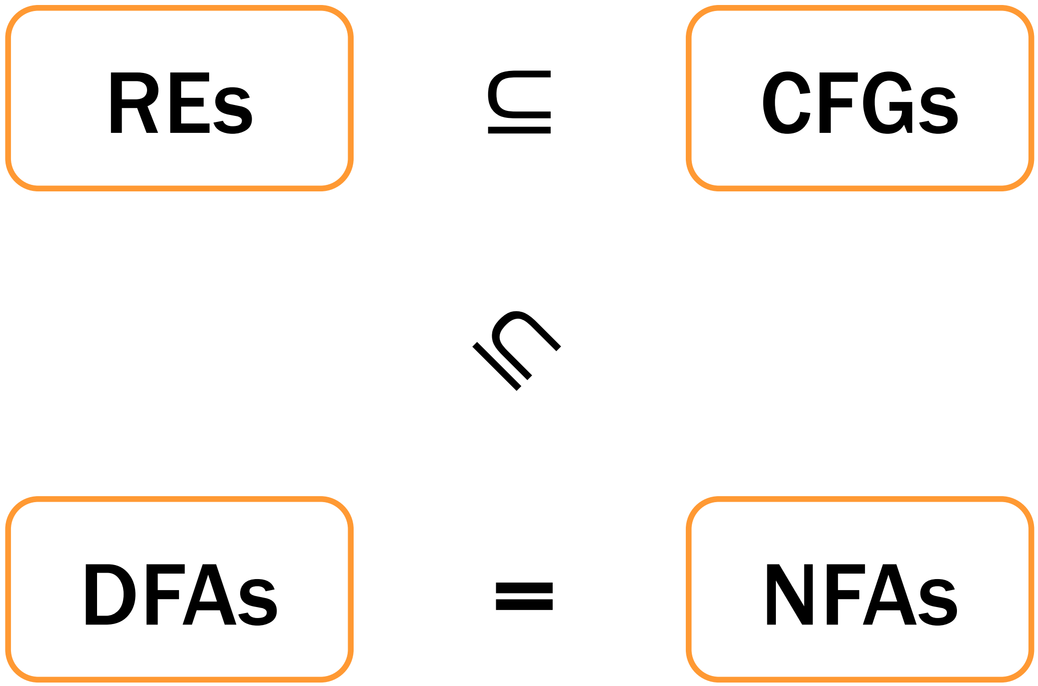
CSE 311: Foundations of Computing

Lecture 24b: NFAs → Regular expressions

The story so far...



Regular expressions $\subseteq$ NFAs $\equiv$ DFAs

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...

REs $\subseteq$ **CFGs**

DFAs $=$ **NFAs**

Is this $\subseteq$ really "$=$" or "$\subsetneq$"?

Regular expressions ≡ NFAs ≡ 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 sketched below but you will not be tested on it

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 **ε** or ***a***

- Machine can follow an edge labeled by **A** by reading a <u>string of input characters</u> in the language of A
  - (if A is ***a*** or ε, 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**

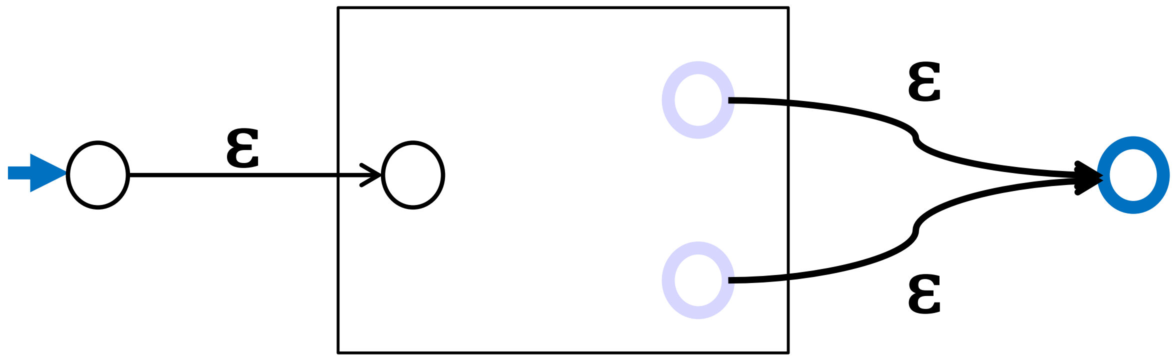
    NFAs already have edges labeled **ε** or ***a***

