CSC240

Lecture 13 week 9

Author: Joseph Siu

Email: joseph.siu@mail.utoronto.ca

Date: March 21, 2024

M'(m,n)

```
$w \leftarrow m$
$y \leftarrow n$
$z \leftarrow 0$
while $w \neq 0$ do
    if $w \mod 2 = 1$ then
        $z \leftarrow z + y$
    $w \leftarrow w$ div 2                    ▷ quotient when $w$ is divided by 2
    $y \leftarrow 2 \times y$                  ▷ or $y \leftarrow y + y$
return z
```

Precondition: $m \in \mathbb{N}, n \in \mathbb{Z}$

Postcondition: $z = n \times m$, m and n are unchanged.

Find Loop Invariants

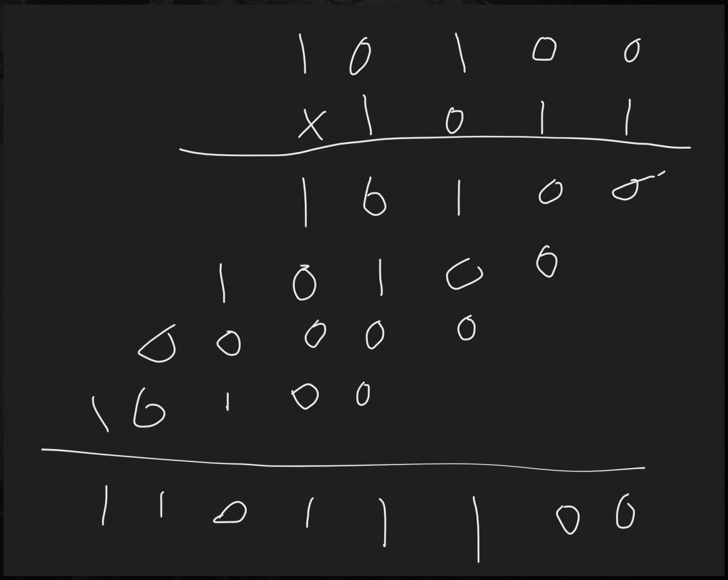
1. Look at examples

Example 1. $m = 11 = 1011_2$, $n = 20 = 10100_2$.
Let w_i, y_i and z_i denote the values of w, y, and z immediately after the i^{th} iteration of the while loop.

i	w_i	w_i in binary	y_i	z_i
0	11	101**1**	20	0
1	5	10**1**	40	20
2	2	1**0**	80	60
3	1	**1**	160	60
4	0	**0**	320	220
		$w_1 \mod 2$		

Here $w_i = w$ shifted to the right i positions,
$y_i = n2^i = n$ shifted to the left i positions,
z_i=?

2. Use your understnading of how the algorithm works



Let $m_{k-1} \cdots m_0$ denote the binary representation of m:

$$m = \sum_{j=0}^{k-1} m_j 2^j,$$

where $m_j \in \{0,1\}$ for $0 \le j \le k-1$.

For example $m = 1011_2$, then $m_0 = 1, m_1 = 1, m_2 = 0, m_3 = 1$.

$$w_i = \sum_{j=i}^{k-1} m_j 2^{j-i} = m_{k-1} \cdots m_i$$

$$w_i \mod 2 = m_i$$

$$2^i w_i = \sum_{j=1}^{k-1} m_j 2^j$$

$$2^{i+1} w_{i+1} = \sum_{j=i+1}^{k-1} m_j 2^j = m - \sum_{j=0}^{i} m_j 2^j$$

So,

$$\sum_{j=0}^{i} m_j 2^j = m - w_{i+1} 2^{i+1}$$

$$\begin{aligned}
z_0 &= 0\\
z_{i+1} &= \begin{cases} z_i + y & \text{if } w_i \mod 2 = 1\\ z_i & \text{if } w_i \mod 2 = 0 \end{cases}\\
&= z_i + m_i y^i\\
&= z_i + m_i n 2^i\\
&= m_0 n 2^0 + m_1 n 2^1 + \cdots + m_i n 2^i\\
&= n \sum_{j=0}^{i} m_j 2^j\\
&= n\left(m - w_{i+1} 2^{i+1}\right)\\
&= nm - w_{i+1} y_{i+1}
\end{aligned}$$

Lemma 1

If $m \in \mathbb{N}$ and $n \in \mathbb{Z}$ then $z = n \times m - w \times y$ is a loop invariant.

