_L 1111$, $111 \equiv_L 111111$, etc.
- All pairs that have the same length mod 3 are related.

$\equiv_L$ is an Equivalence Relation (for any L)

- Recall $x \equiv_L y$ on Σ^* is the same as:
  $L/x = L/y$

- Reflexive property: $x \equiv_L x$:

  i.e. $L/x = L/x$

- Symmetric property: $x \equiv_L y \rightarrow y \equiv_L x$:

  i.e. $L/x = L/y \rightarrow L/y = L/x$

- Transitive property: $x \equiv_L y \wedge y \equiv_L z \rightarrow x \equiv_L z$

  i.e. $(L/x = L/y \wedge L/y = L/z) \rightarrow L/x = L/z$

Equivalence Classes

- Any equivalence relation $\equiv$ on a set S induces a set of **equivalence classes**:

  Subsets C of S such that

  $$x, y \in C \text{ iff } x \equiv y.$$

  These subsets are disjoint and their union is S.

- The **index** of an equivalence relation is its number of equivalence classes.

- In our case, the equivalence classes are exactly the abstract states.

More than an Equivalence Relation

- In addition to being an equivalence relation, $\equiv_L$ satisfies the property of being a **congruence**:

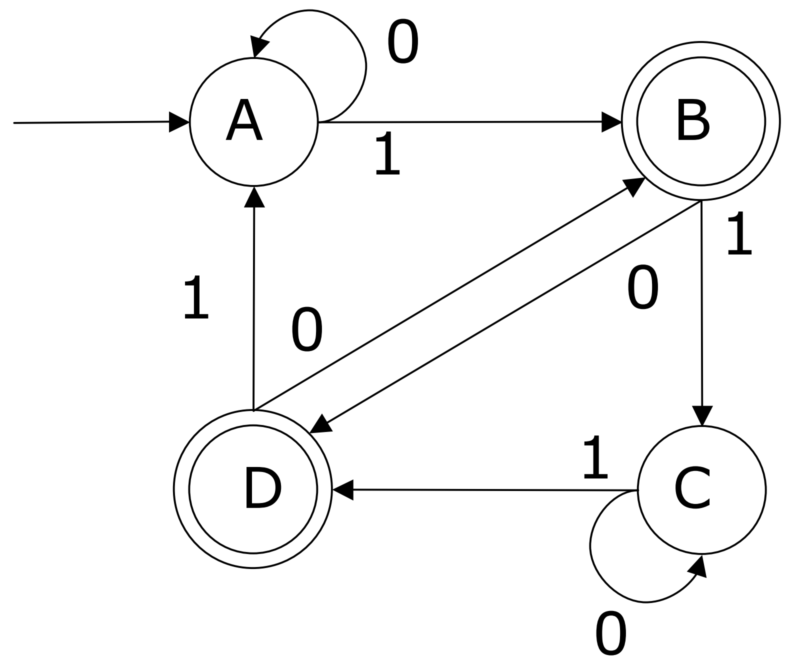
$$x \equiv_L y \rightarrow (\forall\sigma \in \Sigma)\ x\sigma \equiv_L y\sigma$$

- We proved this in the process of showing that the transition function f is well-defined.

Proof of the Myhill-Nerode Theorem (2)

- ($\Rightarrow$) Suppose L is regular. Then there is a DFA that accepts L.
- For each state q of the DFA, define S(q) to be the set of strings that lead from the initial state to q.
- For any strings x, y $\in$ S(q), for any string w, xw $\in$ L iff yw $\in$ L, since the **state q alone** determines whether or not xw $\in$ L.
- Thus any two strings in S(q) are $\equiv_L$ equivalent (but not necessarily conversely).
- In other words, each S(q) is a ***subset*** of an equivalence class of $\equiv_L$.
- Put another way, each equivalence class is the ***union*** of some of the sets S(q).
- Hence the number of equivalence classes of L is finite, because the number of states is finite.

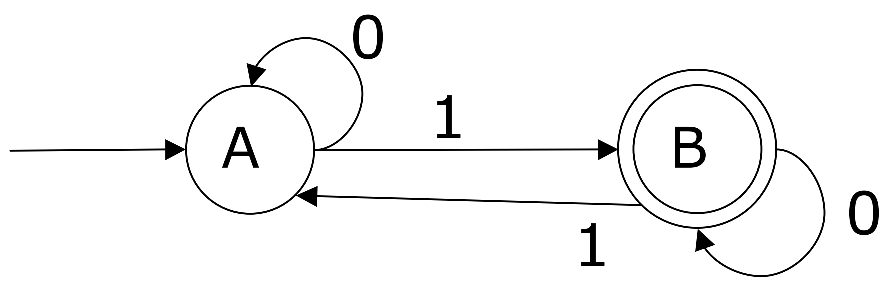
Example



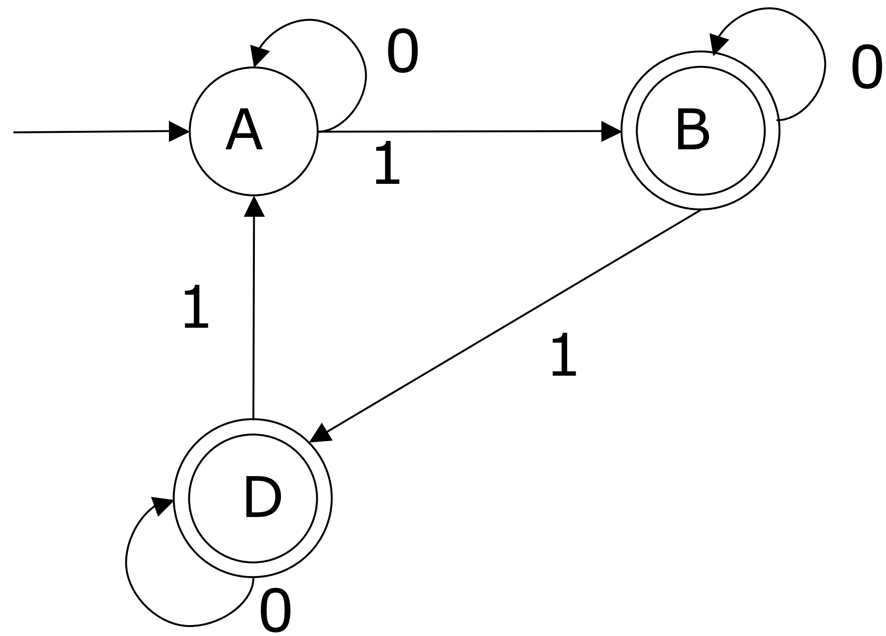
- The language accepted is
  $L = \{x \in \{0, 1\}^* \mid \text{the number of 1's is odd}\}$.
- There are two equivalence classes:
- $L = \{x \in \{0, 1\}^* \mid \text{the number of 1's is odd}\}$
- $L' = \{x \in \{0, 1\}^* \mid \text{the number of 1's is even}\}$
- $L = S(A) \cup S(C)$
- $L' = S(B) \cup S(D)$

