Topic 10: Finite State Machines
The story so far...

\[ \text{REs} \subseteq \text{CFGs} \]

\[ \text{DFAs} = \text{NFAs} \]
We have shown how to build an optimal DFA for every regular expression:
- Build NFA
- Convert NFA to DFA using subset construction
- Minimize resulting DFA

Thus, we could now implement a RegExp library:
- most RegExp libraries actually simulate the NFA
- (even better: one can combine the two approaches: apply DFA minimization lazily while simulating the NFA)
The story so far...

\[
\text{REs} \subseteq \text{CFGs} \subseteq \text{DFAs} = \text{NFAs}
\]

Is this \(\subseteq\) really “=” or “\(\not\subseteq\)”?
Regular expressions \equiv NFAs \equiv DFAs

**Theorem:** For any NFA, there is a regular expression that accepts the same language

**Corollary:** A language is recognized by a DFA (or NFA) if and only if it has a regular expression

You need to know these facts

– the construction for the Theorem is included in the slides after this, but you will not be tested on it
New Machinery: Generalized NFAs

• Like NFAs but allow
  – parallel edges
  – regular expressions as edge labels
    NFAs already have edges labeled $\varepsilon$ or $a$

• The label of a path is now the concatenation of the regular expressions on those edges, making it a regular expression

• Def: A string $x$ is accepted by a generalized NFA iff there is a path from start to final state labeled by a regular expression whose language $\text{contains } x$
Construction Idea

Add new start state and final state

Then delete the original states one by one, adding edges to keep the same language, until the graph looks like:
Starting from an NFA

Then delete the original states one by one, adding edges to **keep the same language**, until the graph looks like:

Final graph has only one path to the accepting state, which is labeled by A, so it accepts iff x is in the language of A.

Thus, A is a regular expression with the same language as the original NFA.
Only two simplification rules

- **Rule 1**: For any two states $q_1$ and $q_2$ with parallel edges (possibly $q_1=q_2$), replace

  If the machine would have used the edge labeled $A$ by consuming an input $x$ in the language of $A$, it can instead use the edge labeled $A \cup B$.

  Furthermore, this new edge does not allow transitions for any strings other than those that matched $A$ or $B$.

  ![Diagram](attachment:diagram.png)
Only two simplification rules

- **Rule 2**: Eliminate non-start/accepting state \( q_3 \) by creating direct edges that skip \( q_3 \)

for every pair of states \( q_1, q_2 \) (even if \( q_1 = q_2 \))

Any path from \( q_1 \) to \( q_2 \) would have to match \( AB^nC \) for some \( n \) (the number of times the self loop was used), so the machine can use the new edge instead. New edge *only* allows strings that were allowed before.
While the box contains some state \( s \): for all states \( r, t \) with \( (r, s) \) and \( (s, t) \) in \( E \): create a direct edge \( (r, t) \) by Rule 2 delete \( s \) (no longer needed) merge all parallel edges by Rule 1

When the loop exits, the graph looks like this:

\[ \begin{align*}
A
\end{align*} \]

\( A \) is a regular expression with the same language as the original NFA.
Converting an NFA to a regular expression

Consider the DFA for the mod 3 sum

– Accept strings from \{0,1,2\}^* where the digits mod 3 sum of the digits is 0
Splicing out a state $t_1$

Create direct edges between neighbors of $t_1$ (so that we can delete it afterward)
Splicing out a state $t_1$

Regular expressions to add to edges

$t_0 \rightarrow t_1 \rightarrow t_0 : 10^*2$
$t_0 \rightarrow t_1 \rightarrow t_2 : 10^*1$
$t_2 \rightarrow t_1 \rightarrow t_0 : 20^*2$
$t_2 \rightarrow t_1 \rightarrow t_2 : 20^*1$
Splicing out a state $t_1$

Delete $t_1$ now that it is redundant

$t_0 \rightarrow t_1 \rightarrow t_0 : 10*2$
$t_0 \rightarrow t_1 \rightarrow t_2 : 10*1$
$t_2 \rightarrow t_1 \rightarrow t_0 : 20*2$
$t_2 \rightarrow t_1 \rightarrow t_2 : 20*1$
Splicing out a state $t_1$