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

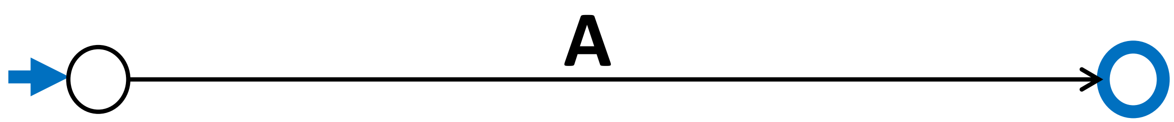
- **Def: A string** x **is accepted by a generalized NFA iff there is a *path* from start to final state labeled by a regular expression whose language contains** x

Construction Idea

Add new start state and final state

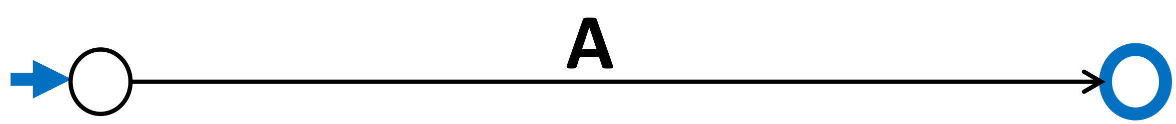


Then delete the original states one by one,
adding edges to keep the same language,
until the graph looks like:



Starting from an NFA

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

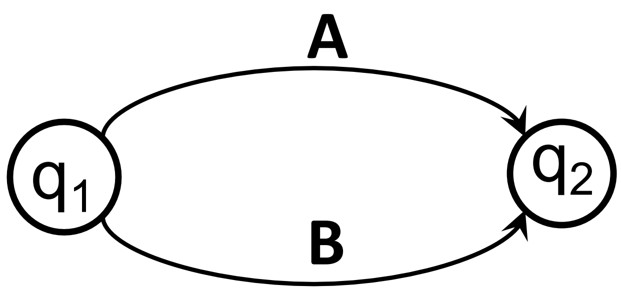


Final graph has only one path to the accepting state, which is labeled by A, so it accepts iff x is in the language of A

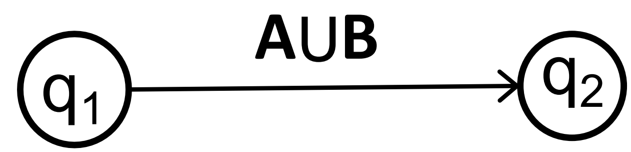
Thus, A is a regular expression with the same language as the original NFA.

Only two simplification rules

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



by

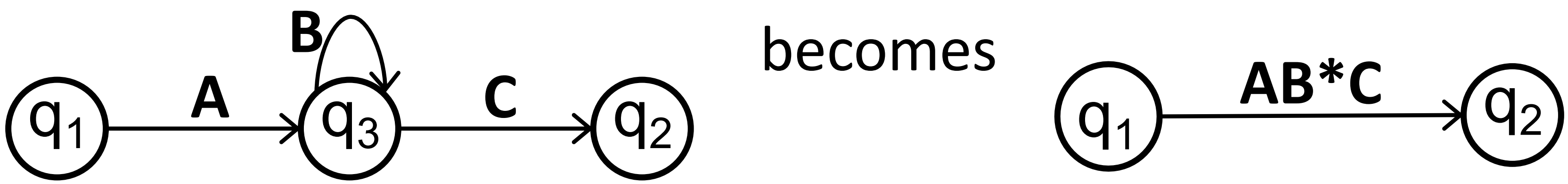


If the machine would have used the edge labeled A by consuming an input x in the language of A, it can instead use the edge labeled A∪B.

