COMP330 Notes

Chany Ahn

September 2021 - December 2021

Contents

1 Logic

1.1 Equivalence Relations

A binary relation R on a set X is a subset of $X \times X$. Notation: $(a, b) \in \mathbb{R}$ or aRb. R must satisfy:

1. Reflexivity: $\forall x \in X, xRx$.
2. Symmetry: $\forall x, y \in X, xRy \Rightarrow yRx$.
3. Transitivity: $\forall x, y, z \in X, xRy, yRz \Rightarrow xRz$.

1.2 Partial Order

This is an abstraction of $\leq$. A binary relation R is a partial order of X if:

1. $\forall x \in X, xRx$.
2. $\forall x, y \in X, xRy, yRx \Rightarrow x = y$ (Antisymmetry).
3. $\forall x, y, z \in X, xRy, yRz \Rightarrow xRz$.

If every pair of elements can be compared, then we have a *total order* (aka *linear order*).

1.3 Well-Founded Orders

A parital order $\leq$ on S is well founded if every non-empty subset $U \subseteq S$ has a minimal element. We say $u \in U$ is minimal if there is nothing else in U strictly less than u.

Theorem 1. *The principle of induction can be used if and only if the order is well-founded.*

2 Deterministic Finite Automata

Definition 1. *Some useful definitions:*

- Σ*: A set of letters (or an alphabet).*
- Σ^**: Set of all words that can be made by the alphabet* Σ*.*
- $L \subseteq \Sigma^*$*: A language.*

Definition 2. *A deterministic finite automaton (DFA) is a 4-tuple:*

- S*: A finite set of states.*
- $s_0 \in S$*: The start state.*
- $\delta : S \times \Sigma \to S$*: Transition function.*
- $F \subseteq S$*: Accepting states.*

Definition 3. *A languages that is recognized by a DFA is called a regular language.*

Understand the examples from class!

3 Nondeterministic Finite Automata

Definition 4. *A nondeterministic finite automata is a 4-tuple:*

- Q*: A finite set of states.*
- $Q_0 \subseteq Q$*: A set of start states (*$Q_0 \neq \varnothing$*).*
- $F \subseteq Q$*: A set of accepting states.*
- Δ*: A transition relation.*
  - $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$
  - $\Delta(q, a)$*: all the places the machine can go to if it is in* q *and reads an* a*.*

Definition 5. *An NFA with* ε*-moves is a machine that can jump to a new state without reading a letter.*

Theorem 2. *The language accepted by an NFA is a regular language.*

Definition 6.

$$\delta^* : S \times \Sigma^* \to S \tag{1}$$
$$\Delta^* : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q) \tag{2}$$

These equalities follow from the above definition for (1) which is defined inductively for $a \in \Sigma$ and $w \in \Sigma^*$.

$$\begin{aligned} \delta^*(s, \varepsilon) &= \varepsilon \\ \delta^*(s, wa) &= \delta(\delta^*(s, w), a) \\ \delta^*(s, aw) &= \delta^*(\delta(s, a), w) \end{aligned}$$

Similary, we have for (2), given $A \subseteq Q$:

$$\begin{aligned} \Delta^*(A, \varepsilon) &= A \\ \Delta^*(A, wa) &= \bigcup_{q \in \Delta^*(A, w)} \Delta(q, a) \end{aligned}$$

Facts:

- $\delta^*(s, xy) = \delta^*(\delta^*(s, x), y)$
- $\Delta^*(A \cup B, w) = \Delta^*(A, w) \cup \Delta^*(B, w)$
- $\Delta^*(A, xy) = \Delta^*(\Delta^*(A, x), y)$

Definition 7. *Given a DFA,* $M = (S, s_0, \delta, F)$*,* $L(M) = \{w \in \Sigma^* | \delta^*(s_0, w) \in F\}$*.*

Definition 8. *Given an NFA,* $N = (Q, Q_0, \Delta, F)$*,* $L(N) = \{w \in \Sigma^* | \Delta(Q_0, w) \cap F \neq \varnothing\}$*.*

