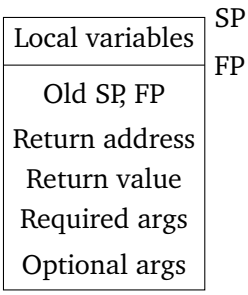
Sample Solution

Assignment 7

CSCI 3136—Winter 2019

Question 1

The stack frame looks exactly as for normal function calls, with the exception that the optional arguments are pushed on the stack *before* the required arguments:



This ensures that the number of optional arguments has no impact on the offset from the frame pointer of any other part of the stack frame.

Call sequence: The caller pushes

- Optional arguments
- Required arguments

on the stack and reserves space for the return value. It then makes the call, thereby pushing the return address.

The callee

- Pushes the current FP and SP onto the stack,
- Sets FP = SP, and
- Allocates space for local variables.

The callee can now access local variables, required arguments, and the return value as usual, at fixed offsets from the FP. Similarly, the first optional argument is stored at a fixed offset from the FP, so it is easiy to implement C-style iteration over the arguments by initializing a pointer to the first argument and then advancing this pointer as each argument is read.

When returning, the callee

- Cleans up its local variables,
- Restores the FP and SP from the stack, and
- Returns control to the caller.

This happens exactly as for a normal function call because all this information is stored at fixed offsets.
The caller

- Retrieves the return value and
- Restores the SP to the end of the list of optional arguments.

This requires knowledge of the number and types of the optional arguments passed to the call, but the caller has this information and this information can be generated at compile time for each function call. Thus, this translates into a single addition to the current SP value.

Question 2

Question 2a

The problem is that the two string representations in this case are infinite. Indeed, both types have the representation

```
struct char pointer struct char pointer struct char pointer ...
```

The notion of equality of infinite strings is poorly defined. Given that these infinite strings are obtained recursively from a finite description (the set of types in the program), it can be shown that two types are structurally equivalent if their strings have a sufficiently long common prefix. (Indeed, this is a rough approximation of the idea behind the solution in part (b).) However, the notion of "sufficiently long" depends on the program being checked, so this condition is difficult to check by a compiler.

Question 2b

I had an earlier version of this proof that was half as long but not 100% rigorous. I decided to make it completely rigorous. I hope you learn something from working through this proof. Note, however, that if you decided to attempt the bonus question, I did not expect you to provide a proof that achieves even close to this level of rigour.

Our proof strategy is as follows:

1. First we formalize the notion of structural equivalence defined in the assignment by introducing classification functions ϕ_A and ϕ_B over the set of all sequences of element selectors. The construction will ensure that A and B are structurally equivalent if and only if $\phi_A = \phi_B$, that is, if and only if $\phi_A(\sigma) = \phi_B(\sigma)$ for every sequence σ of element selectors.

2. Then we define two DFA-like automata M_A and M_B that compute ϕ_A and ϕ_B, that is, $\psi_{M_A}(\sigma) = \phi_A(\sigma)$ and $\psi_{M_B}(\sigma) = \phi_B(\sigma)$ for every sequence σ of element selectors, where $\psi_{M_A}(\sigma)$ and $\psi_{M_B}(\sigma)$ denote the outputs of M_A and M_B when given σ as an input.

3. Next we consider the two automata $\mu(M_A)$ and $\mu(M_B)$ obtained by applying a simple extension of the DFA minimization procedure discussed in class to M_A and M_B. $\mu(M)$ refers to the minimized version of M, for any automaton M. The minimization procedure ensures that $\psi_{\mu(M)} = \psi_M$ for every automaton M.

4. The next (and hardest) step is to prove that $\mu(M_A) = \mu(M_B)$ if and only if $\psi_{M_A} = \psi_{M_B}$. In other words, *the minimized classification automaton for any classification function ϕ is unique.*

5. Finally, we discuss a simple algorithm that determines whether two classification automata are identical.

The procedure to test the structural equivalence of two types A and B now works as follows: First we construct the automata M_A and M_B. Then we minimize them to obtain $\mu(M_A)$ and $\mu(M_B)$. Finally, we test whether $\mu(M_A) = \mu(M_B)$ and output the result. This procedure is correct because, by the properties of the construction outlined above, A and B are structurally equivalent if and only if $\phi_A = \phi_B$, which holds if and only if $\psi_{M_A} = \psi_{M_B}$, which holds if and only if $\mu(M_A) = \mu(M_B)$. It remains to provide the different parts of this construction.

Classification Functions for Selector Sequences

Consider the alphabet Σ consisting of all element selectors in the program. Since the program is finite, Σ is finite. Σ^* is the set of all strings over Σ, that is, the set of all sequences of element selectors, valid or not. For a type A and a sequence $\sigma = .s_1.s_2\ldots$ of element selectors, we write $A\sigma$ to refer to the type $A.s_1.s_2\ldots$

Let T be the set of all built-in types plus two special elements called `invalid` and `compound`. We define a classification function $\phi_A(\sigma) : \Sigma^* \to T$ for every type A in the program such that $\phi_A(\sigma) = \texttt{invalid}$ if σ is not a valid sequence of element selectors for the type A; $\phi_A(\sigma) = \texttt{compound}$ if σ is a valid sequence of element selectors for the type A but $A\sigma$ is a compound type; and $\phi_A(\sigma) = t$, where t is a built-in type, if $A\sigma = t$. If A is a built-in type, then the only valid sequence of element selectors is ϵ and $A\epsilon = A$.