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

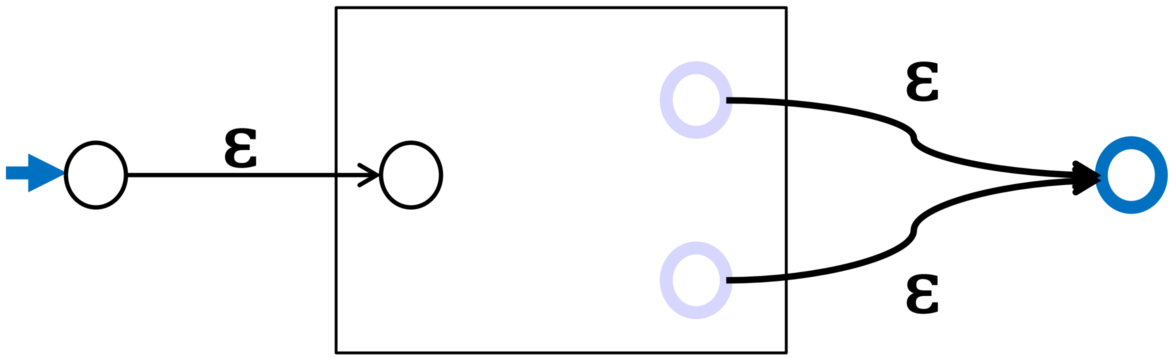


for *every* pair of states q_1, q_2 (even if q_1=q_2)

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.

Construction Overview

Add new start state and final state



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

Construction Overview

While the box contains some state s:

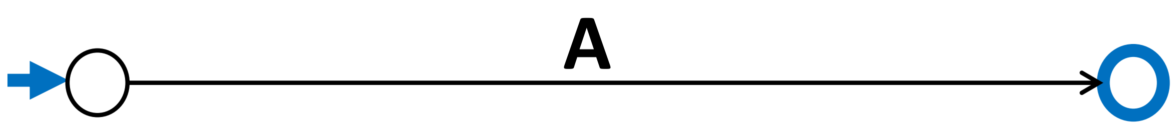
    for all states r, t with (r, s) and (s, t) in E:

        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:

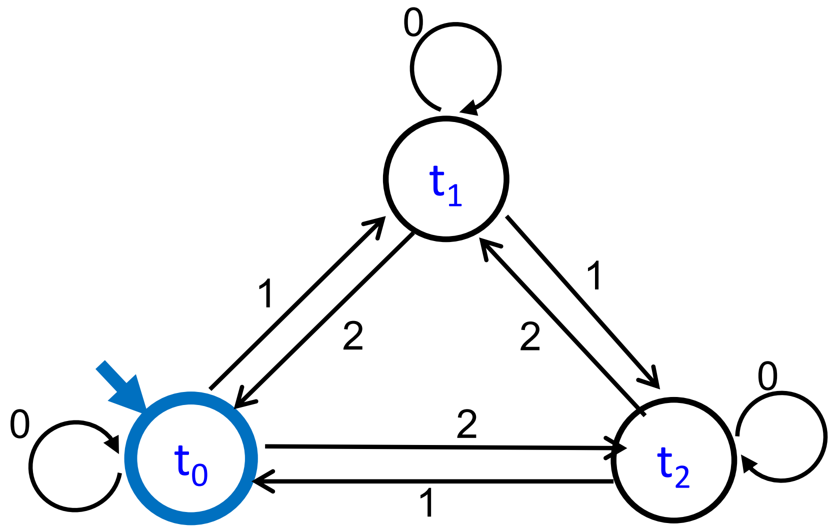


A is a regular expression with the same language as the original NFA.