Create direct edges between neighbors of $t_2$ (so that we can delete it afterward)
Splicing out a state $t_1$

Regular expressions to add to edges

$R_1$: $0 \cup 10^2$

$R_2$: $2 \cup 10^1$

$R_3$: $1 \cup 20^2$

$R_4$: $0 \cup 20^1$
Splicing out state \( t_2 \) (and then \( t_0 \))

Delete \( t_2 \) now that it is redundant

\[
\begin{align*}
R_1 & : \ 0 \cup 10^*2 \\
R_2 & : \ 2 \cup 10^*1 \\
R_3 & : \ 1 \cup 20^*2 \\
R_4 & : \ 0 \cup 20^*1 \\
R_5 & : \ R_1 \cup R_2 R_4^* R_3
\end{align*}
\]
Splicing out state $t_2$ (and then $t_0$)

Create direct (s,f) edge so we can delete $t_0$

$R_1: \ 0 \cup 10*2$
$R_2: \ 2 \cup 10*1$
$R_3: \ 1 \cup 20*2$
$R_4: \ 0 \cup 20*1$
$R_5: \ R_1 \cup R_2 R_4 * R_3$
Splicing out state $t_2$ (and then $t_0$)

Regular expressions to add to edges

$R_1$: $0 \cup 10^*2$
$R_2$: $2 \cup 10^*1$
$R_3$: $1 \cup 20^*2$
$R_4$: $0 \cup 20^*1$
$R_5$: $R_1 \cup R_2 R_4^* R_3$

$t_0 \rightarrow t_1 \rightarrow t_0 : R_5^*$
Splicing out state $t_2$ (and then $t_0$)

Delete $t_0$ now that it is redundant

\[
\begin{align*}
R_1 & : 0 \cup 10^*2 \\
R_2 & : 2 \cup 10^*1 \\
R_3 & : 1 \cup 20^*2 \\
R_4 & : 0 \cup 20^*1 \\
R_5 & : R_1 \cup R_2R_4^*R_3 \\
R_6 & : R_5^*
\end{align*}
\]
Splicing out state \( t_2 \) (and then \( t_0 \))

**Regular expressions to add to edges**

\[
\begin{align*}
R_1 & : 0 \cup 10^*2 \\
R_2 & : 2 \cup 10^*1 \\
R_3 & : 1 \cup 20^*2 \\
R_4 & : 0 \cup 20^*1 \\
R_5 & : R_1 \cup R_2 R_4^* R_3 \\
R_6 & : R_5^* \\
\end{align*}
\]

**Final regular expression:**

\[
R_6 = (0 \cup 10^*2 \cup (2 \cup 10^*1)(0 \cup 20^*1)^*(1 \cup 20^*2))^*
\]
The story so far...

Languages represented by DFA, NFA, or regular expressions are called **Regular Languages**.
Recall: Algorithms for Regular Languages

We have seen algorithms for
• RE to NFA
• NFA to DFA
• DFA/NFA to RE
• DFA minimization
Example Corollary of These Results

**Corollary:** If $A$ is the language of a regular expression, then $\overline{A}$ is the language of a regular expression*.

(This is the complement with respect to the universe of all strings over the alphabet, i.e., $\overline{A} = \Sigma^* \setminus A$.)
The story so far...

Now: Is this $\subseteq$ really “=” or “$\in$”? 

- REs $\subseteq$ CFGs
- DFAs $=$ NFAs
What languages have DFAs? CFGs?

All of them?
Languages and Representations!

- All
- Context-Free
- Regular
- Finite
- $\{001, 10, 12\}$
- $0^*$
- DFA
- NFA
- Regex
Languages and Representations!

Reminder: All finite languages are regular.
DFAs Recognize Any Finite Language

Construct a DFA for each string in the language.

Then, put them together using the union construction.
Languages and Machines!