Theorem 3. *Given an NFA,* N*, there exists a DFA,* M*, s.t.* $L(M) = L(N)$*.*

Proof. Let, $M = (S, s_0, \delta, F)$, $N = (Q, Q_0, \Delta, F)$, $S = \mathcal{P}(Q)$, $s_0 = Q_0$, $\hat{F} = \{A \subseteq Q | A \cap F \neq \varnothing\}$, $\delta(A, a) = \bigcup_{q \in A} \Delta(q, a) = \delta^*(A, a)$. We require the following lemma:

Lemma 1. $\Delta(A, w) = \delta^*(A, w)\ \forall w \in \Sigma^*$.

The proof is as follows:

$$\begin{aligned} L(N) &= \{w | \Delta^*(Q_0, w) \cap F \neq \varnothing\} \\ &= \{w | \Delta^*(Q_0, w) \in \hat{F}\} \text{ (Defn of } \hat{F}) \\ &= \{w | \delta^*(Q_0, w) \in \hat{F}\} \\ &= \{w | \delta^*(s_0, w) \in \hat{F}\} = L(M) \end{aligned}$$

□

4 Closure Properties of Regular Languages

Theorem 4. *The following operations preserve regularity: if L_1, L_2 are regular languages, then*

1. *$L_1 \cup L_2$ is regular.*
2. *$L_1 \cap L_2$ is regular.*
3. *$L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$.*
4. *$\overline{L} = \{x \in \Sigma^* | x \notin L\}$.*
5. *$L^* = \{x_1 \dots x_n | \forall n \geq 0, x_i \in L\}$ (ε is always in L^* even if it is not in L).*

Proof. Let $M_1 = (S_1, s_0^{(1)}, \delta_1, F_1)$, $M_2 = (S_2, s_0^{(2)}, \delta_2, F_2)$, $L(M_1) = L_1$, $L(M_2) = L_2$.

1. We construct an NFA, N:
   - $Q = S_1 \cup S_2$ $(S_1 \cap S_2 \neq \varnothing)$
   - $Q_0 = \{s_0^{(1)}, s_1^{(2)}\}$
   - $F = F_1 \cup F_2$
   - $\Delta(s, a) = \begin{cases} \{\delta_1(s, a)\} & \text{if } s \in S_1 \\ \{\delta_2(s, a)\} & \text{if } s \in S_2 \end{cases}$
2. In this case we construct a DFA
   - $S = S_1 \times S_2$
   - $s_0 = (s_0^{(1)}, s_1^{(2)}) \in S_1 \times S_2$
   - $\delta((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, a))$
   - $F = F_1 \times F_2$
3. Construct NFA, $N = (Q, Q_0, \Delta, F)$:
   - $Q = S_1 \cup S_2$ $(S_1 \cap S_2 = \varnothing)$
   - $Q_0 = \{s_0^{(1)}\}$
   - $\Delta(s, a) = \begin{cases} \{\delta_1(s, a)\} & \text{if } s \in S_1 \\ \{\delta_2(s, a)\} & \text{if } s \in S_2 \end{cases}$
   - $\Delta(t, \varepsilon) = \{s_0^{(2)}\}$ $t \in F_1$
   - $F = F_2$
4. Exercise
5. (Watch this part of the lecture) Let $M = (S, s_0, \delta, F)$, $L(M) = L$. We construct the NFA, $N = (Q, Q_0, \Delta, F)$:
   - $Q = S \cup \{q_0\}$ where q_0 is a new state
   - $Q_0 = \{q_0\}$
   - $\hat{F} = F \cup \{q_0\}$
   - $\Delta(q, a) = \begin{cases} \{\delta_1(q, a)\} & \text{if } q \in S \ \& \ a \neq \varepsilon, q \notin F \\ \{\delta_1(q, a)\} & \text{if } q \in F \ \& \ a \neq \varepsilon \\ \{s_0\} & \text{if } q \in F \ \& \ a = \varepsilon \\ \{s_0\} & \text{if } q = q_0 \ \& \ a = \varepsilon \end{cases}$

   This machine accepts exactly the words in L^*. Argue that any word accepted must be in L^*.