These classification functions simply formalize the notion of structural equivalence defined in the assignment since it is easy to see that two types A and B are structurally equivalent if and only if $\phi_A = \phi_B$, that is, if and only if $\phi_A(\sigma) = \phi_B(\sigma)$ for all $\sigma \in \Sigma^*$.

Automata to Compute Classification Functions

The construction here is very similar to the extension of DFA used by scanners to recognize multiple token types instead of just accepting or rejecting a given input. A classification automaton $M = (S, q_0, \Sigma, \Gamma, \delta, \lambda)$ has a set of states S, a start state $q_0 \in S$, an alphabet Σ, a set of classes Γ, a transition function $\delta : S \times \Sigma \to S$, and a labelling function $\lambda : S \to \Gamma$ that assigns a class to each state in S.

Such an automaton computes a classification function ψ_M where $\psi_M(\sigma)$ is the label of the state reached by M after consuming the input σ. Formally, $\psi_M(\sigma) = \lambda(\delta^*(q_0, \sigma))$ where $\delta^*(q, \sigma)$ is defined analogously to the definition for a DFA. Note that a DFA is a special case of a classification automaton where $\Gamma = \{\texttt{accept}, \texttt{reject}\}$ and

$$\lambda(q) = \begin{cases} \texttt{accept} & \text{if } q \in F \\ \texttt{reject} & \text{if } q \notin F \end{cases}.$$

To construct an automaton M_A such that $\psi_{M_A} = \phi_A$, we set S to be the set of all types in the program plus a special state `invalid`. Since the program has finite length, S is finite. $q_0 = A$. Σ is the set of all element selectors, which is finite for the same reason. Γ is the set T defined above.

$$\delta(q, s) = \begin{cases} q' & \text{if } q \text{ is compound, has an element selector } s, \text{ and } q.s \text{ has type } q' \\ \texttt{invalid} & \text{if } q \text{ is compound and } q \text{ has no element selector } s \\ \texttt{invalid} & \text{if } q \text{ is a built-in type or } q = \texttt{invalid} \end{cases}.$$

$$\lambda(q) = \begin{cases} q & \text{if } q \text{ is a built-in type} \\ \texttt{compound} & \text{if } q \text{ is a compound type} \\ \texttt{invalid} & \text{if } q = \texttt{invalid} \end{cases}.$$

Finally, we discard all states that are not reachable from q_0. It should be obvious that these unreachable states have no influence on the classification function computed by the automaton but the arguments in proofs later in this document require all states to be reachable from q_0.

The following lemma shows that this construction is correct:

Lemma 1. $\psi_{M_A} = \phi_A$.

Proof. Consider any state $q \neq \texttt{invalid}$.

If q is a built-in type, then $\phi_q(\epsilon) = q$ and $\phi_q(\sigma) = \texttt{invalid}$ for all $\sigma \neq \epsilon$. We have $\delta^*(q,\epsilon) = q$ and $\lambda(q) = q$, so $\lambda(\delta^*(q,\epsilon)) = \phi_q(\epsilon)$. For $\sigma = s\sigma'$, we have

$$\delta^*(q,\sigma) = \delta^*(\delta(q,s),\sigma') = \delta^*(\texttt{invalid},\sigma') = \texttt{invalid}.$$

Since $\lambda(\texttt{invalid}) = \texttt{invalid}$, we once again obtain $\lambda(\delta^*(q,\sigma)) = \texttt{invalid} = \phi_q(\sigma)$. This shows that $\lambda(\delta^*(q,\sigma)) = \phi_q(\sigma)$ for every built-in type q and every selector sequence $\sigma \in \Sigma^*$.

If q is a compound type, then $\phi_q(\epsilon) = \texttt{compound}$. Since $\delta^*(q,\epsilon) = q$ and $\lambda(q) = \texttt{compound}$, we have $\phi_q(\epsilon) = \lambda(\delta^*(q,\epsilon))$. For any non-empty selector sequence $\sigma = s\sigma'$, we distinguish two cases:

If s is a selector of q, then

$$\phi_q(\sigma) = \phi_{q.s}(\sigma') = \lambda(\delta^*(q.s,\sigma')) = \lambda(\delta^*(\delta(q,s),\sigma')) = \lambda(\delta^*(q,\sigma)).$$

If s is not a selector of q, then $\phi_q(\sigma) = \texttt{invalid}$. However, $\delta(q,s) = \texttt{invalid}$ in this case, and $\delta^*(\texttt{invalid},\sigma') = \texttt{invalid}$. Thus,

$$\lambda(\delta^*(q,\sigma)) = \lambda(\delta^*(\delta(q,s),\sigma')) = \lambda(\delta^*(\texttt{invalid},\sigma')) = \lambda(\texttt{invalid}) = \texttt{invalid}.$$

This shows that $\phi_q(\sigma) = \lambda(\delta^*(q,\sigma))$ for every compound type q and every selector sequence $\sigma \in \Sigma^*$.