Example, continued

- There are two equivalence classes:
- $L = \{x \in \{0, 1\}^* \mid$ the number of 1's is odd$\}$
- $L' = \{x \in \{0, 1\}^* \mid$ the number of 1's is even$\}$
- We can therefore construct the following (smaller, equivalent) DFA:



Example



- The language accepted is L = $\{x \in \{0, 1\}^* \mid$ the number of 1's is *not* divisible by 3$\}$.
- There are three equivalence classes.
- This shows that the language accepted is not necessarily an equivalence class by itself; in general, it will be the **union** of equivalence classes.

An Alternative to the Pumping Lemma for Showing a Language is not Regular

- If a language can be shown to have an infinite number of equivalence classes, then it is not regular.

- Example: $L = \{0^n1^n \mid n \in N\}$.
  - $L/\Lambda = L$.
  - $L/0 = \{0^n1^{n+1} \mid n \in N\}$.
  - $L/00 = \{0^n1^{n+2} \mid n \in N\}$.
  - $L/000 = \{0^n1^{n+3} \mid n \in N\}$.
  - etc.
  - For every k, $L/0^k = \{0^n1^{n+k} \mid n \in N\}$ is a distinct class. Hence the number of classes is infinite.

Regular Expression to DFA Directly

- Elsewhere is described the translation from regular expressions to NFA, and from there to DFA.

- The concept of abstract state provides another route.

- If L is given by a regular expression, then for any $\sigma \in \Sigma$, L/σ can be computed **symbolically**.

- We can do this repeatedly, and check for closure, by testing whether regular expressions are equivalent.

Symbolic Computation of L/σ
(sometimes called the "derivative" of L wrt σ)

- $\emptyset/\sigma = \emptyset$
- $\Lambda/\sigma = \emptyset$
- $\sigma/\sigma = \Lambda$
- $\sigma/\sigma' = \emptyset$ if $\sigma \neq \sigma'$
- $(R \cup S)/\sigma = (R/\sigma \cup S/\sigma)$
- $(RS)/\sigma = (R/\sigma)S$ if $\Lambda \notin L(R)$
- $(RS)/\sigma = (R/\sigma)S \cup S/\sigma$ if $\Lambda \in L(R)$
- $(R^*)/\sigma = (R/\sigma)R^*$

- The initial state is that of the regular expression for the language.

The accepting states are those for which $\Lambda \in$ their regular expression.

Example

- Construct a DFA accepting L(R) where R = $(01)^*0$.

- $((01)^*0)/0 = ((01)^*/0)0 \cup 0/0 = 1(01)^*0 \cup \Lambda$
- $((01)^*0)/1 = ((01)^*/1)0 \cup 0/1 = \varnothing$

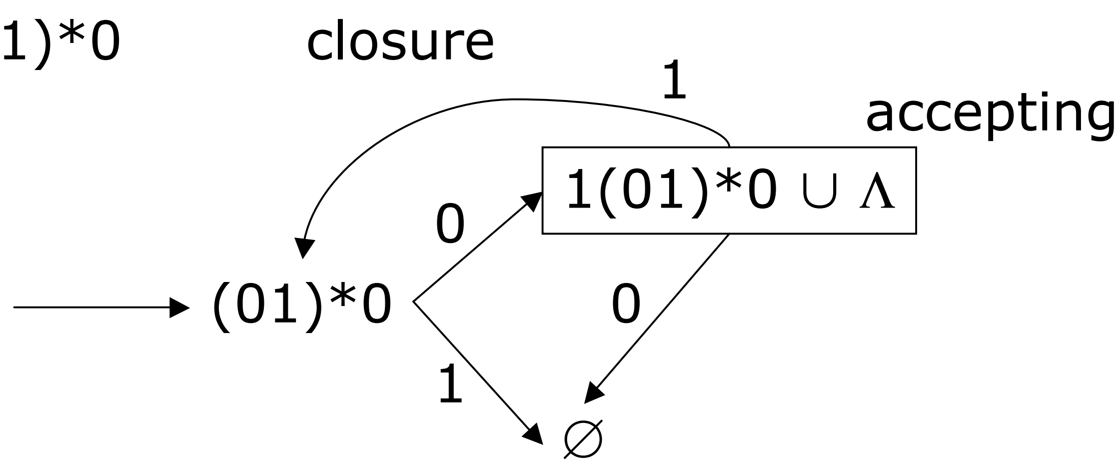
- $(1(01)^*0 \cup \Lambda)/0 = \varnothing$
- $(1(01)^*0 \cup \Lambda)/1 = (01)^*0$ closure

Example

- Construct a DFA accepting L(R) where R = (01)*0.

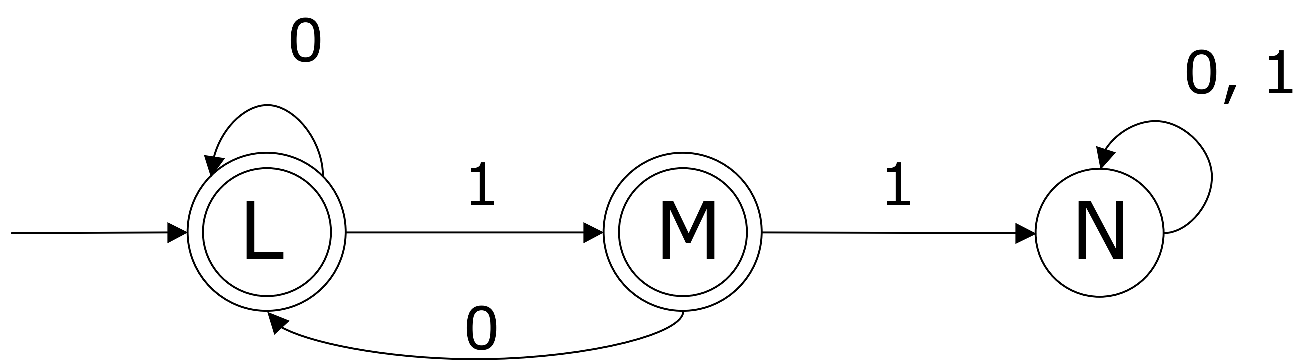
- $((01)^*0)/0 = ((01)^*/0)0 \cup 0/0 = 1(01)^*0 \cup \Lambda$
- $((01)^*0)/1 = ((01)^*/1)0 \cup 0/1 = \varnothing$

