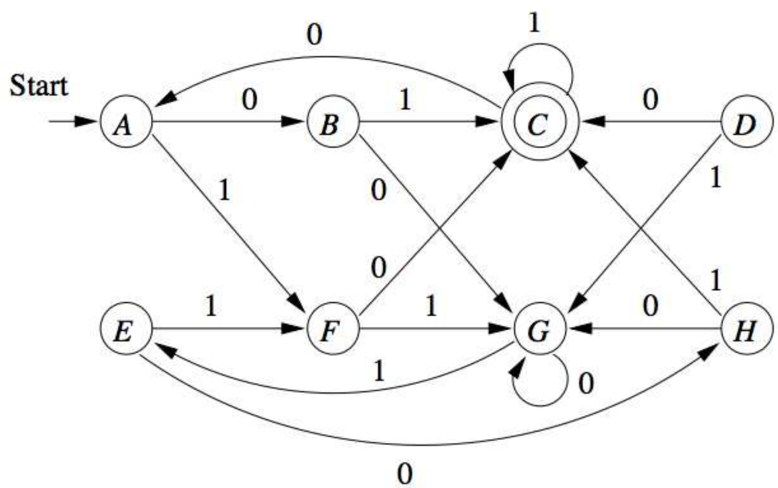
Minimization of DFAs

- "To minimize" a DFA, means to bring the number of DFA states to a minimum.
- Why minimize? To save money and build smaller machines.
- The minimization algorithm is based on the idea of merging all equivalent states. In essence:
  1. Eliminate any state that cannot be reached from a start state.
  2. Partition the remaining states into blocks of equivalent states so that no pair of states from different blocks are equivalent.

Partitioning of states

Partitioning algorithm:

For each state q, construct a block that consists of q and all states equivalent to q.

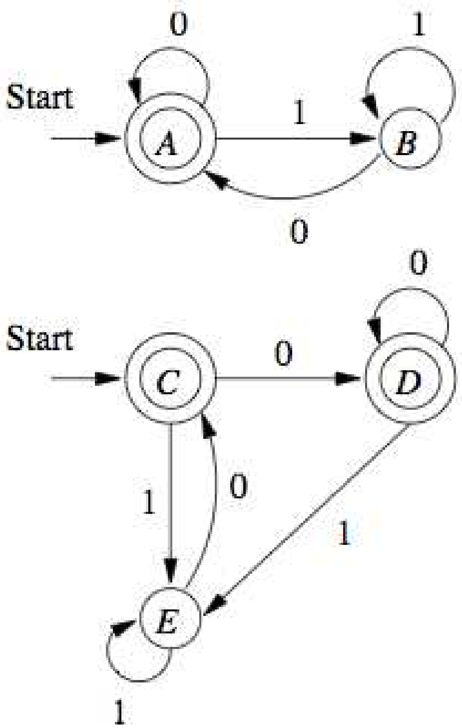


	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x

$$(\{A,E\},\{B,H\},\{D,F\},\underbrace{\{C\},\{G\}})$$

- Notice that C and G are distinguishable from all other states. I.e., each is equivalent only to itself.

Partitioning of states: example



	A	B	C	D
B	x			
C		x		
D		x		
E	x		x	x

$$(\{A, C, D\}, \{B, E\})$$

- Notice: $\{C, D\}$ should not be included, because no pairs of states from different blocks are equivalent.

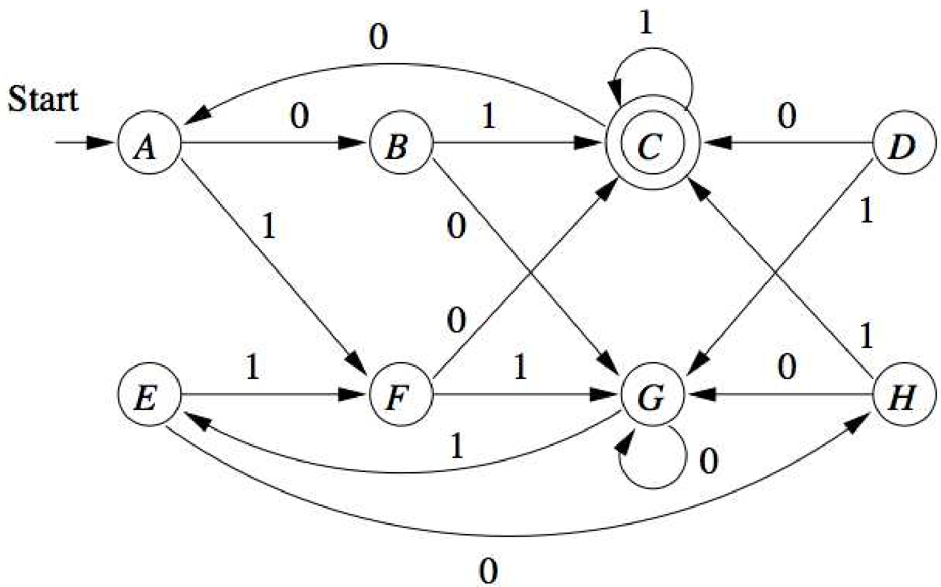
DFA minimization algorithm

Given DFA $A = (Q, \delta_A, \Sigma, q_0, F)$:

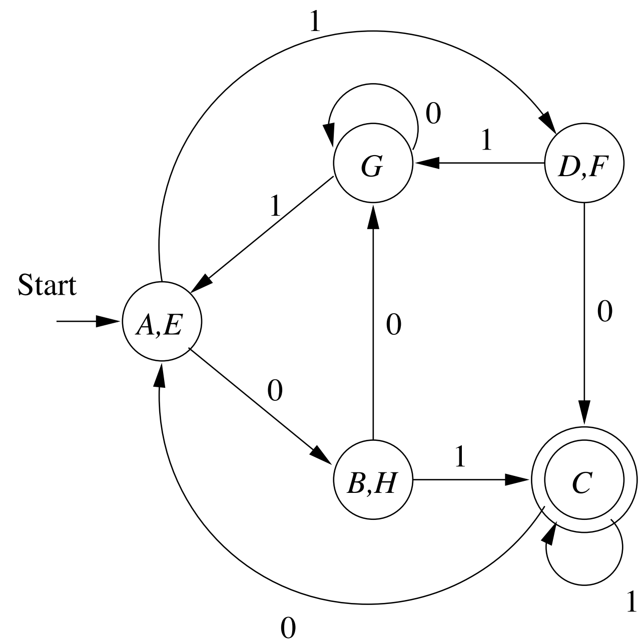
1. Remove unreachable states.
2. Use the TDFA to find all pairs of equivalent states.
3. Partition Q into blocks of mutually equivalent states.
4. Construct the minimum-state equivalent DFA B by using the blocks from step 3, as follows:
   (a) Let S be a block (i.e., state of B), and $a \in \Sigma$. Then the transition function for B is defined as: $\delta_B(S, a) = T$, where T is a block and $\delta_A(p, a) \in T$ for all states p in S.
   (b) The start state of B is the block containing the start state of A.
   (c) The set of accepting states of B is the set of blocks containing accepting states of A.

Example

Minimizing

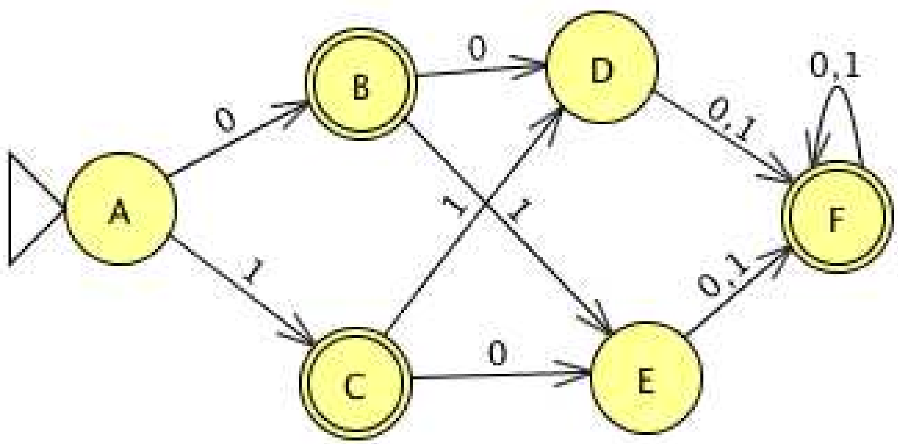


to obtain



Exercise

Draw the table of distinguishabilities and use it to construct the minimum-state DFA for the DFA below



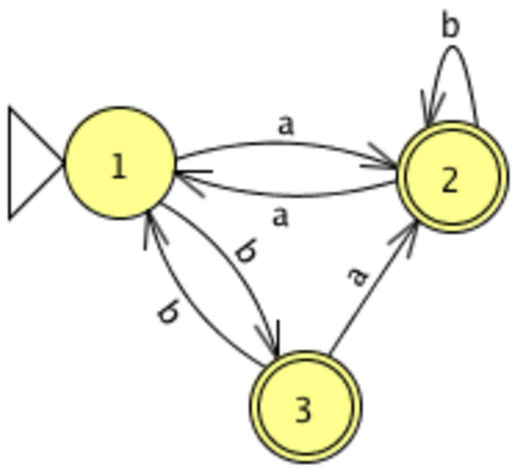
Exercise

Describe the error in the following "proof" that 0^*1^* is not a regular language.
Steps of the "proof":

1. Assume that 0^*1^* is regular.
2. Lep p be the pumping length for 0^*1^*.
3. Choose w to be the string 0^p1^p.
4. You know that w is a member of 0^*1^*.
5. But from an example shown in class we know that w cannot be pumped. Thus you have a contradiction.
6. So 0^*1^* is not regular.

Exercise

Convert the following DFA to a RE using the state elimination-based algorithm.