Converting an NFA to a regular expression

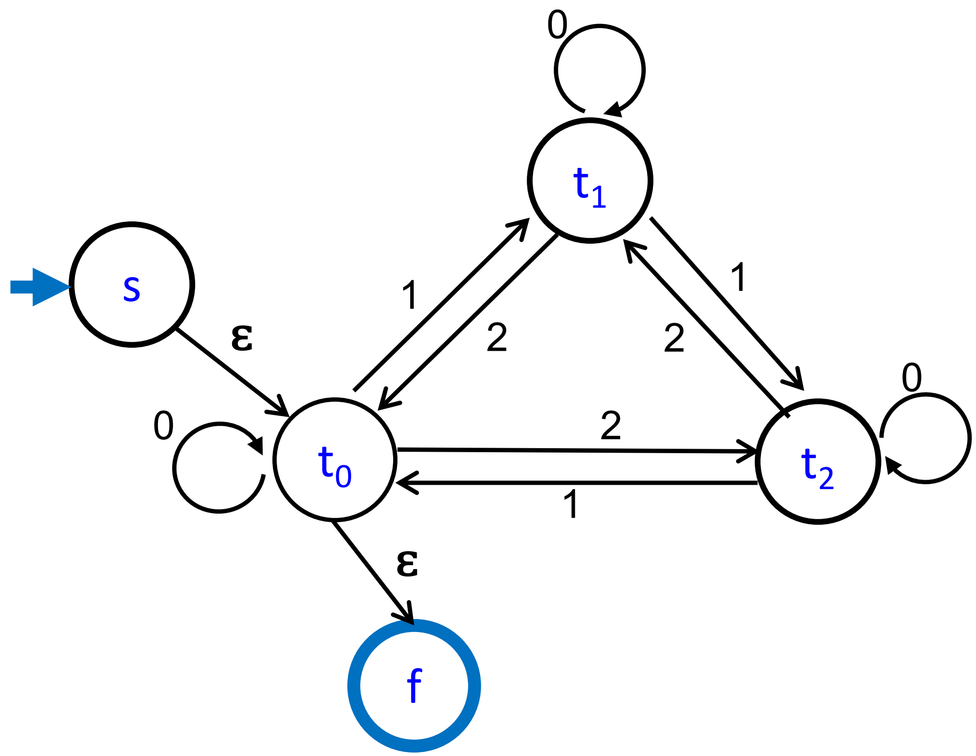
Consider the DFA for the mod 3 sum

- Accept strings from {0,1,2}* where the digits mod 3 sum of the digits is 0



Splicing out a state t_1

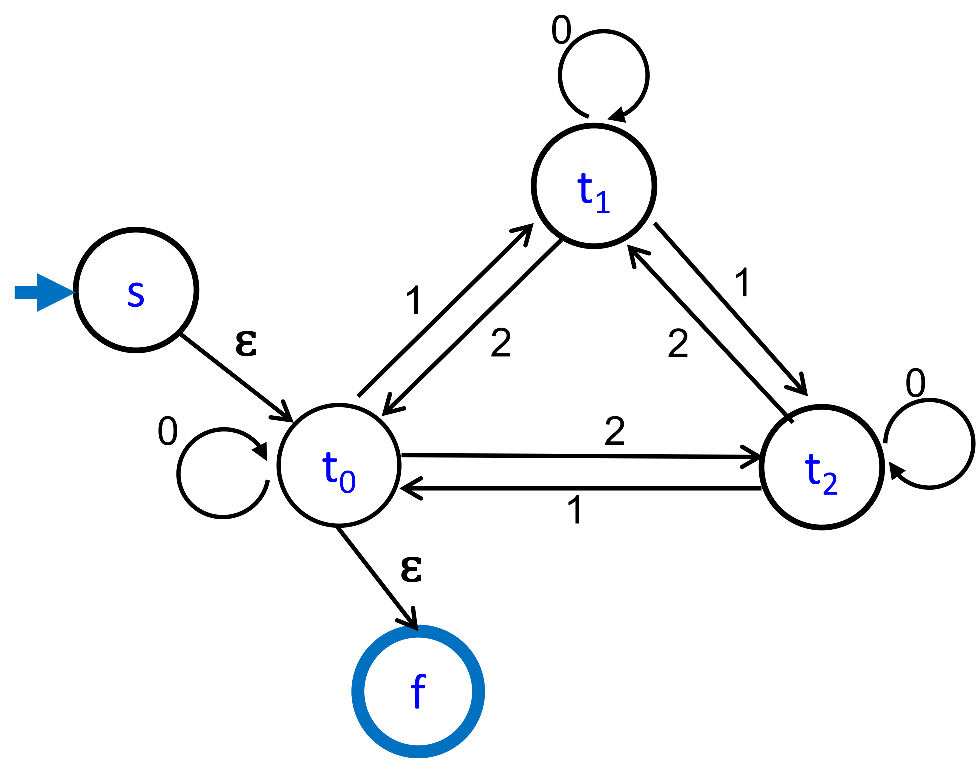
Create direct edges between neighbors of t_1 (so that we can delete it afterward)