- $(1(01)^*0 \cup \Lambda)/0 = \varnothing$
- $(1(01)^*0 \cup \Lambda)/1 = (01)^*0$



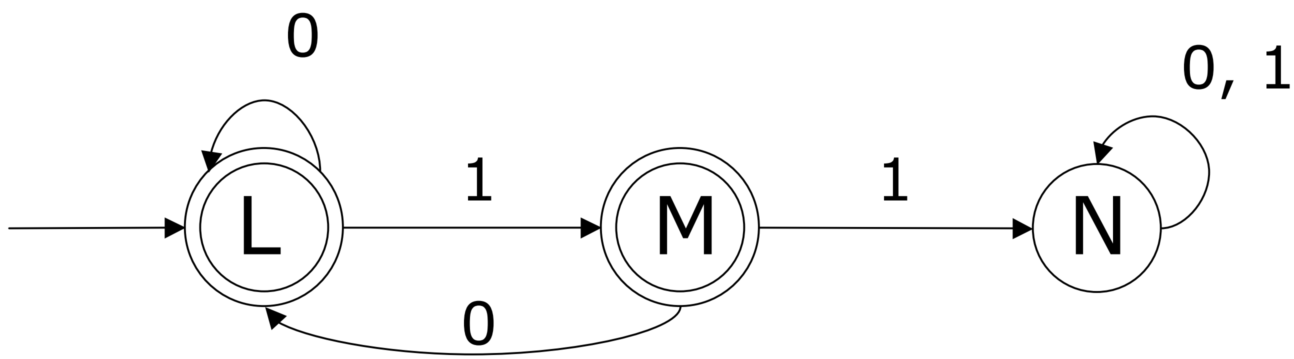
Regular Expression from DFA

- Label the States



- Identify each state with the set of paths from the start state to it. This set is a language.
- The language accepted by the FSA is the **union** of the paths to each of the accepting states, in this case $L \cup M$.

Deriving Closed Forms



- View the acceptor as a set of **regular-expression equations**:
  - $L = L0 \cup M0 \cup \Lambda$
  - $M = L1$
  - $N = M1 \cup N(0 \cup 1)$
- The Λ is on the RHS of the starting state only.
- We want to **solve** for L and M, and take the union of the solutions.

Solving a Regular-Expression Equation

- An equation $X = XA \cup B$ in one variable X
  has a **solution** $X = BA^*$ (which is unique provided that $\Lambda \notin A$).

  This is called **Arden's Rule** (Dean Arden, 1960).
- To see this intuitively, repeatedly substitute $XA \cup B$ for X:

$$
\begin{aligned}
X &= XA \cup B \\
&= (XA \cup B)A \cup B = XAA \cup BA \cup B \\
&= (XA \cup B)AA \cup BA \cup B = XAAA \cup BAA \cup BA \cup B \\
&= B(\{\Lambda\} \cup A \cup AA \cup AAA \cup \ldots) \\
&= BA^*
\end{aligned}
$$

- To see that BA^* is a solution, substitute for X:

$$
\begin{aligned}
&(BA^*)A \cup B \\
&= B(A^*A \cup \{\Lambda\}) \\
&= BA^*
\end{aligned}
$$

- (If $\Lambda \in A$, then BA^* is still the minimal solution but it is not unique; $(B \cup C)A^*$, for arbitrary C, is a solution.)

Example

- Consider the equation
  $$X = X(0 \cup 11) \cup 1$$
- The stated solution is $X = 1(0 \cup 11)^*$.
- $$\begin{aligned} 1(0 \cup 11)^* \overset{?}{=} & \ 1(0 \cup 11)^* (0 \cup 11) \cup 1 \\ = & \ 1((0 \cup 11)^* (0 \cup 11) \cup \Lambda) \\ = & \ 1(0 \cup 11)^* \end{aligned}$$

Example with $\Lambda \in A$

- Consider the equation
  $$X = X(0 \cup \Lambda) \cup 1$$
- The stated minimal solution is $X = 1(0 \cup \Lambda)^*$ but $(0 \cup \Lambda)^* = 0^*$, so $X = 10^*$.
- $$\begin{aligned} 10^* &?= 10^*(0 \cup \Lambda) \cup 1 \\ &= 10^*0 \cup 10^* \cup 1 \\ &= 10^* \end{aligned}$$
- But 1^*0^*, for example, is also a solution:
  $$\begin{aligned} 1^*0^* &?= 1^*0^*(0 \cup \Lambda) \cup 1 \\ &= 1^*0^*0 \cup 1^*0^* \cup 1 \\ &= 1^*0^* \end{aligned}$$

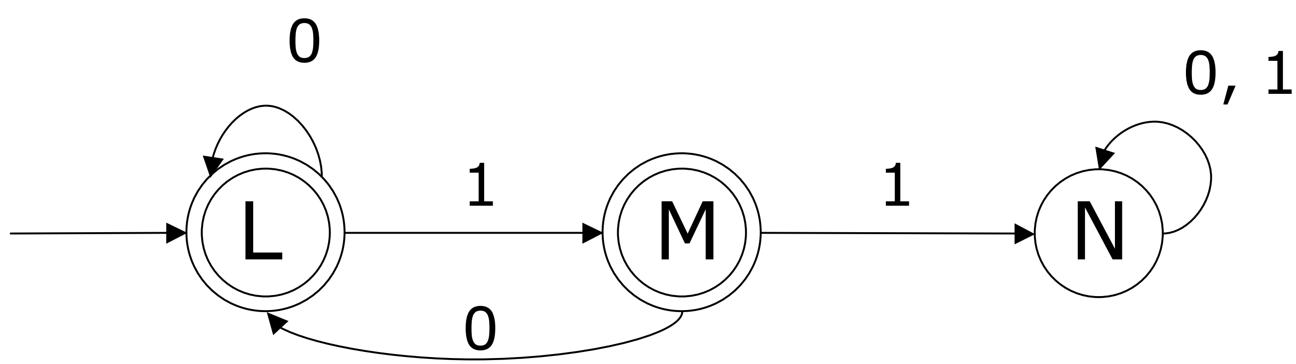
Solving Systems of RE Equations

- **Solve** for L and M:
  - $L = L0 \cup M0 \cup \Lambda$
  - $M = L1$
  - $N = M1 \cup N(0 \cup 1)$

