Second Part of Regular Expressions Equivalence with Finite Automata

September 11, 2013
Lemma 1.60

If a language is regular then it is specified by a regular expression

**Proof idea:** For a given regular language $A$ we will construct a regular expression that describes $A$. 
Because $A$ is regular, there is a DFA $D_A$ that recognizes $A$. 

Note: This procedure is broken in two parts:
1. Convert a DFA into a generalized nondeterministic finite automaton (GNFA)
2. Convert GNFA into a regular expression
Because $A$ is regular, there is a DFA $D_A$ that recognizes $A$

$D_A$ will be converted into a regular expression $R_A$ that specifies $A$
Procedure

- Because $A$ is regular, there is a DFA $D_A$ that recognizes $A$
- $D_A$ will be converted into a regular expression $R_A$ that specifies $A$

**Note:** This procedure is broken in two parts:
Because $A$ is regular, there is a DFA $D_A$ that recognizes $A$

$D_A$ will be converted into a regular expression $R_A$ that specifies $A$

**Note:** This procedure is broken in two parts:

1. Convert a DFA into a generalized nondeterministic finite automaton GNFA
Procedure

▶ Because $A$ is regular, there is a DFA $D_A$ that recognizes $A$
▶ $D_A$ will be converted into a regular expression $R_A$ that specifies $A$

**Note:** This procedure is broken in two parts:

1. Convert a DFA into a *generalized nondeterministic finite automaton* GNFA
2. Convert GNFA into a regular expression
What is an GNFA?

A GNFA is an NFA wherein the transition arrows may have any regular expressions as labels, instead only members of the alphabet or \( \epsilon \) \( \). Hence, GNFA reads strings specified by regular expressions (block of symbols) from the input (not necessarily just one symbol). GNFA moves along a transition arrow connecting two states representing regular expression, Figure 1.

Second Part of Regular Expressions Equivalence with Finite Au
What is an GNFA?

- A GNFA is an NFA wherein the transition arrows may have any regular expressions as labels, instead only members of the alphabet or $\epsilon$. 

Second Part of Regular Expressions Equivalence with Finite Automata
What is an GNFA?

- A GNFA is an NFA wherein the transition arrows may have any regular expressions as labels, instead only members of the alphabet or $\varepsilon$
- Hence, GNFA reads strings specified by regular expressions (block of symbols) from the input (not necessarily just one symbol)
What is an GNFA?

- A GNFA is an NFA wherein the transition arrows may have any regular expressions as labels, instead only members of the alphabet or $\varepsilon$.
- Hence, GNFA reads strings specified by regular expressions (block of symbols) from the input (not necessarily just one symbol).
- GNFA moves along a transition arrow connecting two states representing regular expression, Figure 1.
Figure 1: A GNFA
A GNFA is nondeterministic and so, it may have many different ways to process the same input string.

A GNFA accepts its input if its processing can cause the GNFA to be in an accept state at the end of the input.
GNFA of special form

- The start state has transition arrows to every other state but no arrow coming from any other state.
- There is only one accept state and it has arrows coming in from every other state, but has no arrows going to any other state; in addition, the accept state is not the same with the start state.
- Except for start and accept states, one arrow goes from every state to every other state and from each state to itself.
GNFA of special form

- The start state has transition arrows to every other state but no arrow coming from any other state.
- There is only one accept state and it has arrows coming in from every other state, but has no arrows going to any other state; in addition, the accept state is not the same with the start state.
- Except for start and accept states, one arrow goes from every state to every other state and from each state to itself.
GNFA of special form

- The start state has transition arrows to every other state but no arrow coming from any other state.
- There is only one accept state and it has arrows coming in from every other state, but has no arrows going to any other state; in addition, the accept state is not the same with the start state.
GNFA of special form

- The start state has transition arrows to every other state but no arrow coming from any other state.
- There is only one accept state and it has arrows coming in from every other state, but has no arrows going to any other state; in addition, the accept state is not the same with the start state.
- Except for start and accept states, one arrow goes from every state to every other state and from each state to itself.
A DFA is converted to a GNFA of special form by the following procedure:
Converting DFA to GNFA

A DFA is converted to a GNFA of special form by the following procedure:

1. Add a new start state with an $\epsilon$ arrow to the old start state and a new accept state with an $\epsilon$ arrow from all old accept states
A DFA is converted to a GNFA of special form by the following procedure:

1. Add a new start state with an $\epsilon$ arrow to the old start state and a new accept state with an $\epsilon$ arrow from all old accept states.

