Assignments:
Assignment 3 out today.

Previously:
Computational models for the regular languages: DFAs, NFAs, regular expressions.

Today:
How do we find the minimal DFA for a language?
This is the last topic in the first part of the course!

Then:
Tuesday we’ll review for Exam 1.

Not in Sipser
Reading from Hopcroft et al. is on the course website.
User name and password are the same as for assignment solutions.
Why are we adding it?

DFA minimization: the idea
Background:
A regular language can be accepted by many DFAs
Questions:
Is there a unique simplest DFA for a regular language?
If so, can we construct it?
The idea
Yes, there is a unique minimal DFA – and we can construct it!

“Minimal”? Minimal number of states.
“Unique”? Unique up to renaming of states. I.e., has same shape. Isomorphic.

Distinguishable strings
Let \( L \) be any language in \( \Sigma^* \). For any two strings \( x \) and \( y \in \Sigma^* \), we can apply the indistinguishability relation \( I_L \), such that \( x I_L y \) if and only if for any \( z \in \Sigma^* \), either both \( xz \) and \( yz \) are in \( L \), or both \( xz \) and \( yz \) are in \( \bar{L} \) (the complement of \( L \)).

States and strings
We can think of a state in an automaton as representing a certain minimal amount of information we need to remember about the string we are processing.

Each state can be identified by the set of strings that cause us to remember exactly that information,
I.e., the set of strings that cause us to be at that state in the recognition algorithm
A “state” is an abstraction anyway, so why not think of a state as a set of strings?

Distinguishable states
If we see states as sets of strings, it’s easy to see that distinguishability applies to states as well.

States \( p \) and \( q \) are indistinguishable iff for all strings \( z \), either both \( \hat{\delta}(p,z) \in F \) and \( \hat{\delta}(q,z) \in F \) or \( \hat{\delta}(p,z) \notin F \) and \( \hat{\delta}(q,z) \notin F \)

\[
\begin{align*}
\text{or} \quad p & \quad z \\
q & \quad z
\end{align*}
\]
Example (very simple)
Consider:

$p$ is distinguishable from $q$ and $r$ because $p$ is non-accepting and $q$ and $r$ are accepting.
- So appending the empty string to the sets of strings the states represent leads to a non-accepting state in one case and an accepting state in the other.

Can we distinguish $q$ from $r$?
- No string beginning with 0 works, because both states go to $p$, and therefore any string of the form $0x$ takes $q$ and $r$ to the same state.
- No string beginning with 1 works.

Technically, $\delta(q, 1) = r$ and $\delta(r, 1) = q$ are not distinguishable.

What happens is that, starting in either $q$ or $r$, as long as we have inputs 1, we are in one of the accepting states, and when a 0 is read, we go to the same state ($p$) and then regardless of input, the same states forever after.

Equivalence classes
- For strings: $\Sigma^*$ can be partitioned into sets of indistinguishable strings
- For states: The states of an automaton $Q$ can be partitioned into sets of indistinguishable states
- This defines equivalence classes over $\Sigma^*$ or $Q$
  - We will talk about equivalence classes in general, since states can be viewed as sets of strings.

Myhill–Nerode Theorem
$L$ is a regular language if and only if the set of equivalence classes is finite

Example: $L = \{x \in \{1,0\}^* | x$ ends with $10\}$
- Consider 3 strings: $\varepsilon$, 1, and 10
- Any two of these strings are distinguishable with respect to $L$
  - The string $\varepsilon$ distinguishes $\varepsilon$ and 10 and also 1 and 10
  - The string 0 distinguishes $\varepsilon$ and 1
So the three equivalence classes \([\varepsilon], [1], \text{ and } [10]\) are distinct

Any string \(y\) is indistinguishable from one of these strings
- If \(y\) ends in 10 it is indistinguishable from 10
- If \(y\) ends in 1 it is indistinguishable from 1
- Otherwise (if \(y = \varepsilon, y = 0,\) or \(y\) ends with 00), \(y\) is indistinguishable from \(\varepsilon\)

The decision made for any input string depends only on its last two symbols
- no need to distinguish between one string ending in 10 and any other string ending with 10, or between any two strings ending in 01
- So the equivalence classes for this automaton are

\[
(0 \cup 1)^*1 \\
(0 \cup 1)^*0 \\
(0 \cup 1)^*10
\]

Based on the need to “remember” only the final two symbols, could build the following FA

Once the FA has seen two symbols, it cycles back and forth among 4 states “remembering” the last two symbols it saw

Can this FA be simplified?