- **Substitution** Operation:
  - A LHS variable can be replaced with its RHS, so replacing M in the L equation:
  - $L = L0 \cup L10 \cup \Lambda$, or more simply
  - $L = L(0 \cup 10) \cup \Lambda$
- **Elimination** Operation:
  - Using Arden's rule, $L = LA \cup B$ has the solution $L = BA^*$, so:
  - $L = \Lambda(0 \cup 10)^*$, or more simply $L = (0 \cup 10)^*$
- Substitution again:
  - $M = L1$
  - $M = (0 \cup 10)^*1$

Conclusion

- The language accepted by the DFA below is
  - $L \cup M$
  - which is $(0 \cup 10)^* \cup (0 \cup 10)^*1$
  - or, by factoring,
  - $(0 \cup 10)^*(\Lambda \cup 1)$



Summary: DFA $\Rightarrow$ RE Algorithm

- Express the DFA as a set of RE equations
  - Each state is a variable.
  - Each variable is equated to a union of expressions showing how to get to that state in one step from other states.
  - The start state has Λ on the RHS as well.
- Solve the RE equations for the variables:
  - The variables, along with their equations, are solved for one at a time.
  - Choose a variable for elimination.
  - Expression that variable in terms of the remaining variables only, using the * operator ($L = LA \cup B$ has the solution $L = BA^*$).
  - Substitute the solution for all occurrences of the variable in the remaining equations.
  - Repeat the above steps until no variables remain.
- Work backward, substituting the solutions found for other variables, until each variable is expressed in closed form.

Another Example

- **Solve**:
  - $L = L1 \cup M0 \cup N0 \cup \Lambda$
  - $M = L0 \cup M1 \cup N1$
  - $N = L1 \cup M1 \cup N0$

- Note that these equations don't really correspond to a DFA, but rather an NFA, but it doesn't matter.
- Eliminate N, using $N = (L1 \cup M1)0^*$
  - $L = L1 \cup M0 \cup (L1 \cup M1)0^*0 \cup \Lambda$
  - $M = L0 \cup M1 \cup (L1 \cup M1)0^*1$
- Regroup:
  - $L = L(1 \cup 10^*0) \cup M(0 \cup 10^*0) \cup \Lambda$
  - $M = L(0 \cup 10^*1) \cup M(1 \cup 10^*1)$

Solution, continued

- Solving:
  - $L = L(1 \cup 10^*0) \cup M(0 \cup 10^*0) \cup \Lambda$
  - $M = L(0 \cup 10^*1) \cup M(1 \cup 10^*1)$
- Eliminate M using $M = L(0 \cup 10^*1)\,(1 \cup 10^*1)$, giving:
  - $L = L(1 \cup 10^*0) \cup L(0 \cup 10^*1)\,(1 \cup 10^*1)(0 \cup 10^*0) \cup \Lambda$
- Regrouping:
  - $L = L((1 \cup 10^*0) \cup (0 \cup 10^*1)\,(1 \cup 10^*1)(0 \cup 10^*0)) \cup \Lambda$
- Solving:
  - $L = ((1 \cup 10^*0) \cup (0 \cup 10^*1)\,(1 \cup 10^*1)\,(0 \cup 10^*0))^*$
- Working backward:
  - $M = ((1 \cup 10^*0) \cup (0 \cup 10^*1)\,(1 \cup 10^*1)\,(0 \cup 10^*0))^*$ $(0 \cup 10^*1)\,(1 \cup 10^*1)$
  - $N = (L1 \cup M1)0^* = \ldots$

Summary for Regular Languages

- The following are equivalent:
  - L is denoted by some regular expression.
  - L is accepted by a DFA.
  - L is accepted by an NFA.
  - There is a type 3 grammar generating L.
  - The set of abstract states of L is finite.
  - L is the union of the equivalence classes of a Nerode-Myhill congruence relation of finite index.

Additional Paths to Regularity

- If L and M are regular, so are:
  - LM
  - $L \cup M$
  - L^*
  - $L \cap M$
  - $L - M$
  - $L^{reverse}$
  - prefixes(L)
  - suffixes(L)
  - substrings(L)
  - subsequences(L)
  - $L^{1/2}$
  - $L/M = \{x \mid (\exists w \in M)\ xw \in L\}$ M regular, or not!

Closure Under Substitution (Homomorphism)

- Suppose that L is a language over Σ.
- By a **substitution map**, we mean a function that assigns to each element of a string from an alphabet Δ.
- Example: $\Sigma = \{0, 1\}$, $\Delta = \{a, b, c\}$, $s(0) = ab$, $s(1) = cbaba$.

- We can “extend” s to map any language over by simply applying s to the letters in each string in the language and concatenating the results for that string.
- Example: $L = \{1\}^*\{0\}$

$$s(L) = \{cbaba\}^*\{ab\}$$

- Both type 3 and type 2 languages are closed under homomorphism.

Grammars vs. Regular Expressions

- Every regular expression also corresponds to some ***type 2*** grammar in a natural way, but not conversely. (The connection to a type 3 grammar is through Kleene's theorem.)

- Each sub-expression is identifiable with an auxiliary or a terminal symbol. The productions are:
  - $R \rightarrow ST$ if R is a product of sub-expressions S and T
  - $R \rightarrow S$ and $R \rightarrow T$ if R is a union of sub-expressions S and T
  - $R \rightarrow SR$ and $R \rightarrow \Lambda$ if R is S*
  - $R \rightarrow \sigma$ if $\sigma \in \Sigma$
  - $R \rightarrow \Lambda$ if R is Λ
  - none if R is $\varnothing$

Example

- Regular expression: $0((10)^* \cup 01)^*$
  - $R \rightarrow ST$ // $R = 0((10)^* \cup 01)^* = ST$
  - $S \rightarrow 0$ // $S = 0$
  - $T \rightarrow VT$ // $T = ((10)^* \cup 01)^* = V^*$
  - $T \rightarrow \Lambda$
  - $V \rightarrow W$ // $V = (10)^* \cup 01 = W \cup X$
  - $V \rightarrow X$
  - $W \rightarrow YW$ // $W = (10)^* = Y^*$
  - $W \rightarrow \Lambda$
  - $Y \rightarrow 10$ // $Y = 10$
  - $X \rightarrow 01$ // $X = 01$
- Note the connection with language equations.

There are languages that are type 1 but not type 2.

- $\{a^k b^k c^k \mid n \in N, n > 0\}$ can be shown to be type 1. However, there is no type 2 grammar that generates it.