To finish the proof, we observe that, by definition,

$$\psi_{M_A}(\sigma) = \lambda(\delta^*(q_0,\sigma)) = \lambda(\delta^*(A,\sigma)) = \phi_A(\sigma). \qquad \square$$

Minimizing a Classification Automaton

We use the same minimization procedure for DFA that we used in class, with one difference: Instead of starting with only two equivalence classes consisting of accepting and non-accepting states, respectively, we start with $|\Gamma|$ equivalence classes $\{C_\gamma \mid \gamma \in \Gamma\}$ defined as $C_\gamma = \{q \in S \mid \lambda(q) = \gamma\}$. This is the same extension to the minimization procedure we used when minimizing the DFA of a scanner, which may have different types of accepting states. Again, the standard minimiziation procedure is just a special case of this more general minimization procedure after defining

$$\lambda(q) = \begin{cases} \texttt{accept} & \text{if } q \in F \\ \texttt{reject} & \text{if } q \notin F \end{cases}.$$

We describe this procedure in detail here and prove that it preserves the classification function computed by the automaton, since we need this in proofs later in this document.

As stated above, the procedure starts with a set of equivalence classes $\mathcal{C} = \{C_\gamma \mid \gamma \in \Gamma\}$, where $C_\gamma = \{q \in S \mid \lambda(q) = \gamma\}$. As long as there exists an equivalence class $C \in \mathcal{C}$, two states $q, q' \in C$, and a letter $s \in \Sigma$ such that $\delta(q,s) \in C'$ and $\delta(q',s) \in C''$, where $C' \neq C''$, we partition C into subclasses $C_1, \ldots, C_k$ such that

- For any class C_i and any two states $q, q' \in C_i$, $\delta(q,s)$ and $\delta(q',s)$ belong to the same class in $\mathcal{C}$.

- For two classes $C_i \neq C_j$ and any two states $q \in C_i$ and $q' \in C_j$, $\delta(q,s)$ and $\delta(q',s)$ belong to different classes in $\mathcal{C}$.

Then we replace C with $C_1, \ldots, C_k$ in $\mathcal{C}$.

The procedure terminates once there exists, for all $s \in \Sigma$ and every equivalence class $C \in \mathcal{C}$, an equivalence class $C' \in \mathcal{C}$ such that $\delta(q,s) \in C'$ for all $q \in C$. We define $\mu(M) = (\mathcal{C}, C_0, \Sigma, \Gamma, \delta_\mu, \lambda_\mu)$, where $q_0 \in C_0$, $\delta_\mu(C,s) = C'$ such that $\delta(q,s) \in C'$ for all $q \in C$, and $\lambda_\mu(C) = \lambda(q)$ for all $q \in C$. Once the algorithm terminates, there exists a class $C' \in \mathcal{C}$ for every pair $(C,s) \in \mathcal{C} \times \Sigma$ such that $\delta(q,s) \in C'$ for all $q \in C$. Thus, δ_μ is well defined. Similarly, every class $C \in \mathcal{C}$ satisfies $C \subseteq C_\gamma$ for some $\gamma \in \Gamma$. Thus, $\lambda(q) = \lambda(q')$ for all $q, q' \in C$, that is, λ_μ is well defined.

First we prove that minimization preserves the computed classification function.

Lemma 2. $\psi_{\mu(M)} = \psi_M$.

Proof. We prove by induction on $|\sigma|$ that every class $C \in \mathcal{C}$ satisfies $\delta^*(q,\sigma) \in \delta^*_\mu(C,\sigma)$ for all $q \in C$ and every string $\sigma \in \Sigma^*$. Thus, since $\lambda_\mu(C) = \lambda(q)$ for every class $C \in \mathcal{C}$ and all $q \in C$, we have

$$\psi_{\mu(M)}(\sigma) = \lambda_\mu(\delta^*_\mu(C_0,\sigma)) = \lambda(\delta^*(q_0,\sigma)) = \psi_M(\sigma)$$

for all $\sigma \in \Sigma^*$, that is, $\psi_{\mu(M)} = \psi_M$.

For $|\sigma| = 0$, we have $\sigma = \epsilon$. Thus, $\delta^*(q,\sigma) = q \in C$ for all $q \in C$.

For $|\sigma| > 0$, we have $\sigma = s\sigma'$ for some $s \in \Sigma$ and $\sigma' \in \Sigma^*$. By the termination condition of the algorithm, we have $\{\delta(q,s) \mid q \in C\} \subseteq C'$ for some $C' \in \mathcal{C}$. By the induction hypothesis, we have $\delta^*(q',\sigma') \subseteq \delta^*_\mu(C',\sigma')$ for all $q' \in C'$. Since $\delta(q,s) \in C'$ for all $q \in C'$, we have $\delta_\mu(C,s) = C'$ and, thus,

$$\delta^*(q,\sigma) = \delta^*(q,s\sigma') = \delta^*(\delta(q,s),\sigma') \subseteq \delta^*_\mu(C',\sigma') = \delta^*_\mu(\delta_\mu(C,s),\sigma') = \delta^*_\mu(C,s\sigma') = \delta^*_\mu(C,\sigma). \quad \square$$

The next lemma will be useful in proving that the minimized classification automaton for a given classification function is unique.

Lemma 3. *Every pair of equivalence classes $C \neq C'$ in $\mathcal{C}$ has a string $\sigma_{C,C'}$ such that $\lambda_\mu(\delta^*_\mu(C,\sigma_{C,C'})) \neq \lambda_\mu(\delta^*_\mu(C',\sigma_{C,C'}))$.*

Proof. We prove by induction on the number of splits of equivalence classes performed by the algorithm that there exists a string $\sigma_{C,C'}$ for every pair of equivalence classes $C \neq C'$ such that $\lambda(\delta^*(q,\sigma_{C,C'})) \neq \lambda(\delta^*(q',\sigma_{C,C'}))$ for all $q \in C$ and $q' \in C'$. Since $\lambda_\mu(\delta^*_\mu(C,\sigma_{C,C'})) = \lambda(\delta^*(q,\sigma_{C,C'}))$ for all $q \in C$ and $\lambda_\mu(\delta^*_\mu(C',\sigma_{C,C'})) = \lambda(\delta^*(q',\sigma_{C,C'}))$ for all $q' \in C'$, the lemma follows.