• Consider states 0 and 00
  - Neither is an accepting state
  - Both need at least one more symbol to have a string in \(L\)
  - Rows in the table look exactly alike
• Consider strings that cause the FA to be in states 0 and 00 respectively
  - Do not need to be distinguished now, because neither state is accepting
  - Cannot be distinguished one symbol later, because the FA is in the same state
• Therefore these two states are not distinguishable
• The two can be merged
Merging states

- **State A**: Merge 0 and 00
- **State B**: Merge 1, 01, 11

<table>
<thead>
<tr>
<th>state</th>
<th>input</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>00</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>00</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>01</td>
<td></td>
</tr>
</tbody>
</table>

**New FA with 4 states**

- 0 and 00 are now **A**
- 1, 11, and 01 are now **B**
- In the new table, rows for $\varepsilon$ and **A** are the same (i.e., 0 and 1)
  - Neither is accepting
  - **Indistinguishable**
- Merge $\varepsilon$ into **A**
- No further reductions possible so this is the minimal FA

**Old FA with 7 states**

**Constructing the minimum-state DFA**

For each group of indistinguishable states, pick a “representative”.

Note a group can be large, e.g., $q_1$, $q_2$, ..., $q_k$ if all pairs are indistinguishable.

Indistinguishability is transitive (why?), so indistinguishability partitions states.

If $p$ is a representative, and $\delta(p, a) = q$, in minimum-state DFA the transition from $p$ on $a$ is the representative of $q$’s group (or to $q$ itself, if $q$ is either alone in a group or a representative).

*Those seven states were just three states in a trench coat (with a broom for a hand).*
Start state is representative of the original start state
Accepting states are representatives of groups of accepting states.

Notice we could not have a “mixed” (accepting & non-accepting) group. (Why?)
Delete any state that is not reachable from the start state

Formal algorithm

Build table to compare each unordered pair of distinct states $p, q$.

Each table entry has
- a “mark” as to whether $p$ and $q$ are known to be not equivalent, and
- a list of entries, recording dependencies: “If this entry is later marked, also mark these.”

Algorithm

- **Key idea**: find states $p$ and $q$ that are distinguishable because there is some input $w$ that takes exactly one of $p$ and $q$ to an accepting state.
- **Basis**: any non-accepting state is distinguishable from any accepting state
- **Induction**: $p$ and $q$ are distinguishable if there is some input symbol $a$ such that $\delta(p, a)$ is distinguishable from $\delta(q, a)$.
  - All other pairs of states are indistinguishable, and can be merged into one state.

Example

For the DFA given earlier, $p$ is in a group by itself; $\{q, r\}$ is the other group.
Algorithm

1. Initialize all entries as unmarked and with no dependencies
2. Mark all pairs of a final and non-final state
3. For each unmarked pair \( p, q \) and input symbol \( a \):
   - Let \( r = \delta(p, a) \), \( s = \delta(q, a) \).
   - If \((r, s)\) unmarked, add \((p, q)\) to \((r, s)\)'s dependencies,
   - Otherwise mark \((p, q)\), and recursively mark all dependencies of newly marked entries.
4. Coalesce unmarked pairs of states
5. Delete inaccessible states

Example

1. Initialize table entries:
   - Unmarked, empty list
2. Mark pairs of final & non-final states
3. For each unmarked pair & symbol, …
3. For each unmarked pair & symbol, ...

\[ \delta(d,0) \text{ is distinguishable from } \delta(a,0) \]

Mark \( (a,d) \)

3. For each unmarked pair & symbol, ...

\[ \delta(e,0) =? \delta(a,0) \]
\[ \delta(e,1) =? \delta(a,1) \]

Mark \( (a,e) \) as distinguishable
3. For each unmarked pair & symbol, ...

- Add (a,e) to (h,b)'s dependencies

- Add (g,a) to (g,b)'s and (f,e)'s dependencies

- Add (g,a) to (g,b)'s and (f,e)'s dependencies

- Add (g,a) to (g,b)'s and (f,e)'s dependencies
3. For each unmarked pair & symbol, ...

\[
\delta(d,0) = c \text{ (final state)} \\
\delta(b,0) = g \text{ (non-final state)} \\
\text{Distinguishable}
\]

3. For each unmarked pair & symbol, ...

\[
\text{Distinguishable}
\]

3. For each unmarked pair & symbol, ...

\[
\text{Distinguishable}
\]
4. Coalesce unmarked pairs of states
   \[ a \equiv e \]
   \[ b \equiv h \]
   \[ d \equiv f \]

5. Delete unreachable states

Final minimal FA

\[ (e,h) \] – distinguishable
\[ (f,g) \] – distinguishable
\[ (f,h) \] – distinguishable
\[ (g,h) \] – distinguishable

None.
NFA minimization

The minimization algorithm doesn’t find a unique minimal NFA.