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

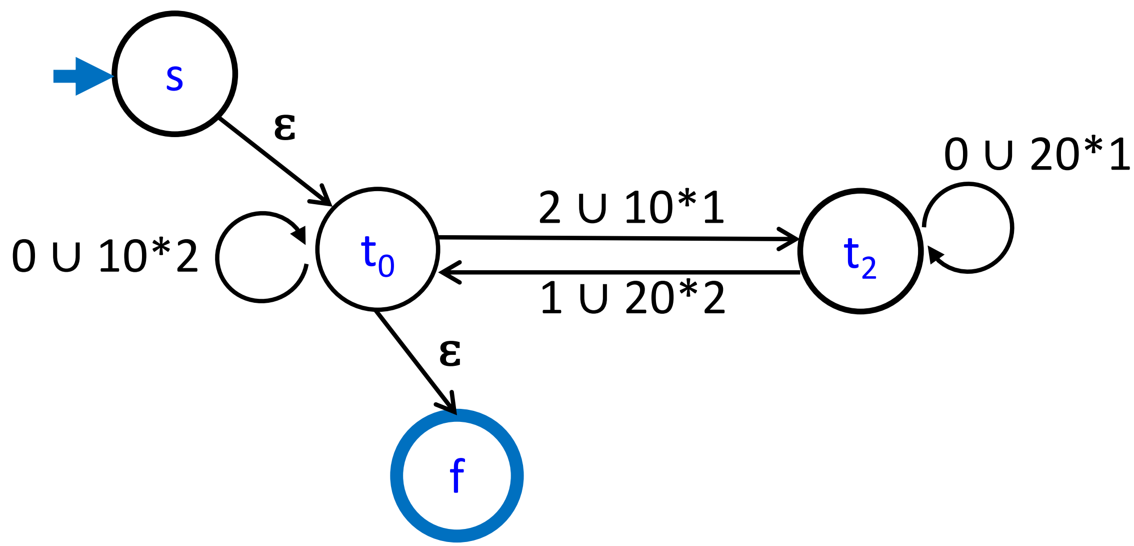
Delete t_1 now that it is redundant

$t_0 \rightarrow t_1 \rightarrow t_0$: 10^*2

$t_0 \rightarrow t_1 \rightarrow t_2$: 10^*1

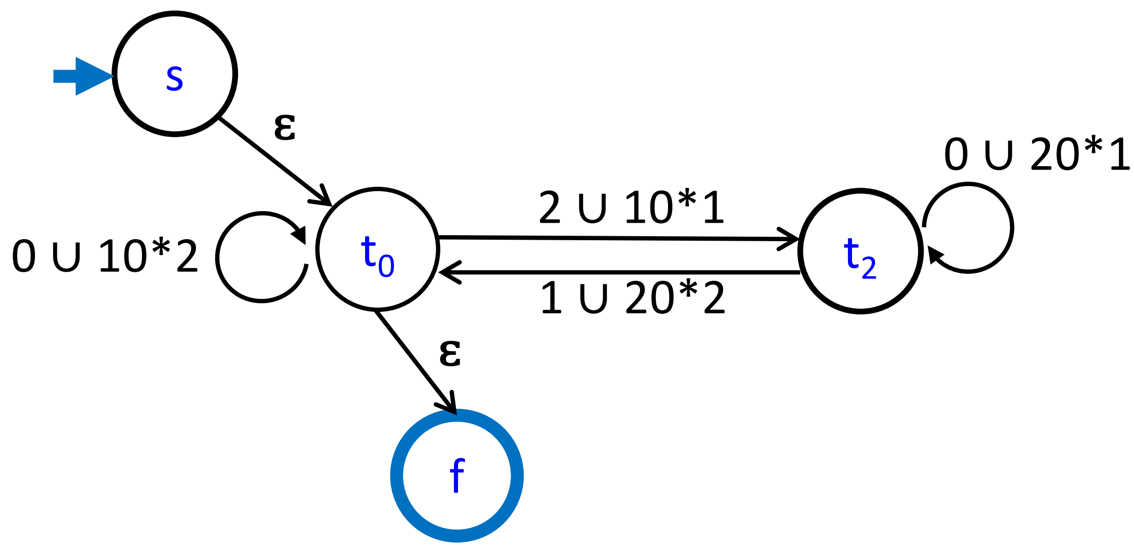
$t_2 \rightarrow t_1 \rightarrow t_0$: 20^*2