Context-Free Grammars and Languages

- We have studied Regular Languages and its descriptors: finite automata and regular expressions.
  - we learned that even some simple languages are not regular, such as $\{(^n)^n, n \geq 0\}$.
  - What kind of language are those? What descriptors do they have?
- Today we will start studying context-free grammars (CFG)
  - they are useful in a variety of applications, e.g., compilers, translators, specification of markup languages, etc.
- CFGs define context-free languages (CFL), which includes regular languages.

CFG: informal example

Let G_1 be the following grammar. Let's see some new terminology:

$$A \rightarrow (A)$$
$$A \rightarrow \epsilon$$

- A grammar is a collection of substitution (production) rules.
- A rule consists of a variable, an ' $\rightarrow$ ' symbol, and a sequence of symbols consisting of variables and terminals.
- The start variable occurs on the Left-Hand-Sside of the topmost rule.
- For convenience, we may write the rules above as $A \rightarrow (A) \mid \epsilon$

Using a grammar to infer a string

Inferring a string via a derivation:

1. Write down the start variable.
2. Find a variable that is written down and a rule that has that variable in the LHS, then "expand" that variable by replacing it with the RHS of the rule.
3. Repeat step 2 until no variables remain.

Example: a derivation of string $((()))$ using the grammar

$$A \rightarrow (A) \mid \epsilon$$

$$A \Rightarrow (A) \Rightarrow ((A)) \Rightarrow (((A))) \Rightarrow ((()))$$

Notice: ' $\Rightarrow$ ' denotes a derivation step.

Formal Definition of CFG

A CFG G is a 4-tuple $G = (V, \Sigma, R, S)$, where:

- V is a finite set called the variables.
- Σ is a finite set, disjoint from V, called the terminals,
- R is a finite set of rules, with each rule being a variable and a string of variables and terminals, and
- S is the start symbol.

If u, v and w are strings of variables and terminals, and $A \to w$ is a rule of the grammar, we say that

- uAv *yields* uwv, denoted $uAv \Rightarrow uwv$
- $u \stackrel{*}{\Rightarrow} v$ if $u = v$ or $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$

Example of derivation

Let G be the following grammar $G = (\{E, I\}, T, P, E)$, where, $T = \{+, *, (,), a, b, 0, 1\}$ and P is the following set of productions:

$$E \to I \mid E + E \mid E * E \mid (E)$$
$$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

Derivation of $a * (a + b00)$:

$$E \Rightarrow E * E \Rightarrow I * E \Rightarrow a * E \Rightarrow a * (E) \Rightarrow$$

$$a*(E+E) \Rightarrow a*(I+E) \Rightarrow a*(a+E) \Rightarrow a*(a+I) \Rightarrow$$

$$a * (a + I0) \Rightarrow a * (a + I00) \Rightarrow a * (a + b00)$$

Note: not all choices lead to successful derivations of a particular string, e.g.: $E \Rightarrow E + E$

Leftmost and Rightmost Derivations

- *Leftmost derivation* $\underset{lm}{\Rightarrow}$: Always replace the leftmost variable by one of its rule-bodies.
- *Rightmost derivation* $\underset{rm}{\Rightarrow}$: Always replace the rightmost variable by one of its rule-bodies.

Example: rightmost derivation of $a * (a + b00)$

$$
\begin{aligned}
E \rightarrow \quad & I \mid E + E \mid \\
& E * E \mid (E) \\
I \rightarrow \quad & a \mid b \mid Ia \mid \\
& Ib \mid I0 \mid I1
\end{aligned}
$$

$$
\begin{gathered}
E \underset{rm}{\Rightarrow} E * E \underset{rm}{\Rightarrow} \\
E*(E) \underset{rm}{\Rightarrow} E*(E+E) \underset{rm}{\Rightarrow} E*(E+I) \underset{rm}{\Rightarrow} E*(E+I0) \\
\underset{rm}{\Rightarrow} E*(E+I00) \underset{rm}{\Rightarrow} E*(E+b00) \underset{rm}{\Rightarrow} E*(I+b00) \\
\underset{rm}{\Rightarrow} E*(a+b00) \underset{rm}{\Rightarrow} I*(a+b00) \underset{rm}{\Rightarrow} a*(a+b00)
\end{gathered}
$$

The language of a Grammar

- The language of a grammar G with start symbol S is the set of strings of terminals that have derivations from S, i.e.:

$$L(G) = \{w \in \Sigma^* : S \overset{*}{\Rightarrow} w\}$$

- If G is a CFG, then we call $L(G)$ a context-free language.

Designing CFGs

The structure of strings in a CFL can be of two basic kinds:

- a string with multiple regions that must occur in some fixed order but do not have any correspondence between each other, e.g.. $a^*(b^* + c^*)$. In that case, to generate such string use a rule of the form

$$A \to BC \cdots$$

- a string with two regions that must occur in some fixed order and must correspond to each other, e.g., $\{a^n b^n : n \geq 1\}$. In that case, to generate such string start at the outside edges of the string and generate toward the middle.

Exercises

Design CFG for the following languages:

- a^*
- $a^*b(ba^*b + a)^*$
- $\{a^n b^n : n \geq 1\}$
- $\{a^i b^j : i \leq j\}$
- $\{a^i b^j : i \neq j\}$