Initially, we have $\mathcal{C} = \{C_\gamma \mid \gamma \in \Gamma\}$. For any two states $q \in C_\gamma$ and $q' \in C_{\gamma'}$, where $\gamma \neq \gamma'$, we have

$$\lambda(\delta^*(q,\epsilon)) = \lambda(q) = \gamma \neq \gamma' = \lambda(q') = \lambda(\delta^*(q',\epsilon)).$$

Thus, we can choose $\sigma_{C_\gamma,C_{\gamma'}} = \epsilon$.

So assume we have a collection $\mathcal{C}$ of equivalence classes that satisfy the lemma and the algorithm splits a class $C \in C$ into subclasses $C_1, \ldots, C_k$. Then the resulting collection of equivalence classes is $\mathcal{C}' = \mathcal{C} \setminus \{C\} \cup \{C_1, \ldots, C_k\}$. Consider two classes $C' \neq C''$ in $\mathcal{C}'$.

If $\{C', C''\} \cap \{C_1, \ldots, C_k\} = \emptyset$, then $C', C'' \in \mathcal{C}$. Thus, there exists a string $\sigma_{C',C''}$ that satisfies $\lambda(\delta^*(q',\sigma_{C',C''})) \neq \lambda(\delta^*(q'',\sigma_{C',C''}))$ for all $q' \in C'$ and $q'' \in C''$.

If $C' \in \mathcal{C}$ and $C'' = C_i$ for some $1 \leq i \leq k$, then $C'' \subseteq C$, so every state $q'' \in C''$ satisfies $q'' \in C$. Thus, for every state $q' \in C'$, we have $\lambda(\delta^*(q',\sigma_{C',C})) \neq \lambda(\delta^*(q'',\sigma_{C',C}))$, that is, we can set $\sigma_{C',C''} = \sigma_{C',C}$.

Finally, if $C' = C_i$ and $C'' = C_j$ for some $1 \leq i < j \leq k$, then there exist two classes $C'_i \neq C'_j$ in $\mathcal{C}$ and a letter $s \in \Sigma$ such that $\delta(q,s) \in C'_i$ and $\delta(q',s) \in C'_j$ for all $q \in C_i$ and $q' \in C_j$. Thus,

$$\lambda(\delta^*(q,s\sigma_{C'_i,C'_j})) = \lambda(\delta^*(\delta(q,s),\sigma_{C'_i,C'_j})) \neq \lambda(\delta^*(\delta(q',s),\sigma_{C'_i,C'_j})) = \lambda(\delta^*(q',s\sigma_{C'_i,C'_j}))$$

for all $q \in C_i$ and $q' \in C_j$. Therefore, we can choose $\sigma_{C_i,C_j} = s\sigma_{C'_i,C'_j}$. $\square$

The Minimized Classification Automaton is Unique

We consider two classification automata $M_1 = (S_1, q_1, \Sigma, \Gamma, \delta_1, \lambda_1)$ and $M_2 = (S_2, q_2, \Sigma, \Gamma, \delta_2, \lambda_2)$ to be the same, written $M_1 \equiv M_2$, if there exists a bijection $\beta : S_1 \to S_2$ such that

(i) $\beta(q_1) = q_2$ (M_1 and M_2 have "the same start state"),
(ii) $\beta(\delta_1(q,s)) = \delta_2(\beta(q),s)$ for all $q \in S_1$ and $s \in \Sigma$ (M_1 and M_2 have "the same transition function"), and
(iii) $\lambda_2(\beta(q)) = \lambda_1(q)$ (the states of M_1 and M_2 have "the same labels").

Lemma 4. *M_1 and M_2 are two classification automata with $\psi_{M_1} = \psi_{M_2}$ if and only if $\mu(M_1) \equiv \mu(M_2)$.*

Proof. Let $\mu(M_1) = (S_1, q_1, \Sigma, \Gamma, \delta_1, \lambda_1)$ and $\mu(M_2) = (S_2, q_2, \Sigma, \Gamma, \delta_2, \lambda_2)$. We proved that $\psi_{M_1} = \psi_{\mu(M_1)}$ and $\psi_{M_2} = \psi_{\mu(M_2)}$. Thus, it suffices to prove that $\psi_{\mu(M_1)} = \psi_{\mu(M_2)}$ if and only if $\mu(M_1) \equiv \mu(M_2)$.