More importantly, in general, there is no unique minimal NFA.

Example NFAs for $0^+$:

Both minimal, but not isomorphic.

decision properties of regular languages

Given (a representation, e.g., RE, FA, of) a regular language $L$, what can we tell about $L$?

Since there are algorithms to convert between any two representations, we can choose the representation that makes the test easiest.

Membership

Is string $w$ in regular language $L$?

Choose DFA representation for $L$.

Simulate the DFA on input $w$.

Emptiness

Is $L = \emptyset$?

Use DFA representation.

Use a graph-reachability algorithm to test if at least one accepting state is reachable from the start state.

Finiteness

Is $L$ a finite language?

Note every finite language is regular, but a regular language is not necessarily finite.

DFA method:

Given a DFA for $L$, eliminate all states that are not reachable from the start state and all states that do not reach an accepting state.

Test if there are any cycles in the remaining DFA; if so, $L$ is infinite, if not, then $L$ is finite.

RE method

Almost, we can look for a $*$ in the RE and say its language is infinite if there is one, finite if not. However, there are exceptions, e.g. $0\varepsilon^*1$ or $0^*\emptyset$. Thus:

1. Find sub-expressions equivalent to $\emptyset$ by:
   - (Basis) $\emptyset$ is; $\varepsilon$ and $a$ are not.
   - (Induction) $E \cup F$ is iff both $E$ and $F$ are; $EF$ is if either $E$ or $F$ is; $E^*$ never is.

2. Eliminate sub-expressions equivalent to $\emptyset$ by:
   - Replace $E \cup F$ or $FuE$ by $F$ whenever $E$ is and $F$ isn’t.
   - Replace $E^*$ by $\varepsilon$ whenever $E$ is equivalent to $\emptyset$. 

\[ \emptyset \]
3. Now, find sub-expressions that are equivalent to $\varepsilon$ by:
   - (Basis) $\varepsilon$ is; $a$ isn’t.
   - (Induction) $E\cup F$ is iff both $E$ and $F$ are; ditto $EF$; $E^*$ is iff $E$ is.

- Now, we can tell if $L(R)$ is infinite by looking for a sub-expression $E^*$ such that $E$ is not equivalent to $\varepsilon$.

---

**Notes on FA minimization algorithm**

Order of selecting state pairs was arbitrary.
- All orders give same ultimate result.
- But, may record more or fewer dependencies.
- Choosing states by working backwards from known non-equivalent states produces fewest dependencies.

This algorithm: Huffman (1954), Moore (1956).

$O(n^2)$ time.
- Constant work per entry: initial mark test and possibly later chasing of its dependencies.
- More efficient algorithms exist, e.g., Hopcroft (1971).

---

**Correctness of the minimization algorithm**

Why is new DFA no larger than old DFA?
- Only removes states, never introduces new states.
- Obvious.

Why is new DFA equivalent to old DFA?
- Only identify states that provably have same behavior.
- Could prove $x \in L(M) \iff x \in L(M')$ by inductions on derivations.

Why is “minimal” DFA unique (up to isomorphism)?
- Depends on the uniqueness of minimal equivalence classes of strings in the language.
Why the minimization algorithm can’t be beaten

Suppose we have a DFA $A$, and we minimize it to construct a DFA $M$. But there is another DFA $N$ that accepts the same language as $A$ and $M$, yet has fewer states than $M$.

Proof by contradiction that this can’t happen:

- Run the state-distinguishability process on the states of $M$ and $N$ together.
- Start states of $M$ and $N$ are indistinguishable because $L(M) = L(N)$.
- If {p, q} are indistinguishable, then their successors on any one input symbol are also indistinguishable.
- Thus, since neither $M$ nor $N$ could have an inaccessible state, every state of $M$ is indistinguishable from at least one state of $N$.

Since $N$ has fewer states than $M$, there are two states of $M$ that are indistinguishable from the same state of $N$, and therefore indistinguishable from each other.

But $M$ was designed so that all its states are distinguishable from each other.

We have a contradiction, so the assumption that $N$ exists is wrong, and $M$ in fact has as few states as any equivalent DFA for $A$.

In fact (stronger), there must be a one-to-one correspondence between the states of any other minimum-state $N$ and the DFA $M$, showing that the minimum-state DFA for $A$ is unique up to renaming of the states.