Proof.
Initially $z = 0$, $w = m \in \mathbb{N}$, $y = n$ by lines 1,2,3.
So $n \times m - w \times y = n \times m - m \times n = 0 = z$.
Consider an arbitrary iteraiton of the loop.
Let w', y', z' denote the values of w, y and z at the beginning of the iteration and let w'', y'', z'' denote the values at the end.
Suppose the claim is true at the beginning of the iteration $z' = n \times m - w' \times y'$, and $w' \in \mathbb{N}$.
From line 7, $y'' = 2y'$.
Case 1. w' is odd
From lines 5 and 6, $z'' = z' + y'$ and $w'' = \frac{(w'-1)}{2} \in \mathbb{N}$
So

$$\begin{aligned}
n \times m - w'' \times y'' &= n \times m - \frac{(w'-1)}{2} \times 2y'\\
&= n \times m - (w'-1) \times y'\\
&= m \times n - w' \times y' + y\\
&= z' + y' = z''.
\end{aligned}$$

Case 2. w' is even
From lines 5 and 6, $z'' = z'$ and $w'' = \frac{w'}{2}$.
So,

$$\begin{aligned}
n \times m - w'' \times y'' &= n \times m - \frac{w'}{2} \times 2y'\\
&= n \times m - w' \times y'\\
&= z' = z''.
\end{aligned}$$

Quod
Erat
Dem■

Corollary 1 – Partial Correctness

Let $m \in \mathbb{N}$ and $n \in \mathbb{Z}$.
If $\mathrm{M}'(m, n)$ is run and it halts then when it halts $z = n \times m$ and n and m are unchanged.

Proof. The code does not do any assignments to m or n so they are unchanged.
From the termination condition of the loop, $w = 0$. y the Lemma $z = n \times m - w \times y = n \times m$

Quod Erat Dem■

Why does M'(m,n) terminate?

w gets smaller every complete iteration of the while loop.

Lemma 2 – Termination

If $w \in \mathbb{N}$ and $n \in \mathbb{Z}$ and $\mathrm{M}'(m, n)$ is run, it eventually halts.

Proof.
Suppose the loop does not terminate.
Consider the sequence $\{w_i\}_{i \geq 0}$.
Since $w \in \mathbb{N}$ is a loop invariant, $w_i \in \mathbb{N}$ for all $i \geq 0$.
$w_{i+1} < w_i$ since $w_i \in \mathbb{Z}^+$ and $w_{i+1} = w_i$ div 2.
This contradicts the well ordering principle.
Then the loop eventually terminates.

Quod Erat Dem■

1 Language

Language is just a subset of the words in the alphabet, for example in english, we have works from a to z, and english is a part of the language.

Formally, let Σ denote a finite alphabet, Σ^*= set of all (finite length) strings over Σ.

A language over Σ is a subset of Σ^*.

$\lambda \in \Sigma^*$ denotes the empty string.

If $x, y \in \Sigma^*$, we use $x \cdot y$ to denote the concatenation of x and y, this makes sense since x and y are finite, otherwise it won't make sense at all.

In other words, if $x, y \in \Sigma^*$, then $x \cdot y$ is the string consisting of the letters of x followed by the letters of y.

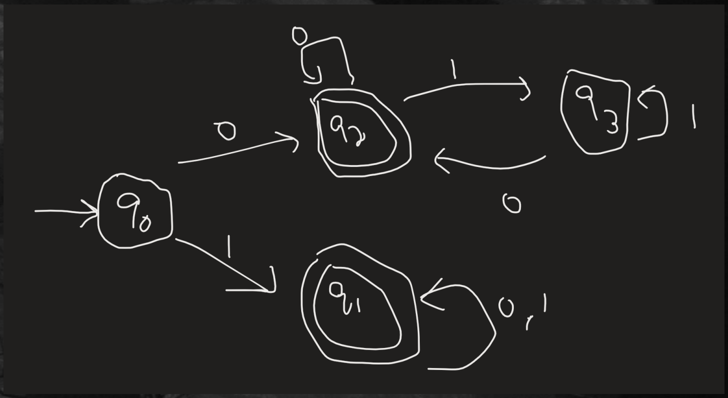
For example, if $x = aab, \quad y = ba$ concatenation,

then

$$\begin{aligned} xy &= aabba \\ yx &= baaab \\ x \cdot \lambda = x &= \lambda \cdot x \end{aligned}$$

1.1 DFA

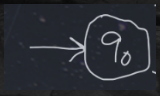
A (deterministic) finite (state) automation DFA or DFSA is a way of describing a language



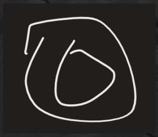
Deterministic finite state automation A.

Finite set of states $Q = \{q_0, q_1, q_2, q_3\}$.

q_0 initial state



$F = \{q_1, q_2\}$ set of final states



$\Sigma = \{0, 1\}$

$\delta : Q \times \Sigma \to Q$ is the state transition function

$\delta(q_0, 0) = q_2, \delta(q_0, 1) = q_1, \delta(q_1, 0) = q_1, \delta(q_1, 1) = q_1;$

$\delta(q_2, 0) = q_2, \delta(q_2, 1) = q_3, \delta(q_3, 0) = q_2, \delta(q_3, 1) = q_3.$

Given an input string $a \in \{0, 1\}^*$, the DFA behaves as follows

- It starts in the initial state, q_0
- It reads the string a from left to right, 1 letter at a time and changes state according to the transition function following the edge labelled by the letter out of the current state, when all of the letters have been read, the DFA accepts if it is in a final state (in F) and rejects if it is in $Q - F$.

For example,

- 0110 accept
- 100 accept
- 0101 rejects

D is a 5-tuple. That is, $D = (Q, \Sigma, \delta, q_0, F)$.