- This is due to the ***pumping lemma for context-free languages***.

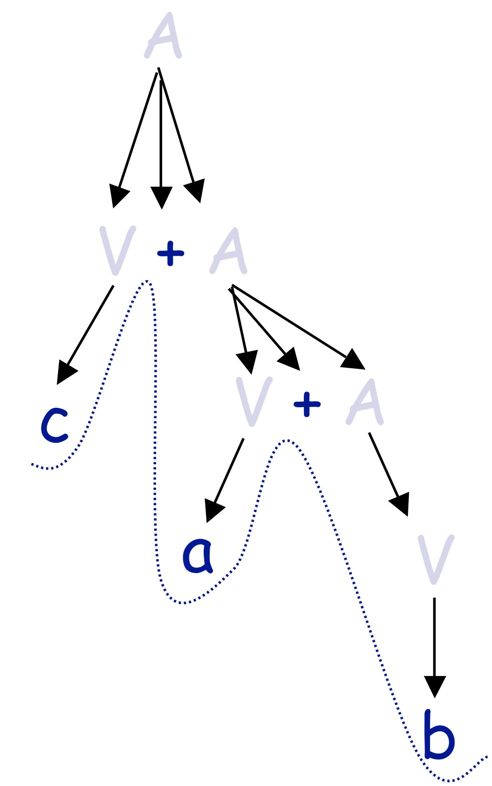
- Before presenting this, we need to review **derivation trees**.

Pumping Lemma for Context-Free Languages

Derivation Tree Visualization

$A \rightarrow V \mid V + A$
$V \rightarrow a \mid b \mid c$

arrows indicate that a production is being applied



Terminal string = "fringe" of tree = "c + a + b"

Derivation Tree Advantage

- The derivation tree has the advantage over linear derviations using $\Rightarrow$.
- Many different derivations can be shown using a single tree.
- These derivations are, in some sense, equivalent.
- Exercise: List all derivations corresponding to the tree on the previous page.

Pumping Lemma for Context-Free Languages

- Let L be a context-free language. Then there is a number n such that

  if $u \in L$ and $|u| > n$ then there are strings v, w, x, y, z, such that

  - $u = vwxyz$
  - $|wy| > 0$ (at least one of w or y is non-empty)
  - $|wxy| \leq n$
  - $(\forall m \geq 0)\ v\, w^m\, x\, y^m\, z \in L$

Proof that $\{a^k b^k c^k \mid k \in N, k > 0\}$ is not context-free using the pumping lemma

- Suppose $\{a^k b^k c^k \mid k \in \omega, k > 0\}$ were context-free. Let n be the integer that exists according to the pumping lemma. Consider $u = a^n b^n c^n$
  and decompose into vwxyz.

- One of w and y is not Λ. Suppose it's w. The other case is symmetric. By the PL, vw^2xy^2z is in L.

- Analyzing the cases for w as to whether it consists of all of one letter or of two letters, in all cases we get a contradiction.

Proof of the CFL Pumping Lemma

- The most direct proof requires a grammar in **Chomsky Normal Form**:

  Every production, with one possible exception*, has one of these two forms:

  $A \rightarrow BC$, where B and C are auxiliaries

  $A \rightarrow \sigma$, where $\sigma \in \Sigma$

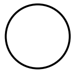
- Every context-free language not containing Λ, is generated by some grammar in Chomsky Normal Form.
- Assume this for now.

Observation

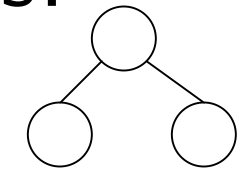
- For a Chomsky Normal Form grammar, the derivation tree is **binary**: each auxiliary node has either:
  - two children, both of which are auxiliary
  - one child, which is terminal

Binary Tree Observation

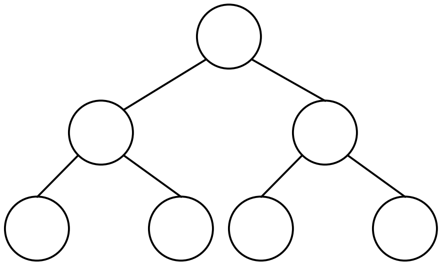
- The **height** of a binary tree is defined as the number of nodes from the root to the longest path.
- A binary tree with height p+1 has at most 2^p leaves.
- A binary tree with at least 2^p leaves has height at least p+1.
- Examples:



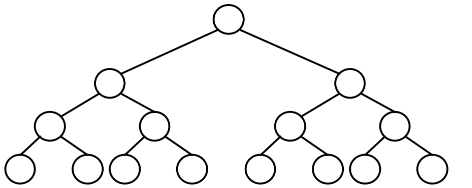
Height 1, 1 leaf



Height 2, 2 leaves



Height 3, 4 leaves



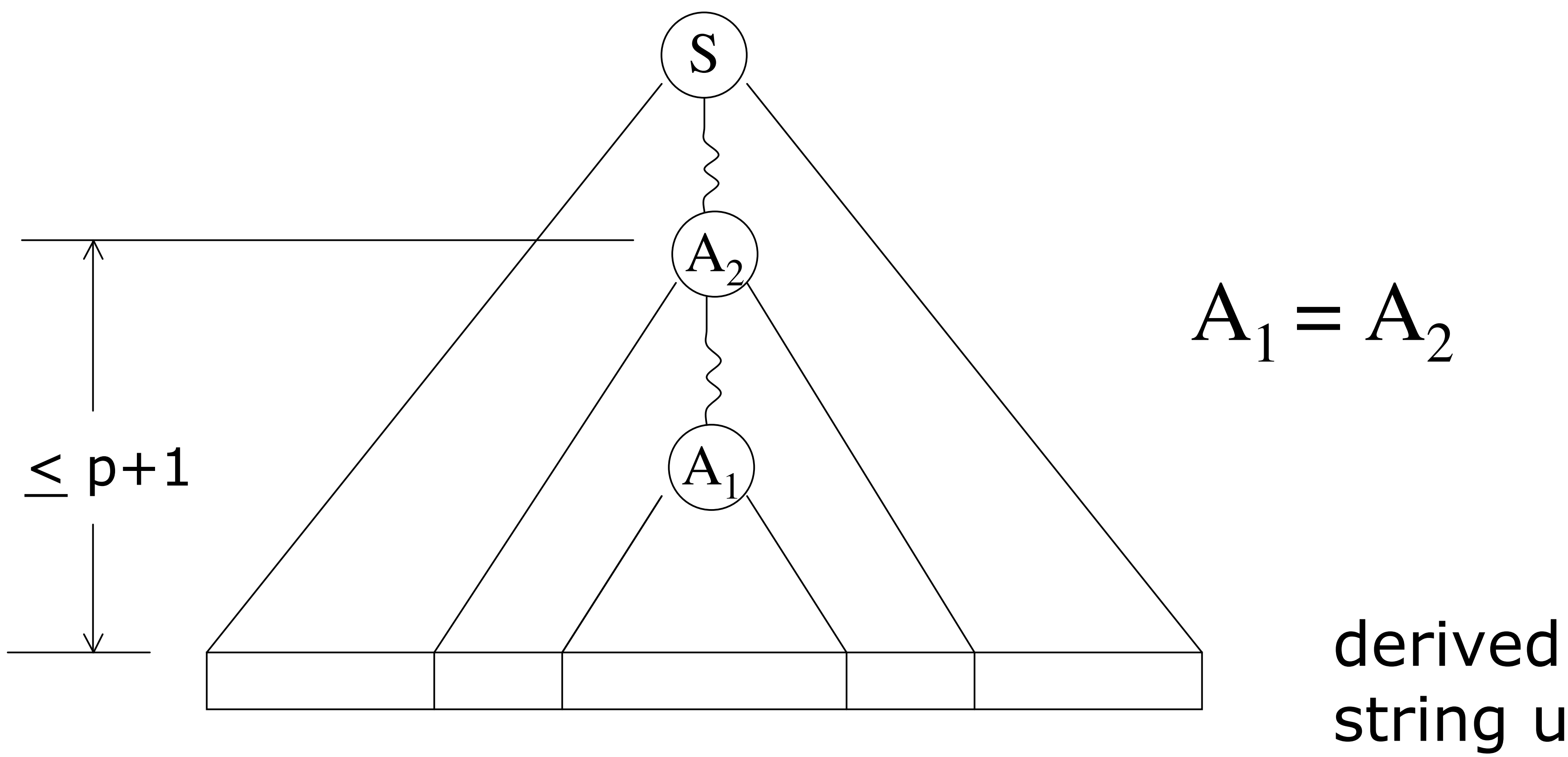
Height 4, 8 leaves

Proof of the Pumping Lemma (1)

- Suppose L is an infinite context-free language, and G is a Chomsky-Normal Form grammar for L.
- **Let p be the number of auxiliary symbols in G, exclusive of the start symbol.**
- We will show that the n that exists in the PL can be satisfied by $\mathbf{n = 2^{p+1}}$.
- Let $u \in L$ be such that $|u| \geq n$. Then the derivation tree for u has at least 2^{p+1} leaves, so the height is at least p+2.
- Consider a maximum length path from leaf to root in this tree. This path has > p+1 auxiliary nodes, therefore some auxiliary must be repeated. Let A_1 be the first instance of a repeated auxiliary on the path and A_2 be the second. Such a repetition must take place in $\leq$ p+1 nodes.

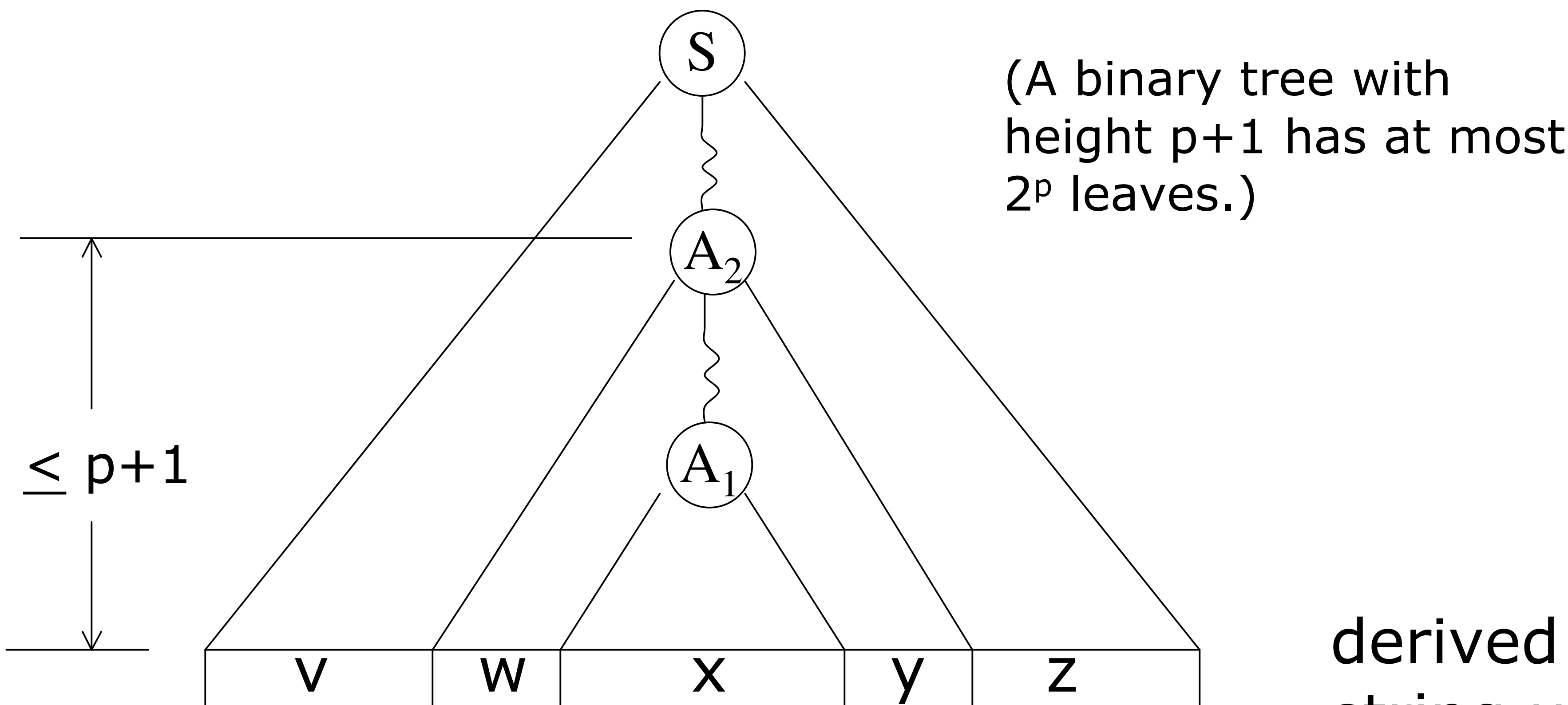
Proof of the Pumping Lemma (2)