□

See class examples!

4.1 Regular and Non-Regular Languages

The purpose of this section is to keep track of the languages seen in class that are regular or not (can be useful for counter-examples).

4.1.1 Regular Languages

4.1.2 Non-Regular Languages

5 Algebra of Regular Expressions

Definition 9. *Fix an alphabet Σ. The collection of regular expressions over Σ, ($Reg(\Sigma)$), is defined by induction as follows:*

1. *$\varnothing$ is a regular expression.*
2. *ε is a regular expression.*
3. *If $a \in \Sigma$, then a is a regular expression.*
4. *If R_1, R_2 are regular expressions, then so is $R_1 + R_2$.*
5. *If R_1, R_2 are regular expressions, the so is $R_1 R_2$.*
6. *If R is a regular expression, so is R^*.*

$[\![R]\!]$ is the set denoted by R.

- $[\![\varnothing]\!] = \varnothing$
- $[\![\varepsilon]\!] = \{\varepsilon\}$
- $[\![R_1 + R_2]\!] = [\![R_1]\!] \cup [\![R_2]\!]$
- $[\![R_1 R_2]\!] = [\![R_1]\!][\![R_2]\!] = \{xy | x \in [\![R_1]\!], y \in [\![R_2]\!]\}$
- $[\![R^*]\!] = \bigcup_{i=0}^{\infty} [\![R]\!] \ldots [\![R]\!]$

Theorem 5. *(Kleene's Theorem) A language is regular if and only if it is defined by a regular expression.*

Proof. Lol this thing is so long. Do later. □

5.1 Kleene Algebra

Regular expression form an algebra. It has two binary operations $+, \cdot$, one unary operation $*$, and 2 constants $\varnothing, \varepsilon$. The basic equations are as follows (more can be derived):

1. $R + \varnothing = \varnothing + R = R$
2. $R + S = S + R$
3. $R + (S + T) = (R + S) + T$
4. $R + R = R$
5. $R \cdot \varnothing = \varnothing \cdot R = \varnothing$
6. $R \cdot \varepsilon = \varepsilon \cdot R = R$
7. $R \cdot (S \cdot T) = (R \cdot S) \cdot T$
8. $R \cdot (S + T) = R \cdot S + R \cdot T$
9. $(R + S) \cdot T = R \cdot T + S \cdot T$
10. $\varepsilon + RR^* = \varepsilon + R^* R = R^*$

Other equations include: $(R^*)^* = R^*$, $(R^* S)^* R^* = (R + S)^*$.

6 Pumping Lemma

Definition 10. *(Informal) The pumping lemma is an essential property of regular languages. It says that all sufficiently long strings in a regular language may be pumped – that is, have a middle section of the string repeated an arbitrary number of times – to produce a new string that is also a part of the language.*

Definition 11. *Pigeon-Hole Principle: If you have n boxes and m objects where $m > n$, if you put all the objects into theses boxes, at least one box will have more than one object.*

Lemma 2. *The Pumping Lemma*
If L is a regular language, $\exists p \in \mathbb{N}\ p > 0$ s.t. $\forall w \in L\ |w| \geq p\ \exists x, y, z \in \Sigma^*$ s.t.:

1. $w = xyz$
2. $|xy| \leq p$
3. $|y| > 0$

$\forall i \in \mathbb{N}$, $xy^iz \in L$.

So, L is regular $\Rightarrow$ L can be pumped. The contrapositive is: L cannot be pumped $\Rightarrow$ L is not regular. The contrapositive is how the pumping lemma is used.
The above statements formally:

$$\begin{aligned} L \text{ regular} \Rightarrow &[\exists p > 0 \forall s \in L\ |s| \geq p \Rightarrow \exists x, y, z \in \Sigma^* \\ &\quad (s = xyz \wedge |xy| \leq p \wedge |y| > 0 \wedge \forall i \geq 0, xy^iz \in L)] \Rightarrow L \text{ can be pumped.} \end{aligned}$$

Note: Negating quantified statements.