First assume that $\mu(M_1) \equiv \mu(M_2)$ and let β be the required bijection that proves this. Then observe that (ii) implies that $\beta(\delta_1^*(q,\sigma)) = \delta_2^*(\beta(q),\sigma)$ for all $q \in S_1$ and $\sigma \in \Sigma^*$. This can be shown by induction on $|\sigma|$. If $\sigma = \epsilon$, then $\beta(\delta_1^*(q,\epsilon)) = \beta(q) = \delta_2^*(\beta(q),\epsilon)$. If $\sigma = s\sigma'$, then

$$\begin{aligned}
\beta(\delta_1^*(q,\sigma)) &= \beta(\delta_1^*(q,s\sigma')) \\
&= \beta(\delta_1^*(\delta_1(q,s),\sigma')) \\
&= \delta_2^*(\beta(\delta_1(q,s)),\sigma') && \text{(by the induction hypothesis)} \\
&= \delta_2^*(\delta_2(\beta(q),s),\sigma') && \text{(by (ii))} \\
&= \delta_2^*(\beta(q),s\sigma') \\
&= \delta_2^*(\beta(q),\sigma).
\end{aligned}$$

This implies that

$$\psi_{\mu(M_1)}(\sigma) = \lambda_1(\delta_1^*(q_1,\sigma)) \overset{\text{(iii)}}{=} \lambda_2(\beta(\delta_1^*(q_1,\sigma))) = \lambda_2(\delta_2^*(\beta(q_1),\sigma)) \overset{\text{(i)}}{=} \lambda_2(\delta_2^*(q_2,\sigma)) = \psi_{\mu(M_2)}(\sigma).$$

Since this holds for all $\sigma \in \Sigma^*$, this shows that $\psi_{\mu(M_1)} = \psi_{\mu(M_2)}$.

To prove that $\mu(M_1) \equiv \mu(M_2)$ if $\psi_{\mu(M_1)} = \psi_{\mu(M_2)}$, we construct two injective mappings $\beta_1 : S_1 \to S_2$ and $\beta_2 : S_2 \to S_1$. Thus, β_1 is a bijection between S_1 and S_2. Then we prove that β_1 satisfies conditions (i)–(iii).

Recall that we consider only automata where all states are reachable from the start state. (I leave it as an exercise to verify that, if all states of an automaton M are reachable from M's start state, then the same is true for $\mu(M)$.) Thus, there exists a string σ_q for all $q \in S_1$ such that $\delta_1^*(q_1,\sigma_q) = q$. Clearly, $\sigma_q \neq \sigma_{q'}$ if $q \neq q'$. We define a string $\tau_{q'}$ for all $q' \in S_2$ analogously. Then we define $\beta_1(q) = \delta_2(q_2,\sigma_q)$ for all $q \in S_1$ and $\beta_2(q') = \delta_1(q_1,\tau_{q'})$ for all $q' \in S_2$.

We prove that β_1 is injective. An analogous argument shows that β_2 is injective. So assume for the sake of contradiction that there exist two states $q \neq q'$ in S_1 such that $\beta_1(q) = \beta_1(q')$. Then let $\sigma_{q,q'} \in \Sigma^*$ such that $\lambda_1(\delta_1^*(q,\sigma_{q,q'})) \neq \lambda_1(\delta_1^*(q',\sigma_{q'q'}))$. By Lemma 3, such a string $\sigma_{q'q'}$ exists. Then

$$\begin{aligned}
\psi_{\mu(M_1)}(\sigma_q\sigma_{q,q'}) &= \lambda_1(\delta_1^*(q_1,\sigma_q\sigma_{q,q'})) = \lambda_1(\delta_1^*(\delta_1^*(q_1,\sigma_q),\sigma_{q,q'})) = \lambda_1(\delta_1^*(q,\sigma_{q,q'})) \\
&\neq \lambda_1(\delta_1^*(q',\sigma_{q,q'})) = \lambda_1(\delta_1^*(\delta_1^*(q_1,\sigma_{q'}),\sigma_{q,q'})) = \lambda_1(\delta_1^*(q_1,\sigma_{q'}\sigma_{q,q'})) \\
&= \psi_{\mu(M_1)}(\sigma_{q'}\sigma_{q,q'}).
\end{aligned}$$

On the other hand, by the definition of β_1, we have $\delta_2^*(q_2,\sigma_q) = \beta_1(q) = \beta_1(q') = \delta_2^*(q_2,\sigma_{q'})$. Thus,

$$\begin{aligned}
\psi_{\mu(M_2)}(\sigma_q\sigma_{q,q'}) &= \lambda_2(\delta_2^*(q_2,\sigma_q\sigma_{q,q'})) = \lambda_2(\delta_2^*(\delta_2^*(q_2,\sigma_q),\sigma_{q,q'})) \\
&= \lambda_2(\delta_2^*(\delta_2^*(q_2,\sigma_{q'}),\sigma_{q,q'})) = \lambda_2(\delta_2^*(q_2,\sigma_{q'}\sigma_{q,q'})) = \psi_{\mu(M_2)}(\sigma_{q'}\sigma_{q,q'}).
\end{aligned}$$

This is a contradiction because $\psi_{\mu(M_1)} = \psi_{\mu(M_2)}$.

Since β_1 and β_2 are both injective, we have $|S_1| = |S_2|$. Thus, since β_1 is injective, it is a bijection from S_1 to S_2. Next we show that β_1 satisfies properties (i)–(iii).

Property (i) is obvious: $\sigma_{q_1} = \epsilon$. Thus, $\beta_1(q_1) = \delta_2^*(q_2, \epsilon) = q_2$, as required.