- Here is a picture of our derivation tree:



Proof of the Pumping Lemma (3)

- Choose v, w, x, y, z as follows:

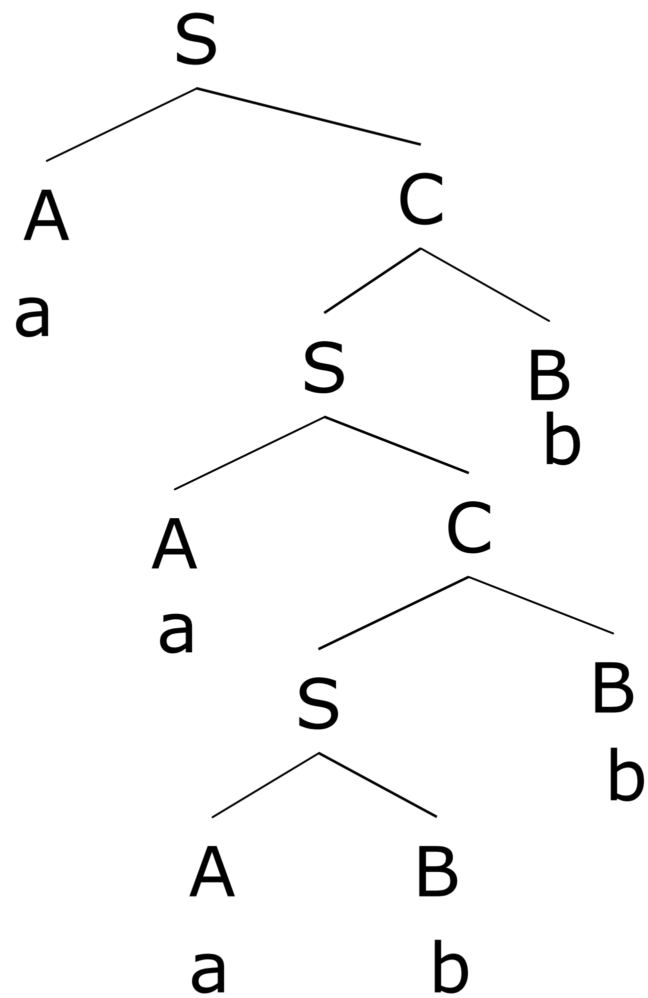


(A binary tree with height p+1 has at most 2^p leaves.)

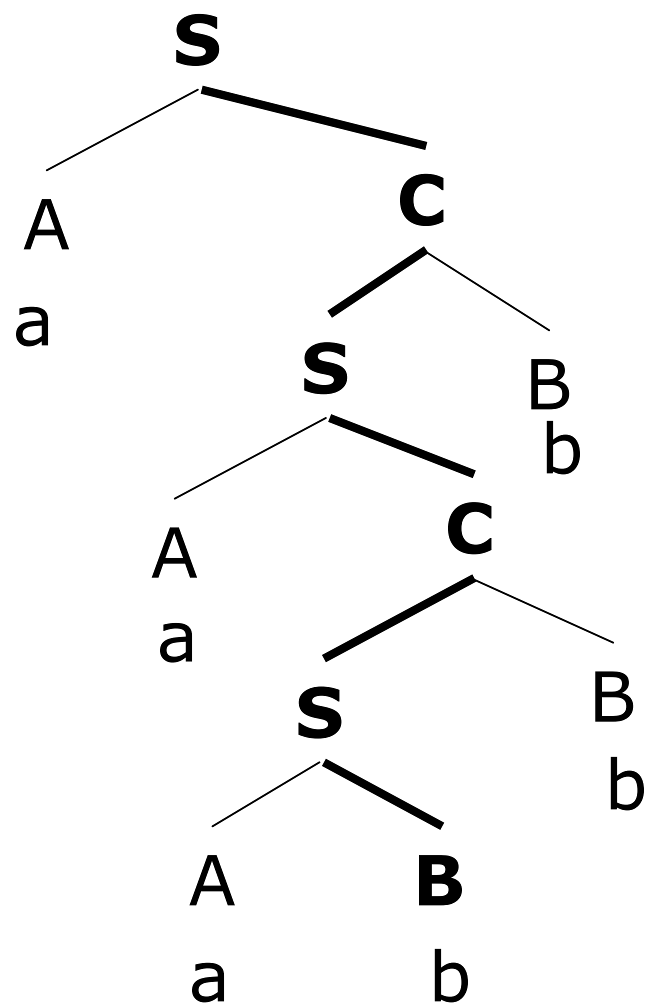
We see that $|wxy| \leq 2^p$. Also, $|wy| > 0$.

Example

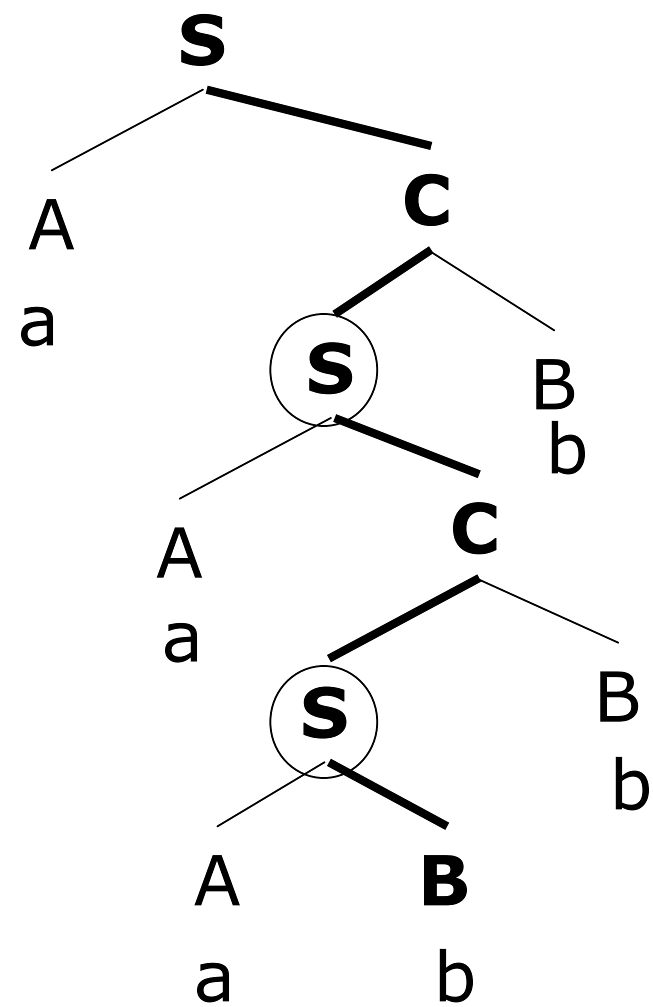
- $S \rightarrow AC$
- $S \rightarrow AB$
- $C \rightarrow SB$
- $A \rightarrow a$
- $B \rightarrow b$
- Derivation tree for aaabbb
- Note: We can illustrate the principle even though this string is not length 16 or longer.