$t_2 \rightarrow t_1 \rightarrow t_2$: 20^*1



Splicing out a state t_1

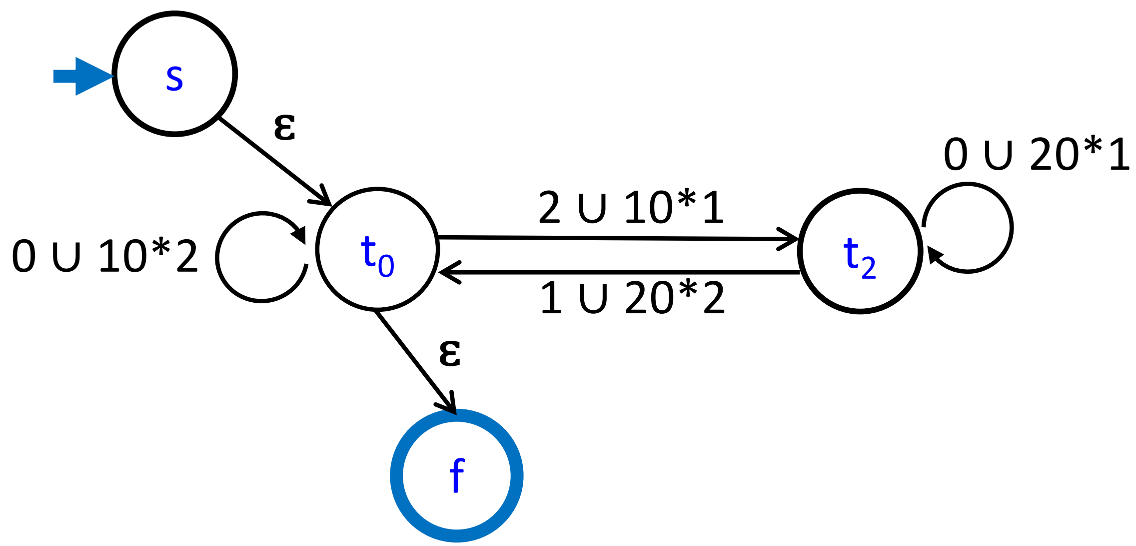
Create direct edges between neighbors of t_2 (so that we can delete it afterward)



Splicing out a state t_1

Regular expressions to add to edges

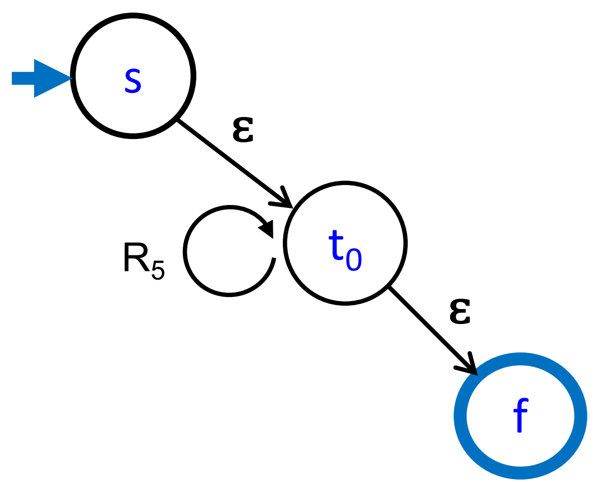
R_1: $0 \cup 10^*2$
R_2: $2 \cup 10^*1$
R_3: $1 \cup 20^*2$
R_4: $0 \cup 20^*1$



Splicing out state t_2 (and then t_0)

Delete t_2 now that it is redundant

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



R_5: $R_1 \cup R_2R_4^*R_3$

Splicing out state t_2 (and then t_0)