To prove (ii), assume for the sake of contradiction that there exists a pair $(q,s) \in S_1 \times \Sigma$ such that $\beta_1(\delta_1(q,s)) \neq \delta_2(\beta_1(q),s)$. Then let $q' = \delta_1(q,s)$, $q'' = \delta_2(\beta_1(q),s) = \delta_2(\delta_2^*(q_2,\sigma_q),s) = \delta_2^*(q_2,\sigma_q s)$, and $q''' = \beta_1(q') = \delta_2^*(q_2,\sigma_{q'})$. Since $q'' \neq q'''$, there exists a string $\tau_{q'',q'''}$ such that $\lambda_2(\delta_2^*(q'',\tau_{q'',q'''})) \neq \lambda_2(\delta_2^*(q''',\tau_{q'',q'''}))$. Thus,

$$\begin{aligned}\psi_{\mu(M_2)}(\sigma_q s \tau_{q'',q'''}) &= \lambda_2(\delta_2^*(q_2,\sigma_q s \tau_{q'',q'''})) = \lambda_2(\delta_2^*(q'',\tau_{q'',q'''})) \\ &\neq \lambda_2(\delta_2^*(q''',\tau_{q'',q'''})) = \lambda_2(\delta_2^*(q_2,\sigma_{q'}\tau_{q'',q'''})) = \psi_{\mu(M_2)}(\sigma_{q'}\tau_{q'',q'''}).\end{aligned}$$

On the other hand, since $q' = \delta_1(q,s)$, we have

$$\begin{aligned}\psi_{\mu(M_1)}(\sigma_q s \tau_{q'',q'''}) &= \lambda_1(\delta_1^*(q_1,\sigma_q s \tau_{q'',q'''})) = \lambda_1(\delta_1^*(q',\tau_{q'',q'''})) \\ &= \lambda_1(\delta_1^*(q_1,\sigma_{q'}\tau_{q'',q'''})) = \psi_{\mu(M_1)}(\sigma_{q'}\tau_{q'',q'''}),\end{aligned}$$

a contradiction because $\psi_{\mu(M_1)} = \psi_{\mu(M_2)}$.

Finally, to prove that β_1 satisfies property (iii), assume there exists a state $q \in S_1$ such that $\lambda_1(q) \neq \lambda_2(\beta_1(q))$. Then

$$\psi_{\mu(M_1)}(\sigma_q) = \lambda_1(\delta_1^*(q_1,\sigma_q)) = \lambda_1(q) \neq \lambda_2(\beta_1(q)) = \lambda_2(\delta_2^*(q_2,\sigma_q)) = \psi_{\mu(M_2)}(\sigma_q),$$

again a contradiction because $\psi_{\mu(M_1)} = \psi_{\mu(M_2)}$. □

Deciding Whether Two Automata are the Same

Given two automata $M_1 = (S_1, q_1, \Sigma, \Gamma, \delta_1, \lambda_1)$ and $M_2 = (S_2, q_2, \Sigma, \Gamma, \delta_2, \lambda_2)$, we have $M_1 \not\equiv M_2$ if $|S_1| \neq |S_2|$. Thus, this is the first condition we test. If $|S_1| = |S_2|$, we attempt to construct a bijection β as above and then test whether it has the desired properties. Specifically, if $n = |S_1| = |S_2|$, we construct two bijections $\nu_1 : S_1 \to [n]$ and $\nu_2 : S_2 \to [n]$ numbering the states in S_1 and S_2. The inverses of these two bijections are ν_1^{-1} and ν_2^{-1}. The construction will ensure that $M_1 \equiv M_2$ if and only if $\delta_1' = \delta_2'$ and $\lambda_1' = \lambda_2'$, where $\delta_i'(x,s) = \nu_i(\delta_i(\nu_i^{-1}(x),s))$ and $\lambda_i'(x) = \lambda_i(\nu_i^{-1}(x))$ for all $i \in \{1,2\}$, $x \in [n]$, and $s \in \Sigma$.

To construct the numbering ν_i for $i \in \{1,2\}$, we run a lexicographic DFS in M_i starting from q_i. "Lexicographic" means that the search explores the out-edges of each state by increasing labels according to some arbitrary but fixed ordering defined on Σ. ν_i now numbers the states in S_i in the order they are discovered by this search. In particular, $\nu_1(q_1) = 1 = \nu_2(q_2)$.

Lemma 5. *$M_1 \equiv M_2$ if and only if $|S_1| = |S_2|$, $\delta_1' = \delta_2'$, and $\lambda_1' = \lambda_2'$.*

Proof. As already observed, $M_1 \not\equiv M_2$ if $|S_1| \neq |S_2|$. So assume from here on that $|S_1| = |S_2|$. First assume that $\delta_1' = \delta_2'$ and $\lambda_1' = \lambda_2'$. We define a bijection $\beta : S_1 \to S_2$ as $\beta = \nu_2^{-1}\nu_1$. This is indeed a bijection because ν_1 and ν_2 are both bijections.

Since $\nu_1(q_1) = 1 = \nu_2(q_2)$, we have $\beta(q_1) = \nu_2^{-2}(\nu_1(q_1)) = q_2$, so β satisfies condition (i). It remains

to prove that β satisfies conditions (ii) and (iii).