A Long Path

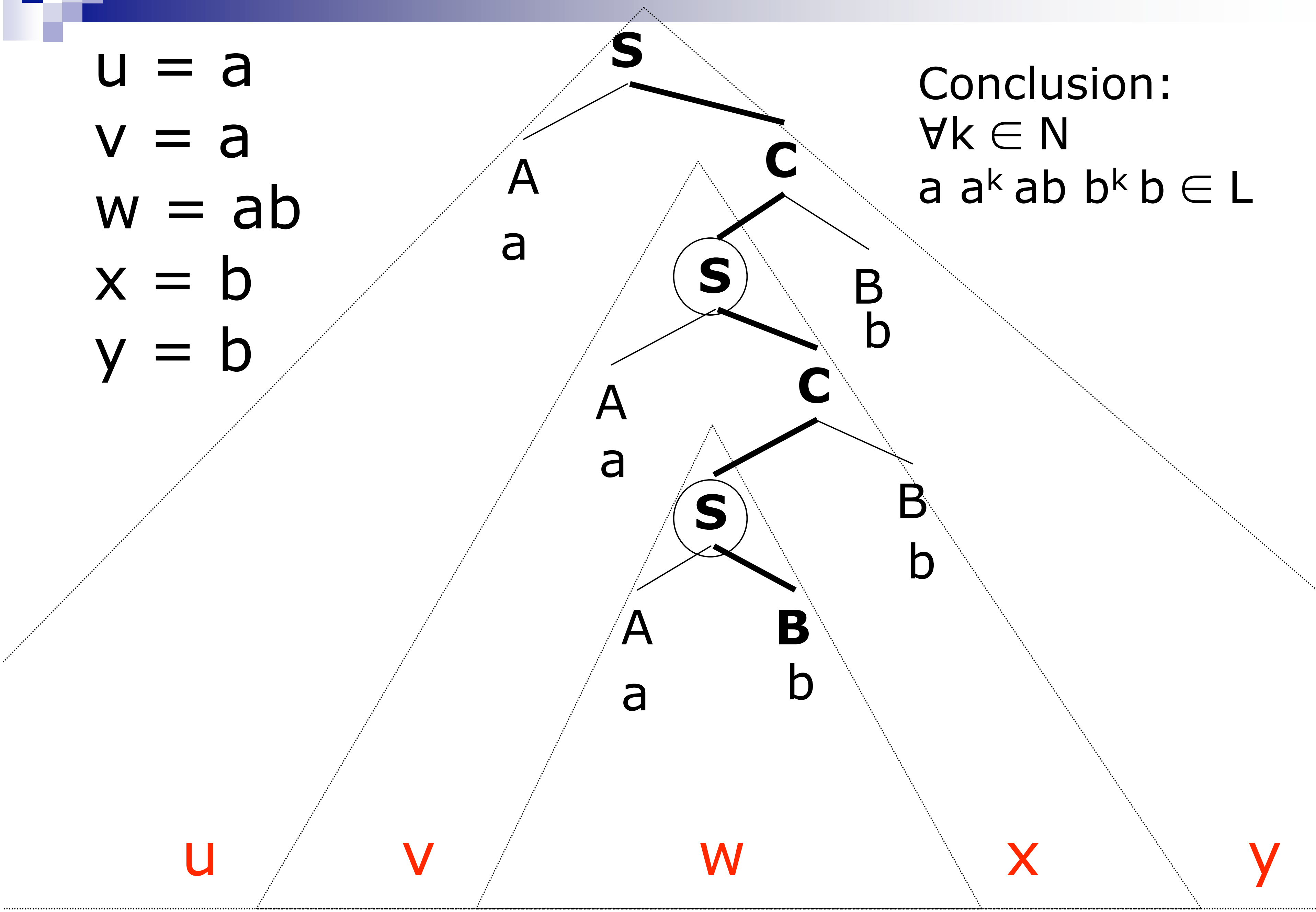


Repeated auxiliaries



$u = a$
$v = a$
$w = ab$
$x = b$
$y = b$

Conclusion:
$\forall k \in N$
$a\ a^k\ ab\ b^k\ b \in L$



Non-Closure Under Intersection

- The context-free languages are not closed under intersection.

- These can be shown to be context-free:
  - $\{a^k b^k c^m \mid k, m \in N\}$
  - $\{a^m b^k c^k \mid k, m \in N\}$

- However, their intersection is:
  - $\{a^k b^k c^k \mid k \in N\}$

  which we know is not context free.

Closure Under Intersection with a Regular Language

- If L is context-free and R is regular, then $L \cap R$ is context-free.

- An easy way to see this is to use a machine characterization of context-free languages, which we discuss subsequently.

Non-Closure Under Complementation

- The context-free languages are not closed under complementation.
- This language can be shown to be not context-free (using the pumping lemma):

  $\{ww \mid w \in \{0, 1\}^*\}$

- However, the complement:

  $\{0, 1\}^* - \{ww \mid w \in \{0, 1\}^*\}$

  is. A grammar for it is given on the next page.

Grammar for $\{0, 1\}^* - \{ww \mid w \in \{0, 1\}^*\}$

$$S \rightarrow AB \mid BA \mid A \mid B$$

$$A \rightarrow CAC \mid 0$$

$$B \rightarrow CBC \mid 1$$

$$C \rightarrow 0 \mid 1$$

- This remains to be shown.

Proof that $\{ww \mid w \in \{0, 1\}^*\}$ is not context free.

- If this language were context free, so would its intersection with a regular language be.
- If we intersect with the regular language $\{0\}^*\{1\}^*\{0\}^*\{1\}^*$, we get a language where all strings are of the form:

  $0^r 1^s 0^r 1^s$

  Let n be the number that exists by the pumping lemma. Select $u = 0^n 1^n 0^n 1^n$ and decompose u in uvwxy where $|vx| > 0$ and $|vwx| \leq n$.
- Show that uv^2wx^2y cannot be of the form $0^r 1^s 0^r 1^s$.