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

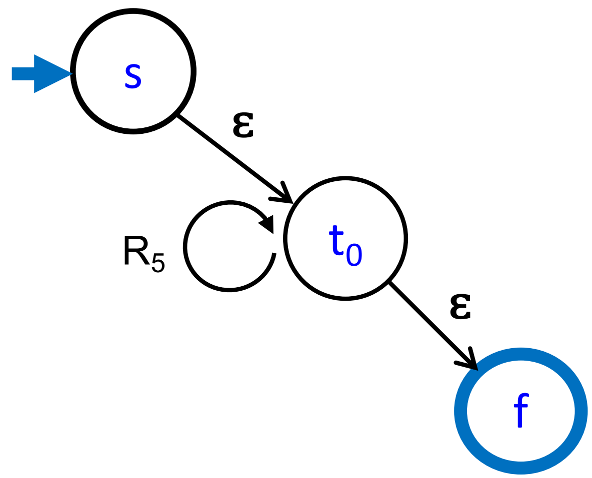
R_1: $0 \cup 10^*2$

R_2: $2 \cup 10^*1$

R_3: $1 \cup 20^*2$

R_4: $0 \cup 20^*1$

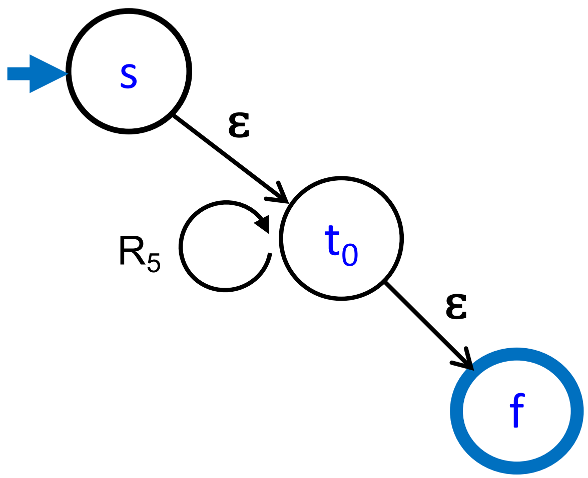
R_5: $R_1 \cup R_2R_4{}^*R_3$



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: $R_1 \cup R_2R_4^*R_3$

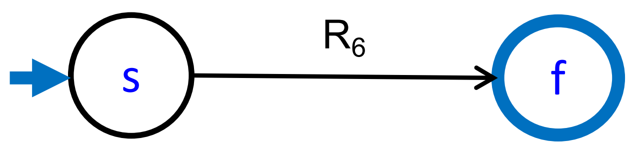


$t_0 \rightarrow t_1 \rightarrow t_0$: $R_5{}^*$

Splicing out state t_2 (and then t_0)

Delete t_0 now that it is redundant

R_1: $0 \cup 10^*2$
R_2: $2 \cup 10^*1$
R_3: $1 \cup 20^*2$
R_4: $0 \cup 20^*1$
R_5: $R_1 \cup R_2R_4{}^*R_3$

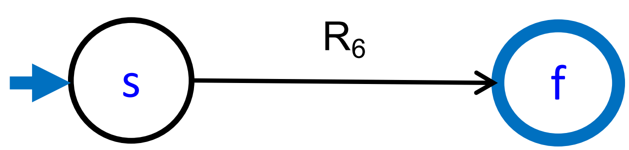


R_6: $R_5{}^*$

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: $R_1 \cup R_2R_4^*R_3$
R_6: R_5^*



Final regular expression: R_6 =
$(0 \cup 10^*2 \cup (2 \cup 10^*1)(0 \cup 20^*1)^*(1 \cup 20^*2))^*$

The story so far...

REs $\subseteq$ CFGs

$\parallel$

DFAs $=$ NFAs

The story so far...

$$\begin{array}{ccc} \textbf{REs} & \subseteq & \textbf{CFGs} \\ \| & & \\ \textbf{DFAs} & = & \textbf{NFAs} \end{array}$$

<u>Next time</u>: Is this $\subseteq$ really “=” or “$\subsetneq$”?