(ii)
$$\begin{aligned}
\beta(\delta_1(q,s)) &= \nu_2^{-1}(\nu_1(\delta_1(q,s))) && \text{(because } \beta = \nu_2^{-1}\nu_1\text{)}\\
&= \nu_2^{-1}(\nu_1(\delta_1(\nu_1^{-1}(\nu_1(q)),s))) && \text{(because } \nu_1^{-1}\nu_1 = \text{id}\text{)}\\
&= \nu_2^{-1}(\delta_1'(\nu_1(q),s)) && \text{(because } \delta_1'(x,s) = \nu_1(\delta_1(\nu_1^{-1}(x),s))\text{)}\\
&= \nu_2^{-1}(\delta_2'(\nu_1(q),s)) && \text{(because } \delta_1' = \delta_2'\text{)}\\
&= \nu_2^{-1}(\nu_2(\delta_2(\nu_2^{-1}(\nu_1(q)),s))) && \text{(because } \delta_2'(x,s) = \nu_2(\delta_2(\nu_2^{-1}(x),s))\text{)}\\
&= \delta_2(\beta(q),s). && \text{(because } \nu_2^{-1}\nu_2 = \text{id and } \nu_2^{-1}\nu_1 = \beta\text{)}
\end{aligned}$$

(iii)
$$\begin{aligned}
\lambda_2(\beta(q)) &= \lambda_2(\nu_2^{-1}(\nu_1(q))) && \text{(because } \beta = \nu_2^{-1}\nu_1\text{)}\\
&= \lambda_2'(\nu_1(q)) && \text{(because } \lambda_2'(x) = \lambda_2(\nu_2^{-1}(x))\text{)}\\
&= \lambda_1'(\nu_1(q)) && \text{(because } \lambda_1' = \lambda_2'\text{)}\\
&= \lambda_1(\nu_1^{-1}(\nu_1(q))) && \text{(because } \lambda_1'(x) = \lambda_1(\nu_1^{-1}(x))\text{)}\\
&= \lambda_1(q). && \text{(because } \nu_1^{-1}\nu_1 = \text{id}\text{)}
\end{aligned}$$

Next assume there exists a bijection β that satisfies conditions (i)–(iii). For every state $q \in S_1$, let σ_q be the *characteristic string* of q defined as the lexicographically least string such that $\delta_1^*(q_1,\sigma_q) = q$. We sort the states in S_1 by their characteristic strings and define $\nu_1'(q)$ to be the position of q in this ordering. Define $\nu_2'(q)$ analogously for the states in S_2 and let $\nu_2''(q) = \nu_1'(\beta^{-1}(q))$ for all $q \in S_2$. We prove in the following two lemmas that $\nu_1 = \nu_1'$ and $\nu_2 = \nu_2' = \nu_2''$. For now, assume this is true. Then we have

$$\begin{aligned}
\delta_1'(x,s) &= \nu_1(\delta_1(\nu_1^{-1}(x),s)) && \text{(by the definition of } \delta_1'\text{)}\\
&= \nu_2(\beta(\delta_1(\nu_1^{-1}(x),s))) && \text{(because } \nu_2 = \nu_2'' = \nu_1'\beta^{-1} = \nu_1\beta^{-1}\\
& && \text{and thus } \nu_2\beta = \nu_1\beta^{-1}\beta = \nu_1\text{.)}\\
&= \nu_2(\delta_2(\beta(\nu_1^{-1}(x)),s)) && \text{(because } \beta(\delta_1(q,s)) = \delta_2(\beta(q),s)\text{)}\\
&= \nu_2(\delta_2(\nu_2^{-1}(x),s)) && \text{(because } \nu_2 = \nu_1\beta^{-1} \text{ and thus } \nu_2^{-1} = (\nu_1\beta^{-1})^{-1} = \beta\,\nu_1^{-1}\text{)}\\
&= \delta_2'(x,s). && \text{(by the definition of } \delta_2'\text{)}
\end{aligned}$$

Thus, $\delta_1' = \delta_2'$.

Similarly,

$$\begin{aligned}
\lambda_2'(x) &= \lambda_2(\nu_2^{-1}(x)) && \text{(by the definition of } \lambda_2'\text{)}\\
&= \lambda_2(\beta(\nu_1^{-1}(x))) && \text{(because } \nu_2 = \nu_1\beta^{-1} \text{ and thus } \nu_2^{-1} = (\nu_1\beta^{-1})^{-1} = \beta\,\nu_1^{-1}\text{)}\\
&= \lambda_1(\nu_1^{-1}(x)) && \text{(because } \lambda_1(q) = \lambda_2(\beta(q)) \text{ for all } q \in S_1\text{)}\\
&= \lambda_1'(x). && \text{(by the definition of } \lambda_1'\text{)}
\end{aligned}$$

□

Lemma 6. *$\nu_1 = \nu_1'$ and $\nu_2 = \nu_2'$.*

Proof. We prove that $\nu_1 = \nu_1'$. The proof that $\nu_2 = \nu_2'$ is analogous.

First we define some notation. For a given string $\sigma = s_1 \cdots s_{|\sigma|} \in \Sigma^*$, we use $\pi(\sigma)$ to denote the "path followed by σ". Formally, $\pi(\sigma) = \langle p_0, \ldots, p_{|\sigma|}\rangle$, where $p_0 = q_1$ and, for $1 \le j \le |\sigma|$, $p_j = \delta_i(p_{j-1}, s_j)$.

Conversely, for a path π in M_i, we use $\sigma(\pi)$ to refer to the "string labelling π", that is, if $\pi = \langle p_0, \ldots, p_{|\pi|}\rangle$ (we use $|\pi|$ to denote the number of edges in π), then $\sigma(\pi) = s_1 \cdots s_{|\pi|}$ such that $\delta_i(p_{j-1}, s_j) = p_j$ for all $1 \le j \le |\pi|$.