- $\neg(\forall x\ \varphi(x)) \equiv \exists x\ \neg\varphi(x)$
- $\neg(\exists x\ \varphi(x)) \equiv \forall x\ \neg\varphi(x)$

$$\begin{aligned} &L \text{ cannot be pumped.} \Rightarrow \neg[\cdot] \\ &\Rightarrow [\forall p, p > 0 \Rightarrow \exists w \in L \wedge |w| \geq p \wedge \forall x, y, z \in \Sigma^* \\ &(w = xyz \wedge |xy| \leq p \wedge |y| > 0) \Rightarrow \exists i\ xy^iz \notin L] \Rightarrow L \text{ not regular.} \end{aligned}$$

Note: A universally quantified statement on an empty set is always true.

6.1 Examples

1. $L = \{a^nb^n | n \geq 0\}$
   Demon chooses p, I choose a^pb^p.

   Demon has to break up a^pb^p into 3 pieces: xyz.

   Assume $|xy| \leq p$, $|y| > 0$. The string looks like $aa \ldots abb \ldots b$. y must be in the string of a's, and $|y| \geq 1$. Choose $i = 2$, in which case we can obtain $xyyz$.

   I have inserted at least one extra a, and I have not changed the b's, so clearly the number of a's and the number of b's cannot be equal. Thus, $xyyz \notin L$. So, L cannot be regular.

2. $L = \{a^nb^ma^{n+m} | n, m > 0\}$

   Demon: p, Class: $a^pb^pa^{2p}$.

   Demon has to choose xy to consist purely of a's and y cannot be empty ($|y| = l > 0$). The class chooses $i = 2$. This gives $xy^2z = a^{p+l}b^pa^{2p}$, but $p + l + p \neq 2p$, so $xy^2z \notin L$.

3. $L = \{ab^n a^n | n > 0\}$

   Demon: p, Class $ab^p a^p$.

   The demon must consider some cases:

   (a) $x = \varepsilon$, $y = a$
   (b) $x = a$, $y = b^l$ where $p \geq l > 0$
   (c) $x = ab^m$, $y = b^l$ where $p \geq l > 0$

   The class must display that, after choosing $i = 2$, $xy^2z \notin L$.

4. Question: Supposed $L \subseteq \{a, b\}^*$, and suppose L is infinite. If every word w in L satisfies $\#_a(w) = \#_b(w)$, is it possible that L could be regular?

   A: YES!

   Consider $(ab)^*$. This is regular, is infinite, and every word has an equal number of a's and b's. Furthermore, it is easy to construct a DFA for this language.

5.

7 Minimization

Sometimes, DFA's may have more states than are required (seeing as we can actually define a lower-bound for the number states for any given DFA). Thus, the point of this section is to define minimization techniques for a DFA to simplify our solution.

Definition 12. *Given a DFA $M = (D, s_0, \delta, F)$ over alphabet Σ, we say $p, q \in S$ are equivalent (and write $p \approx q$) if $\forall x \in \Sigma^*$, $\delta^*(p, x) \in F \iff \delta^*(q, x) \in F$.*

Remark: $p \not\approx q$ means $\exists x \in \Sigma^*$, $\delta^*(p, x) \in F$ and $\delta^*(q, x) \notin F$ or $\delta^*(p, x) \notin F$ and $\delta^*(q, x) \in F$. (Note that $\approx$ is an equivalence relation.)

Lemma 3. *$p \approx q \Rightarrow \forall a \in \Sigma$, $\delta(p, a) \approx \delta(q, a)$.*

Proof. Assume $p \approx q$. Suppose $\delta^*(\delta(p, a), x) \in F$, i.e. $\delta^*(p, ax) \in F$. Since we assumed $p \approx q$ then $\delta^*(q, ax) \in F$ and $\delta^*(\delta(q, a), x) \in F$. Similarly, in the reverse direction. Thus, $\delta(p, a) \approx \delta(q, a)$. □

Note on notation: $p \approx q$ means $[p] = [q]$, so the lemma can be restated as: $[p] = [q] \Rightarrow \forall a \in \Sigma$, $[\delta(p, a)] = [\delta(q, a)]$.
So the shrunken machine is defined as follows:

- S' = equivalence classes: $S/\approx$ (the collection of equivalence classes).
- $S'_0 = [s_0]$.
- $\delta'([p], a) = [\delta(p, a)]$ (which is well-defined because of the lemma).
- $F' = \{[s] | s \in F\}$.