*All

Context-Free

Regular

\(0^*\)

Finite

\(\{001, 10, 12\}\)

DFA

NFA

Regex

Warmup 2: Surprising example here
An Interesting Infinite Regular Language

$L = \{x \in \{0, 1\}^*: x \text{ has an equal number of substrings } 01 \text{ and } 10\}.$

$L$ is infinite.

$0, 00, 000, ...$

$L$ is regular. How could this be?

That seems to require comparing counts...

- easy for a CFG
- but seems hard for DFAs!
An Interesting Infinite Regular Language

\[ L = \{ x \in \{0, 1\}^* : x \text{ has an equal number of substrings 01 and 10} \}. \]

\( L \) is infinite.

\[ 0, 00, 000, \ldots \]

\( L \) is regular. How could this be? It is just the set of binary strings that are empty or begin and end with the same character!

![NFA Diagram](image)
Languages and Representations!

Main Event: Prove there is a context-free language that isn’t regular.

{001, 10, 12}
Tangent: How to prove a DFA minimal?

• Show there is no smaller DFA...

• Find a set of strings that *must* be distinguished
  – Such a set is a lower bound on the DFA size
Recall: Binary strings with a 1 in the 3\textsuperscript{rd} position from the start.

Distinguishing set:

\{\varepsilon, 0, 00, 000, 001\}
The language of “Binary Palindromes” is Context-Free

\[ S \rightarrow \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1 \]
Intuition (NOT A PROOF!):

**Q:** What would a DFA need to keep track of to decide?

**A:** It would need to keep track of the “first part” of the input in order to check the second part against it...

...but there are an infinite # of possible first parts and we only have finitely many states.

Proof idea: any machine that does not remember the entire first half will be wrong for some inputs.
Useful Lemmas about DFAs

**Lemma 1:** If DFA $M$ takes $x, y \in \Sigma^*$ to the same state, then for every $z \in \Sigma^*$, $M$ accepts $x \cdot z$ iff it accepts $y \cdot z$.

$M$ can’t remember that the input was $x$, not $y$.

$x \cdot z = x_1 x_2 \ldots x_n z_1 z_2 \ldots z_k$

$y \cdot z = y_1 y_2 \ldots y_m z_1 z_2 \ldots z_k$
Lemma 2: If DFA $M$ has $n$ states and a set $S$ contains more than $n$ strings, then $M$ takes at least two strings from $S$ to the same state.

$M$ can’t take $n+1$ or more strings to different states because it doesn’t have $n+1$ different states. So, some pair of strings must go to the same state.
$B = \{\text{binary palindromes}\}$ can’t be recognized by any DFA $M$.

Suppose for contradiction that some DFA, $M$, recognizes $B$. We will show $M$ accepts or rejects a string it shouldn’t.

Consider $S = \{1, 01, 001, 0001, 00001, \ldots\} = \{0^n1 : n \geq 0\}$. 
\[ B = \{ \text{binary palindromes} \} \text{ can't be recognized by any DFA} \]

Suppose for contradiction that some DFA, \( M \), accepts \( B \). We will show \( M \) accepts or rejects a string it shouldn't.

Consider \( S = \{1, 01, 001, 0001, 00001, \ldots \} = \{0^n1 : n \geq 0\} \). Since there are finitely many states in \( M \) and infinitely many strings in \( S \), by Lemma 2, there exist strings \( 0^a1 \in S \) and \( 0^b1 \in S \) with \( a \neq b \) that end in the same state of \( M \).

**SUPER IMPORTANT POINT:** You do not get to choose what \( a \) and \( b \) are. Remember, we’ve just proven they exist...we must take the ones we’re given!
$B = \{\text{binary palindromes}\}$ can’t be recognized by any DFA.

Suppose for contradiction that some DFA, $M$, accepts $B$. We will show $M$ accepts or rejects a string it shouldn’t.

Consider $S = \{1, 01, 001, 0001, 00001, \ldots\} = \{0^n1 : n \geq 0\}$.

Since there are finitely many states in $M$ and infinitely many strings in $S$, by Lemma 2, there exist strings $0^a1 \in S$ and $0^b1 \in S$ with $a \neq b$ that end in the same state of $M$.