2. If any arrows have multiple labels or if there are multiple arrows going between the same two states in the same direction replace each with a single arrow whose label is the union of the previous labels.
Converting DFA to GNFA

A DFA is converted to a GNFA of special form by the following procedure:

1. Add a new start state with an $\epsilon$ arrow to the old start state and a new accept state with an $\epsilon$ arrow from all old accept states

2. If any arrows have multiple labels or if there are multiple arrows going between the same two states in the same direction replace each with a single arrow whose label is the union of the previous labels

3. Add arrows labeled $\emptyset$ between states that had no arrows
Note

Adding $\emptyset$ transitions don’t change the language recognized by DFA because a transition labeled by $\emptyset$ can never be used

**Assumption:** now we assume that all GNFAs are in the special form just defined.
Assume that GNFA has $k$ states
Converting $\textit{GNFA} \rightarrow \textit{RE}$

Assume that GNFA has $k$ states

- Because start and accept states are different from each other, it results that $k \geq 2$
Assume that GNFA has $k$ states

- Because start and accept states are different from each other, it results that $k \geq 2$
- If $k > 2$ we construct an equivalent GNFA with $k - 1$ states. This can be repeated for each new GNFA until we obtain a GNFA with $k = 2$ states.
Assume that GNFA has $k$ states

- Because start and accept states are different from each other, it results that $k \geq 2$
- If $k > 2$ we construct an equivalent GNFA with $k - 1$ states. This can be repeated for each new GNFA until we obtain a GNFA with $k = 2$ states.
- If $k = 2$, GNFA has a single arrow that goes from start to accept and is labeled by a regular expression that specifies the language recognized by the original DFA
Example DFA conversion

Assuming that the original DFA has 3 states the process of its conversion is shown in Figure 2

Figure 2: Example DFA conversion to regular expression
The crucial step is the construction of an equivalent GNFA with one fewer states than a GNFA when $k > 2$.

This is done by selecting a state, ripping it out of the machine, and repairing the remainder so that the same language is still recognized.

Any state can be selected for ripping, providing that it is not start or accept state. Such a state exist because $k > 2$. 

Second Part of Regular Expressions Equivalence with Finite Automata
The crucial step is the construction of an equivalent GNFA with one fewer states than a GNFA when GNFA has \( k > 2 \) states.
The crucial step is the construction of an equivalent GNFA with one fewer states than a GNFA when GNFA has $k > 2$ states.

This is done by selecting a state, ripping it out of the machine, and repairing the remainder so that the same language is still recognized.
The crucial step is the construction of an equivalent GNFA with one fewer states than a GNFA when GNFA has $k > 2$ states.

This is done by selecting a state, ripping it out of the machine, and repairing the remainder so that the same language is still recognized.

Any state can be selected for ripping, providing that it is not start or accept state. Such a state exist because $k > 2$.
Assume that the state of a GNFA selected for ripping is $q_{rip}$.
Assume that the state of a GNFA selected for ripping is $q_{rip}$

- After removing $q_{rip}$ we repair the machine by altering the regular expressions that label each of the remaining transitions.
Assume that the state of a GNFA selected for ripping is $q_{rip}$

- After removing $q_{rip}$ we repair the machine by altering the regular expressions that label each of the remaining transitions.
- The new labels compensate for the absence of $q_{rip}$ by adding back the lost computation.
Assume that the state of a GNFA selected for ripping is $q_{rip}$

- After removing $q_{rip}$ we repair the machine by altering the regular expressions that label each of the remaining transitions.
- The new labels compensate for the absence of $q_{rip}$ by adding back the lost computation.
- The new label of the arrow going from state $q_i$ to $q_j$ is a regular expression that specifies all strings that would take the machine from $q_i$ to $q_j$ either directly or via $q_{rip}$.
We illustrate the approach of ripping and repairing in Figure 3

Figure 3: Ripping and repairing an GNFA
New labels are obtained by concatenating regular expressions of the arrows that go through $q_{rip}$ and union them with the labels of the arrows that travel directly between $q_i$ and $q_j$. This construct is carried out for each arrow that goes from state $q_i$ to any state $q_j$ including $q_i = q_j$. 
Formal proof

- First we need to define formally the GNFA
- Since new labels are regular expressions we use the symbol $\mathcal{R}_\Sigma$ to denote the collection of regular expressions over an alphabet $\Sigma$
- To simplify, denote by $q_s$ and $q_a$ the start and accept states of the GNFA
Because an arrow connects every state to every other state, except that no arrows are coming from $q_a$ or going to $q_s$, the domain of the transition function of a GNFA is

$$\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow \mathcal{R}_\Sigma$$

If $\delta(q_i, q_j) = R$ the arrow from $q_i$ to $q_j$ has the label $R$
A generalized nondeterministic finite automaton (GNFA) is a 5-tuple \((Q, \Sigma, \delta, q_s, q_a)\) where:

1. \(Q\) is the finite set of states
2. \(\Sigma\) is the input alphabet
3. \(\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow R_\Sigma\) is the transition function where \(R_\Sigma\) is the set of regular expressions over \(\Sigma\)
4. \(q_s\) is the unique start state
5. \(q_a\) is the unique accept state and \(q_a \neq q_s\).
A GNFA accepts a string $w \in \Sigma^*$ if $w = w_1w_2 \ldots w_k$ where $w_i \in \Sigma^*$, $1 \leq i \leq k$, and a sequence of states $q_0, q_1, \ldots, q_k$ exits such that:

1. $q_0 = q_s$ is the start state
2. $q_k = q_a$ is the accept state
3. For each $i$, $\delta(q_{i-1}, q_i) = R_i$ and $w_i \in L(R_i)$, i.e., $R_i$ is the regular expression labeling the arrow from $q_{i-1}$ to $q_i$ and $w_i$ is an element of the language specified by this expression
More proof ideas

Returning to the proof of Lemma 1.60, we assume that $M$ is a DFA recognizing the language $A$ and proceed as follows:

- Convert $M$ into a GNFA $G$ by adding a new start state and a new accept state and the additional arrows
- Use the procedure $Convert(G)$ that maps $G$ into a regular expression, as explained before, while preserving the language $A$

**Note:** $Convert()$ is recursive; however the case when GNFA has only two states is handled without recursion
Convert\((G)\)

1. Let \(k\) be the number of states of \(G\), \(k \geq 2\).
2. If \(k = 2\) then \(G\) must consists of a start state and an accept state and a single arrow connecting them, labeled by a regular expression \(R\). Return \(R\)
3. While \(k > 2\), select any state \(q_{rip} \in Q\), different from \(q_s\) and \(q_a\) and let \(G'\) be the GNFA \((Q', \Sigma, \delta', q_s, q_a)\) where:
   - \(Q' = Q - \{q_{rip}\}\)
   - for any \(q_i \in Q' - \{q_a\}\) and any \(q_j \in Q' - \{q_s\}\) let \(\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)\) where:
     - \(R_1 = \delta(q_i, q_{rip})\)
     - \(R_2 = \delta(q_{rip}, q_{rip})\)
     - \(R_3 = \delta(q_{rip}, q_j)\)
     - \(R_4 = \delta(q_i, q_j)\)
   - Convert\((G')\);
Claim 1.65

For any GNFA $G$, $\text{Convert}(G)$ is equivalent to $G$

**Proof:** by induction on $k$, the number of states of $G$
Induction Basis:

\( k = 2 \)

- If \( G \) has only two states, by definition, it can have only a single arrow which goes from \( q_s \) to \( q_a \)
- The regular expression labeling this arrow specify the language accepted by \( G \)
- Since this expression is returned by \( Convert(G) \), it means that \( G \) and \( Convert(G) \) are equivalent
Induction Step

Assume that the claim is true for $G$ having $k - 1$ states and use this assumption to show that the claim is true for an GNFA with $k$ states

- Observe from construction that $G$ and $G'$ recognize the same language
- Suppose $G$ accepts the input $w$. Then in an accepting branch of computation, $G$ enters the sequence of states $q_s, q_1, q_2, q_3, \ldots, q_a$
- Show that $G'$ has an accepting computation for $w$, too.
1. If none of the states $q_s, q_1, q_2, \ldots, q_a$ is $q_{rip}$, clearly $G'$ also accepts $w$ because each of the new regular expressions labeling arrows of $G'$ contain the old regular expressions as part of a union.
1. If none of the states $q_s, q_1, q_2, \ldots, q_a$ is $q_{rip}$, clearly $G'$ also accepts $w$ because each of the new regular expressions labeling arrows of $G'$ contain the old regular expressions as part of a union.

2. If $q_{rip}$ does appear in the computation $q_s, q_1, q_2, \ldots, q_a$ by removing each run of consecutive $q_{rip}$ states we obtain an accepting computation for $G'$. This is because states $q_i$ and $q_j$ bracketing a run of consecutive $q_{rip}$ states have a new regular expression on the arrow between them that specify all strings taking $q_i$ to $q_j$ via $q_{rip}$ on $G$. So, $G'$ accepts $w$ in this case too.
For the other direction, suppose that $G'$ accepts $w$.

1. Each arrow between any two states $q_i$ and $q_j$ in $G'$ is labeled by a regular expression that specifies strings specified by arrows in $G$ from $q_i$ directly to $q_j$ or via $q_{rip}$.

2. Hence, by the definition of GNFA it follows that $G$ must also accept $w$.

That is, $G$ and $G'$ accept the same language.
The induction hypothesis states that when the algorithm calls itself recursively on input $G'$, the result is a regular expression that is equivalent to $G'$ because $G'$ has $k - 1$ states.

Hence, that regular expression is also equivalent to $G$ because $G'$ is equivalent to $G$.

Consequently, $\text{Convert}(G)$ and $G$ are equivalent.
Example 1.35

Convert the DFA $D$ in Figure 4 into the regular expression that specifies the language accepted by $D$

![Diagram of DFA D]

**Figure 4**: DFA $D$ to be converted
Figure 5 shows the four-state GNFA obtained from $D$ by adding new start state and accept state and replacing $a, b$ by $a \cup b$. 

![Diagram](#)

**Figure 5**: GNFA $G_1$ obtained from $D$
Removing state 1 and then state 2, Figure 6 shows the GNFA $G_3$:

![Diagram](Image)

Figure 6: GNFA $G_3$ obtained from $G_2$