Lemma 4. *$p \in F$ and $p \approx q \Rightarrow q \in F$. (Gotta think about this one)*

Lemma 5. *$\forall w \in \Sigma^*$, $\delta'^*([p], w) = [\delta^*(p, w)]$.*

Proof. Induction on w.

- Base case: $w = \varepsilon$.

$$\delta'^*([p], \varepsilon) = [p] = [\delta^*(p, \varepsilon)]$$

- Inductive step: Assume $\delta'^*([p], w) = [\delta^*(p, w)] \leftarrow$ WTS $\forall a \in \Sigma$, $\delta'^*([p], wa) = [\delta^*(p, wa)]$.

$$\begin{aligned}
\delta'^*([p], wa) &= \delta'(\delta'^*([p], w), a) \\
&= \delta'([\delta^*(p, w)], a) \text{ Inductive Hypothesis} \\
&= [\delta(\delta^*(p, w), a)] \text{ Def. of } \delta' \\
&= [\delta^*(p, wa)] \text{ Def. of } \delta^*
\end{aligned}$$

□

Theorem 6. *$L(M') = L(M)$.*

Proof.

$$\begin{aligned}
x \in L(M') &\iff \delta'^*([s_0], x) \in F' \\
&\iff [\delta^*(s_0, x)] \in F' \\
&\iff \delta^*(s_0, x) \in F \\
&\iff x \in L(M)
\end{aligned}$$

□

7.1 Algorithm based on this $\approx$

Idea: Start by putting all the states into 2 groups: accept, reject. Keep splitting the groups.

Notation: $p \bowtie q$ if $p \not\approx q$. i.e. $\exists w \in \Sigma^*$ s.t. $\delta^*(p, w) \in F$ and $\delta^*(q, w) \notin F$ or the other way around.

Fact: If $\exists a \in \Sigma$ s.t. $\delta(p, a) \bowtie \delta(q, a)$, then $p \bowtie q$.

Algorithm: Define an $S \times S$ array of booleans:

1. For every pair (p, q) s.t. $p \in F$ and $q \notin F$, we put a 0 in the (p, q) cell.
2. Repeat until no more changes:

   For each pair (p, q) not marked 0, check if $\exists a \in \Sigma$ s.t. $(\delta(p, a), \delta(q, a))$ is marked 0. If yes put a 0 in (p, q).
3. Mark everything remaining with a 1.

Theorem 7. *If two states are not labelled 0 by the algorithm, they are equivalent.*

Proof. Suppose the machine is (S, s_0, δ, F). Assume the theorem is false. There is a pair of states that are not equivalent but which the algorithm fails to mark. Call such a pair a BAD pair. Suppose 2 states are not equivalent, then there is a string that distinguishes them, i.e. $\exists x \in \Sigma^*$ s.t. $\delta^*(p, x) \in F$ and $\delta^*(q, x) \notin F$. Among all such bad pairs, choose the one with the shortest distinguishing string. Note, ε cannot be such a distinguishing string.

Lets assume (s, t) is a bad pair, and its distinguishing string $x = x_1 x_2 x_3 \ldots x_n$ is the shortest one possible. Now consider $\delta(s, x_1)$ and $\delta(s, x_2)$. This pair is not equivalent since:

$$\delta^*(\delta(s, x_1, x_1 \ldots x_n) \in F, \;\; \delta^*(\delta(t, x_1, x_1 \ldots x_n) \notin F$$

so $\delta(s, x_1) \bowtie \delta(t, x_1)$. But it cannot be a bad pair because its distinguishing string is strictly smaller than x. So the algorithm must have marked $(\delta(s, x_1), \delta(t, x_1))$ with a 0 at some stage. But in the next step, it has to mark (s, t) which is a contradiction. Thus, there are no bad points.