Now, consider appending $0^a$ to both strings.
B = \{binary palindromes\} can’t be recognized by any DFA

Suppose for contradiction that some DFA, \(M\), accepts B. We will show \(M\) accepts or rejects a string it shouldn’t.

Consider \(S = \{1, 01, 001, 0001, 00001, \ldots\} = \{0^n1 : n \geq 0\}\).

Since there are finitely many states in \(M\) and infinitely many strings in \(S\), by Lemma 2, there exist strings \(0^a1 \in S\) and \(0^b1 \in S\) with \(a \neq b\) that end in the same state of \(M\).

Now, consider appending \(0^a\) to both strings.

Since \(0^a1\) and \(0^b1\) end in the same state, \(0^a10^a\) and \(0^b10^a\) also end in the same state, call it \(q\). But then \(M\) makes a mistake: \(q\) needs to be an accept state since \(0^a10^a \in B\), but \(M\) would accept \(0^b10^a \notin B\), which is an error.
$B = \{\text{binary palindromes}\}$ can’t be recognized by any DFA

Suppose for contradiction that some DFA, $M$, accepts $B$. We will show $M$ accepts or rejects a string it shouldn’t.

Consider $S = \{1, 01, 001, 0001, 00001, \ldots\} = \{0^n1 : n \geq 0\}$.

Since there are finitely many states in $M$ and infinitely many strings in $S$, by Lemma 2, there exist strings $0^a1 \in S$ and $0^b1 \in S$ with $a \neq b$ that end in the same state of $M$.

Now, consider appending $0^a$ to both strings. Since $0^a1$ and $0^b1$ end in the same state, $0^a10^a$ and $0^b10^a$ also end in the same state, call it $q$. But then $M$ makes a mistake: $q$ needs to be an accept state since $0^a10^a \in B$, but $M$ would accept $0^b10^a \notin B$, which is an error.

This proves that $M$ does not recognize $B$, contradicting our assumption that it does. Thus, no DFA recognizes $B$. 

Showing that a Language \( L \) is not regular

1. “Suppose for contradiction that some DFA \( M \) recognizes \( L \).”
2. Consider an **INFINITE** set \( S \) of prefixes (which we intend to complete later).
3. “Since \( S \) is infinite and \( M \) has finitely many states, there must be two strings \( s_a \) and \( s_b \) in \( S \) for \( s_a \neq s_b \) that end up at the same state of \( M \).”
4. Consider appending the (correct) completion \( t \) to each of the two strings.
5. “Since \( s_a \) and \( s_b \) both end up at the same state of \( M \), and we appended the same string \( t \), both \( s_a t \) and \( s_b t \) end at the same state \( q \) of \( M \). Since \( s_a t \in L \) and \( s_b t \notin L \), \( M \) does not recognize \( L \).”
6. “Thus, no DFA recognizes \( L \).”
Showing that a Language $L$ is not regular

The choice of $S$ is the creative part of the proof

You must find an infinite set $S$ with the property that no two strings can be taken to the same state

- i.e., for every pair of strings $S$ there is an “accept” completion that the two strings DO NOT SHARE
Prove $A = \{0^n1^n : n \geq 0\}$ is not regular

Suppose for contradiction that some DFA, $M$, recognizes $A$.

Let $S =$
Prove $A = \{0^n1^n : n \geq 0\}$ is not regular

Suppose for contradiction that some DFA, $M$, recognizes $A$.

Let $S = \{0^n : n \geq 0\}$. Since $S$ is infinite and $M$ has finitely many states, there must be two strings, $0^a$ and $0^b$ for some $a \neq b$ that end in the same state in $M$. 
Prove $A = \{0^n1^n : n \geq 0\}$ is not regular

Suppose for contradiction that some DFA, $M$, recognizes $A$.

Let $S = \{0^n : n \geq 0\}$. Since $S$ is infinite and $M$ has finitely many states, there must be two strings, $0^a$ and $0^b$ for some $a \neq b$ that end in the same state in $M$.

Consider appending $1^a$ to both strings.
Prove \( A = \{0^n1^n : n \geq 0\} \) is not regular

Suppose for contradiction that some DFA, \( M \), recognizes \( A \).