The definitions of $\pi_i(q,\sigma)$ and $\sigma(\pi)$ are, in a sense, inverses of each other because $\sigma(\pi(\sigma)) = \sigma$ and, if the start vertex of π is q_1, then $\pi(\sigma(\pi)) = \pi$.

Now assume that $\nu_1 \neq \nu_1'$. Then there exists a state $q \in S$ such that $\nu_1(q) \neq \nu_1'(q)$. We choose this state q so that $\nu_1'(q)$ is minimized. Let q' be the state such that $\nu_1(q') = \nu_1'(q) \neq \nu_1'(q')$. Then $q \neq q'$ and, by the choice of q, $\nu_1'(q') > \nu_1'(q)$. Since every vertex q'' with $\nu_1'(q'') < \nu_1'(q)$ satisfies $\nu_1(q'') = \nu_1'(q'')$, by the choice of q, we also have $\nu_1(q) > \nu_1'(q)$.

If $q = q_1$, then $\nu_1(q) = 1$ and $\sigma_q = \epsilon < \sigma_{q'}$ for all $q' \neq q$. Thus, $\nu_1'(q) = 1$. Since we assume that $\nu_1(q) \neq \nu_1'(q)$, this shows that $q \neq q_1$ and, thus, $\sigma_q \neq \epsilon$. Therefore, $|\pi(\sigma_q)| \geq 1$ and q has a predecessor p in $\pi(\sigma_q)$. Let $\sigma_q = \sigma_p' s$. Then $\delta_1^*(q_1, \sigma_p') = p$ and thus, by the choice of σ_p, we have $\sigma_p \leq \sigma_p'$. Conversely, $\delta_1^*(q_1, \sigma_p s) = q$. Thus, by the choice of σ_q, we have $\sigma_p' s = \sigma_q \leq \sigma_p s$, that is, $\sigma_p' \leq \sigma_p$. This shows that $\sigma_p = \sigma_p'$ and $\sigma_q = \sigma_p s$. This in turn implies that $\sigma_p < \sigma_q$ and, thus, $\nu_1'(p) < \nu_1'(q)$. By the choice of q, we thus have $\nu_1(p) = \nu_1'(p)$ and, therefore, $\nu_1(p) = \nu_1'(p) < \nu_1'(q) = \nu_1(q') < \nu_1(q)$, that is, the lexicographic DFS of M_1 discovers q after p, and q' between p and q.

In particular, due to the existence of the edge (p,q), this implies that q is a descendant of p in the tree T constructed by the lexicographic DFS and, since DFS numbers all descendant nodes of p consecutively, q' is also a descendant of p in T.

If q is a child of p, then let π be the path from p to q' in T. Since q' is visited before q, (p,q) is not the first edge in π. Since we use *lexicographic* DFS, the first edge (p,q'') in π has a smaller label than (p,q). In particular, $\sigma_{q'} \leq \sigma_p \sigma(\pi) < \sigma_p \sigma(\langle p,q\rangle) = \sigma_q$. This implies that $\nu_1'(q') < \nu_1'(q)$, a contradiction.

If q is not a child of p, then let π' be the path from p to q in T. The first edge (p,q''') on this path must have a label less than that of (p,q) because otherwise the edge (p,q) would be explored first and q would become a child of p. Thus, $\sigma_p \sigma(\pi') < \sigma_p \sigma(\langle p,q\rangle) = \sigma_q$. Since $\delta_1^*(q_1, \sigma_p \sigma(\pi')) = q$, this contradicts the choice of σ_q.

This proves that $\nu_1 = \nu_1'$. □

Lemma 7. $\nu_2' = \nu_2''$.

Proof. First we prove that $\sigma_q = \tau_{\beta(q)}$ for all $q \in S_1$, where $\tau_{q'}$ is the characteristic string of the state $q' \in S_2$. Recall that (ii) implies that $\beta(\delta_1^*(q,\sigma)) = \delta_2^*(\beta(q),\sigma)$ for all $q \in S_1$ and $\sigma \in \Sigma^*$. This was shown as part of the proof of Lemma 4. Since $\beta(q_1) = q_2$, by (i), this implies that $\delta_1^*(q_1,\sigma) = q$ if and only if $\delta_2^*(q_2,\sigma) = \beta(q)$. Therefore, σ_q and $\tau_{\beta(q)}$ are chosen from the same set of candidate strings and, thus, $\sigma_q = \tau_{\beta(q)}$.

For two states $q_1', q_2' \in S_2$, we have $\nu_2''(q_1') < \nu_2''(q_2')$ if and only if $\nu_1'(\beta^{-1}(q_1')) < \nu_1'(\beta^{-1}(q_2'))$, by the definition of ν_2''. Since $\sigma_{\beta^{-1}(q_1')} = \tau_{q_1'}$ and $\sigma_{\beta^{-1}(q_2')} = \tau_{q_2'}$, this holds if and only if $\nu_2'(q_1') < \nu_2'(q_2')$. Since both ν_2' and ν_2'' are bijections from S_2 to $[n]$, $\nu_2'(q_1') < \nu_2'(q_2') \Leftrightarrow \nu_2''(q_1') < \nu_2''(q_2'')$ for all $q_1', q_2' \in S_2$ implies that $\nu_2'(q') = \nu_2''(q')$ for all $q' \in S_2$, that is, $\nu_2' = \nu_2''$. □