□

Running Times:

- Basic algorithm: $O(n^4)$.
- Improved algorithm maintains a list of pairs that will be immediately distinguished if s, t are distinguished: $O(n^2 k)$ (n^2 comes from the states and k is the size of the alphabet).
- Clever algorithm (John Hopcroft): $O(n \log n)$ which is the best possible.
- Old algorithm: exponential time.

8 Myhille-Nerode Theorem

Definition 13. *An equivalence relation, R, on Σ^* is said to be right-invariant if $\forall x, y \in \Sigma^*$, $xRy \Rightarrow \forall z \in \Sigma^*$, $xzRyz$.*

- Major example: Given a DFA $M = (Q, \Sigma, q_0, \delta, F)$, where $\delta^* : Q \times \Sigma^* \to Q$: $\forall q \in Q$, $\delta^*(q, \varepsilon) = q$. $a \in \Sigma$, $x \in \Sigma^*$, $\delta^*(q, ax) = \delta^*(\delta(q, a), x)$, it follows $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$. (Watch this part of the lecture)

Definition 14. *We define R_M based on M. $xR_M y$ if and only if $\delta^*(q_0, x) = \delta^*(q_0, y)$.*

This is clearly an equivalence relation, and it is right invariant (prove if your bored).

Definition 15. *Given any language $L \subseteq \Sigma^*$ (not necessarily regular), we define an equivalence relation as $\equiv_L$:*

$$x \equiv_L y \text{ iff } \forall z,\ xz \in L \iff yz \in L$$

Lemma 6. *$\equiv_L$ is right-invariant.*

Theorem 8. *(Myhill-Nerode) The following are equivalent:*

1. *L is accepted by a DFA i.e. L is a regular language.*
2. *L is the union of some of the equivalence classes of some right-invariant equivalence relation of finite index.*
3. *The equivalence relation $\equiv_L$ has a finite index.*

Proof. We will show (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (1).

- (1) $\Rightarrow$ (2): We assume L is recognized by a DFA, $M = (Q, \Sigma, q_0, \delta, F)$. We've already seen that R_M defined above is right-invariant. Thus, R_m has, at most, $|Q|$ equivalence classes, thus R_m definitely has a finite index. The equivalence classes are, for $q \in Q$:

$$S_q := \{x | \delta^*(q_0, x) = q\}$$
$$L = \bigcup_{q \in F} S_q$$

- (2) $\Rightarrow$ (3): (3) says $\equiv_L$ has finite index. We must show that any relation of the form in (2) must have an index bigger than $\equiv_L$.

  R refines $\equiv_L$ when the equivalence classes of R sit inside the equivalence classes of $\equiv_L$. Thus, there are clearly more classes of R than there are of $\equiv_L$.

  Let R be any right-invariant equivalence relation of finite index such that L is the union of some of the equivalence classes of R. We must show that $xRy \Rightarrow x \equiv_L y$.

  If $x \in L$, then any string y s.t. xRy must also be in L. Since R is right invariant, $\forall z$, $xzRyz$. If $xz \in L$, then $yz \in L$ and also vice versa. This is exactly the defintion of $\equiv_L$, thus $xRy \Rightarrow x \equiv_L y$.

- (3) $\Rightarrow$ (1): We construct a DFA from L. $(Q', \Sigma, q_0', \delta', F')$:

- Q': set of equivalence classes of $\equiv_L$. We already know that $\equiv_:$ is right invariant. (2) says that $\equiv_L$ has finite index, so Q' is finite.
- $q_0' = [\varepsilon]$.
- $\delta'([x], a) = [xa]$. Claim: this is well-defined. If x' from $[x]$ was chosen instead of x, then we would have gotten $[x'a]$ instead, but $x \equiv_L x'$, hence $xa \equiv_L x'a$, so $[xa] = [x'a]$.
- $F' = \{[x] | x \in L\}$.