Let \( S = \{0^n : n \geq 0\} \). Since \( S \) is infinite and \( M \) has finitely many states, there must be two strings, \( 0^a \) and \( 0^b \) for some \( a \neq b \) that end in the same state in \( M \).

Consider appending \( 1^a \) to both strings.

Note that \( 0^a1^a \in A \), but \( 0^b1^a \notin A \) since \( a \neq b \). But they both end up in the same state of \( M \), call it \( q \). Since \( 0^a1^a \in A \), state \( q \) must be an accept state but then \( M \) would incorrectly accept \( 0^b1^a \notin A \) so \( M \) does not recognize \( A \). Thus, no DFA recognizes \( A \).
Prove $P = \{\text{balanced parentheses}\}$ is not regular

Suppose for contradiction that some DFA, $M$, accepts $P$.

Let $S =$
Prove $P = \{\text{balanced parentheses}\}$ is not regular

Suppose for contradiction that some DFA, $M$, recognizes $P$.

Let $S = \{ (n : n \geq 0) \}$. Since $S$ is infinite and $M$ has finitely many states, there must be two strings, $a^n$ and $b^n$ for some $a \neq b$ that end in the same state in $M$. 
Prove \( P = \{ \text{balanced parentheses} \} \) is not regular

Suppose for contradiction that some DFA, \( M \), recognizes \( P \).

Let \( S = \{ n^n : n \geq 0 \} \). Since \( S \) is infinite and \( M \) has finitely many states, there must be two strings, \( a^n \) and \( b^n \) for some \( a \neq b \) that end in the same state in \( M \).

Consider appending \( \,^a \) to both strings.
Prove $P = \{\text{balanced parentheses}\}$ is not regular

Suppose for contradiction that some DFA, $M$, recognizes $P$.

Let $S = \{ (n : n \geq 0) \}$. Since $S$ is infinite and $M$ has finitely many states, there must be two strings, $(a)$ and $(b)$ for some $a \neq b$ that end in the same state in $M$.

Consider appending $)a$ to both strings.

Note that $(a)^a \in P$, but $(b)^a \notin P$ since $a \neq b$. But they both end up in the same state of $M$, call it $q$. Since $(a)^a \in P$, state $q$ must be an accept state but then $M$ would incorrectly accept $(b)^a \notin P$ so $M$ does not recognize $P$. Thus, no DFA recognizes $P$. 
Showing that a Language \( L \) is not regular

1. “Suppose for contradiction that some DFA \( M \) recognizes \( L \).”

2. Consider an **INFINITE** set \( S \) of prefixes (which we intend to complete later). It is imperative that for **every pair** of strings in our set there is an “accept” completion that the two strings **DO NOT SHARE**.

3. “Since \( S \) is infinite and \( M \) has finitely many states, there must be two strings \( s_a \) and \( s_b \) in \( S \) for \( s_a \neq s_b \) that end up at the same state of \( M \).”

4. Consider appending the (correct) completion \( t \) to each of the two strings.

5. “Since \( s_a \) and \( s_b \) both end up at the same state of \( M \), and we appended the same string \( t \), both \( s_a t \) and \( s_b t \) end at the same state \( q \) of \( M \). Since \( s_a t \in L \) and \( s_b t \not\in L \), \( M \) does not recognize \( L \).”

6. “Thus, no DFA recognizes \( L \).”
Fact: This method is optimal

- Suppose that for a language \( L \), the set \( S \) is a *largest* set of prefixes with the property that, for every pair \( s_a \neq s_b \in S \), there is some string \( t \) such that one of \( s_a t, s_b t \) is in \( L \) but the other isn’t.
- If \( S \) is infinite, then \( L \) is not regular
- If \( S \) is finite, then the minimal DFA for \( L \) has precisely \( |S| \) states, one reached by each member of \( S \).
Fact: This method is optimal

- Suppose that for a language $L$, the set $S$ is a largest set of prefixes with the property that, for every pair $s_a \neq s_b \in S$, there is some string $t$ such that one of $s_a t, s_b t$ is in $L$ but the other isn’t.
- If $S$ is infinite, then $L$ is not regular.
- If $S$ is finite, then the minimal DFA for $L$ has precisely $|S|$ states, one reached by each member of $S$.

Corollary: Our minimization algorithm was correct.
- we separated exactly those states for which some $t$ would make one accept and another not accept.
Important Notes

• It is not necessary for our strings $xz$ with $x \in L$ to allow any string in the language
  – we only need to find a small “core” set of strings that must be distinguished by the machine

• It is not true that, if $L$ is irregular and $L \subseteq U$, then $U$ is irregular!
  – we always have $L \subseteq \Sigma^*$ and $\Sigma^*$ is regular!
  – our argument needs different answers: $xz \in L \iff yz \in L$
    for $\Sigma^*$, both strings are always in the language

Do not claim in your proof that, because $L \subseteq U$, $U$ is also irregular
New Machinery: Generalized NFAs

• Like NFAs but allow
  – parallel edges (between the same pair of states)
  – regular expressions as edge labels
    NFAs already have edges labeled $\varepsilon$ or $a$

• Machine can follow an edge labeled by $A$ by reading a string of input characters in the language of $A$
  – (if $A$ is $a$ or $\varepsilon$, this matches the original definition, but we now allow REs built with recursive steps.)
New Machinery: Generalized NFAs

• Like NFAs but allow
  – parallel edges
  – regular expressions as edge labels
    NFAs already have edges labeled $\varepsilon$ or $a$

• The label of a path is now the concatenation of the regular expressions on those edges, making it a regular expression

• Def: A string $x$ is accepted by a generalized NFA iff there is a path from start to final state labeled by a regular expression whose language contains $x$
Construction Idea

Add new start state and final state

Then delete the original states one by one, adding edges to keep the same language, until the graph looks like:
Starting from an NFA

Then delete the original states one by one, adding edges to keep the same language, until the graph looks like:

Final graph has only one path to the accepting state, which is labeled by $A$, so it accepts iff $x$ is in the language of $A$.

Thus, $A$ is a regular expression with the same language as the original NFA.
Only two simplification rules

- **Rule 1**: For any two states $q_1$ and $q_2$ with parallel edges (possibly $q_1 = q_2$), replace

  If the machine would have used the edge labeled $A$ by consuming an input $x$ in the language of $A$, it can instead use the edge labeled $A \cup B$.

  Furthermore, this new edge does not allow transitions for any strings other than those that matched $A$ or $B$.

Furthermore, this new edge does not allow transitions for any strings other than those that matched $A$ or $B$. 
Only two simplification rules

- Rule 2: Eliminate non-start/accepting state $q_3$ by creating direct edges that skip $q_3$

for every pair of states $q_1, q_2$ (even if $q_1 = q_2$)

Any path from $q_1$ to $q_2$ would have to match $AB^nC$ for some $n$ (the number of times the self loop was used), so the machine can use the new edge instead. New edge only allows strings that were allowed before.
While the box contains some state $s$:
   for all states $r$, $t$ with $(r, s)$ and $(s, t)$ in $E$:
       create a direct edge $(r, t)$ by Rule 2
       delete $s$ (no longer needed)
       merge all parallel edges by Rule 1

Add new start state and final state
Construction Overview

While the box contains some state $s$: for all states $r, t$ with $(r, s)$ and $(s, t)$ in $E$: create a direct edge $(r, t)$ by Rule 2; delete $s$ (no longer needed); merge all parallel edges by Rule 1.

When the loop exits, the graph looks like this:

A is a regular expression with the same language as the original NFA.
Converting an NFA to a regular expression

Consider the DFA for the mod 3 sum

– Accept strings from \{0,1,2\}^* where the digits mod 3 sum of the digits is 0
Splicing out a state $t_1$

Create direct edges between neighbors of $t_1$ (so that we can delete it afterward)
Splicing out a state $t_1$

Regular expressions to add to edges

- $t_0 \rightarrow t_1 \rightarrow t_0: 10^*2$
- $t_0 \rightarrow t_1 \rightarrow t_2: 10^*1$
- $t_2 \rightarrow t_1 \rightarrow t_0: 20^*2$
- $t_2 \rightarrow t_1 \rightarrow t_2: 20^*1$
Splicing out a state $t_1$