Now, we have a DFA.

$$\delta'^*(q_0', x) = \delta'^*([\varepsilon], x) = [x]$$

x is accepted iff $x \in L$, thus (3) $\Rightarrow$ (1).

Note: this machine is the unique minimal DFA for L.

$\square$

8.1 Algorithms for Regular Languages

We want high-level descriptions using known algorithms as building blocks.

- Minimization.
- Determinization: NFA $\to$ DFA.
- Graph algorithms to test for reachability or cycles.
- From a DFA, we can construct a regular expression.
- From a regular expression, we can construct an NFA.

Examples:

- **Ex**: Given an NFA, design an algorithm to decide if it accepts anything.

  Ans: Run a reachability algorithm to check if any accept state can be reached from any start state.
- **Ex**: Given an NFA, does it reject anything?

  Ans: Turn the NFA into a DFA, then check if any non accept stat is reachable from the start state.

Hint: Never try to use *regexp* in an algorithm! For example, see if $regexp = \Sigma^*$. A bad way to check if $L(M_1) = L(M_2)$ is check every word in Σ^* on each machine.

9 Automata Learning

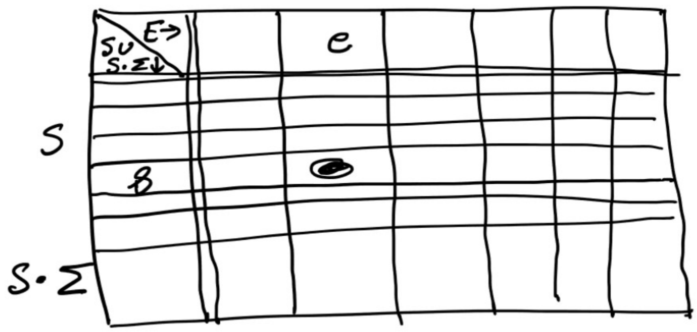
We are given a language and we design a DFA. We are given a DFA (NFA, NFA + ε) and asked to analyze it. Can we learn a DFA just from examples? Dana Angluin proved that you can learn a DFA from a teacher.
The teacher answers 2 kinds of questions:

1. Is this word, w,in L?
2. Is this the right machine (DFA)? $\begin{cases} \text{Yes} & \text{Done} \\ \text{No} & \text{Provide counter example} \end{cases}$

In polynomial time (in the number of states of the minimal DFA), one can always learn the DFA. In fact, one can construct the minimal DFA.

9.1 Minimally Adequate Teacher

Key data structure: Observation Table. Begin with 2 finite sets: $S, E \subseteq \Sigma^*$



- S: best guess of the states.
- $\delta \cdot \Sigma$: All the "next" states.
- E: what experiments should I do to tell the states apart.

  The entry indexed by s and e is 1 if $se \in L$ and 0 if $se \notin L$, where L is the language we are trying to learn.

$$T(s,e) = \begin{cases} 1 & \text{if } se \in L \\ 0 & \text{if } se \notin L \end{cases}$$

Initially $S = E = \{\varepsilon\}$.

From an observation table, we define a DFA:

- $q_0 = row(\varepsilon)$.
- $Q = \{row(s) | s \in S\}$.
- $F = \{row(s) | s \in S \ \& \ T(s\varepsilon) = T(s) = 1\}$.
- $\delta(row(s), x) = row(sx)$ where $x \in \Sigma$.

An observation table is *closed* if $\forall t \in S \cdot \Sigma \ \exists s \in S$ s.t. $row(t) = row(s)$. The states to which I go are in the machine.

An observation table is *consistent* if, whenever $s_1, s_2 \in S$,

$$row(s_1) = row(s_2) \Rightarrow \forall x \in \Sigma, \ row(s_1 x) = row(s_2 x)$$

0's and 1's are arranged the same.

This algorithm is called L^* (SEE CLASS EXAMPLES FOR CLOSED AND CONSISTENT TABLE).