Delete $t_1$ now that it is redundant

$t_0 \rightarrow t_1 \rightarrow t_0 : 10*2$
$t_0 \rightarrow t_1 \rightarrow t_2 : 10*1$
$t_2 \rightarrow t_1 \rightarrow t_0 : 20*2$
$t_2 \rightarrow t_1 \rightarrow t_2 : 20*1$
Splicing out a state $t_1$

Create direct edges between neighbors of $t_2$ (so that we can delete it afterward)
Splicing out a state $t_1$

Regular expressions to add to edges

- $R_1: 0 \cup 10^2$
- $R_2: 2 \cup 10^1$
- $R_3: 1 \cup 20^2$
- $R_4: 0 \cup 20^1$
Splicing out state $t_2$ (and then $t_0$)

Delete $t_2$ now that it is redundant

\[
R_1: \quad 0 \cup 10*2 \\
R_2: \quad 2 \cup 10*1 \\
R_3: \quad 1 \cup 20*2 \\
R_4: \quad 0 \cup 20*1 \\
R_5: \quad R_1 \cup R_2 R_4^* R_3
\]
Splicing out state $t_2$ (and then $t_0$)

Create direct (s,f) edge so we can delete $t_0$

$R_1$: $0 \cup 10*2$
$R_2$: $2 \cup 10*1$
$R_3$: $1 \cup 20*2$
$R_4$: $0 \cup 20*1$
$R_5$: $R_1 \cup R_2 R_4 * R_3$
Splicing out state $t_2$ (and then $t_0$)

Regular expressions to add to edges

$R_1$: $0 \cup 10*2$
$R_2$: $2 \cup 10*1$
$R_3$: $1 \cup 20*2$
$R_4$: $0 \cup 20*1$
$R_5$: $R_1 \cup R_2 R_4 * R_3$

$t_0 \rightarrow t_1 \rightarrow t_0: R_5$
Splicing out state $t_2$ (and then $t_0$)

Delete $t_0$ now that it is redundant

\[
R_1: \quad 0 \cup 10*2 \\
R_2: \quad 2 \cup 10*1 \\
R_3: \quad 1 \cup 20*2 \\
R_4: \quad 0 \cup 20*1 \\
R_5: \quad R_1 \cup R_2 R_4^* R_3 \\
\]

$R_6: \quad R_5^*$
Splicing out state $t_2$ (and then $t_0$)

Regular expressions to add to edges

- $R_1$: $0 \cup 10^2$
- $R_2$: $2 \cup 10^1$
- $R_3$: $1 \cup 20^2$
- $R_4$: $0 \cup 20^1$
- $R_5$: $R_1 \cup R_2 R_4^* R_3$
- $R_6$: $R_5^*$

Final regular expression: $R_6 = (0 \cup 10^2 \cup (2 \cup 10^1)(0 \cup 20^1)^*(1 \cup 20^2))^*$
Application of FSMs: Pattern matching

• Given
  – a string $s$ of $n$ characters
  – a pattern $p$ of $m$ characters
  – usually $m \ll n$

• Find
  – all occurrences of the pattern $p$ in the string $s$

• Obvious algorithm:
  – try to see if $p$ matches at each of the positions in $s$
    stop at a failed match and try matching at the next
Application of FSMs: Pattern Matching

- With DFAs can do this in $O(m + n)$ time.

- See Extra Credit problem on HW8 for some ideas of how to get to $O(m^2 + n)$. 
Last time: NFA to DFA
Exponential Blow-up in Simulating Nondeterminism

• In general the DFA might need a state for every subset of states of the NFA
  – Power set of the set of states of the NFA
  – $n$-state NFA yields DFA with at most $2^n$ states
  – We saw an example where roughly $2^n$ is necessary
    “Is the $n^{\text{th}}$ char from the end a 1?”

The famous “P=NP?” question asks whether a similar blow-up is always necessary to get rid of nondeterminism for polynomial-time algorithms