If D is a DFA, then the language accepted by D is

$$\begin{aligned} L(D) &= \{x \in \Sigma^* \mid \text{D accepts } x\} \\ &= \{x \in \Sigma^* \mid \delta^*(q_0, x) \in F\} \end{aligned}$$ [1]

For example,

$$L(A) = \{x \mid x \text{ begins with 1 or ends with 0}\}$$

1.2 Extended Transition Function

$\delta^* : Q \times \Sigma^* \to Q$

$\delta^*(q, \lambda) = q$ for all $q \in Q$

For all $q \in Q, a \in \Sigma, x \in \Sigma^*$, define

$$\delta^*(q, x \cdot a) = \delta(\delta^*(q, x), a).$$

To prove $L(A) = \{x \mid x \text{ begins with 1 or ends with 0}\}$

- associate a language $L_i \in \Sigma^*$ with each state q such that $L_i = \{x \in \Sigma^* \mid \delta^*(q_0, x) = q_i\}$
- prove by structural induction or inductions, on the length of λ

$L_0 = \{\lambda\}$

$L_1 = \{x \in \{0,1\}^* \mid x \text{ starts with 1}\}$

$L_2 = \{x \in \{0,1\}^* \mid x \text{ starts with 0 and ends with 0 }\}$

$L_3 = \{x \in \{0,1\}^* \mid x \text{ starts with 0 and ends with 1}\}$

For all $x \in \Sigma^*$, let $P(x) =$ "$\forall i \in \{0,1,2,3\}. [(x \in L_i)$ IFF $\delta^*(q_0, x) = q_i]$"

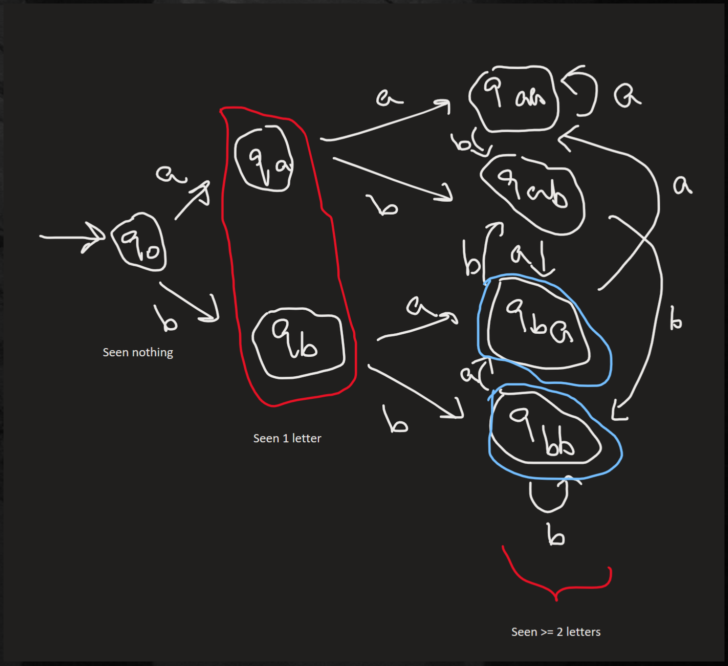
Prove by structural induction on recursive defined $\{0,1\}^*$:

$\lambda \in \{0,1\}^*$

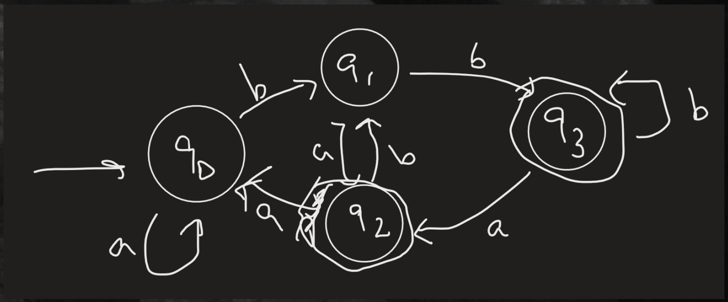
If $x \in \{0,1\}^*$ and $a \in \{0,1\}$, then $x \cdot a \in \{0,1\}^*$

Let $P'(n) = \forall x \in \{0,1\}^n . P(x)$ [2]

Given a DFA that accepts $\{x \in \{a,b\}^* \mid \text{ the second last letter of } x \text{ is } b\}$



[2] all binary string of length n

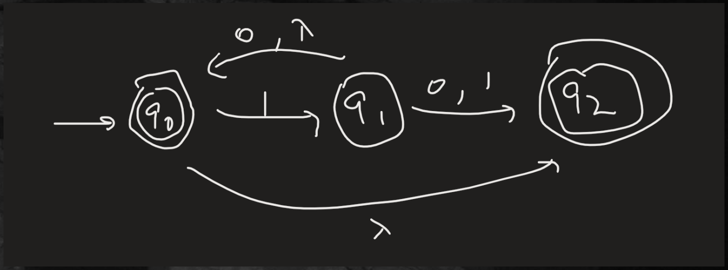


2 Nondeterministic Finite State Automaton

NFA FNSA, $N = (Q, \Sigma, \delta, q_0 \in Q, F \subseteq Q)$ is also a 5-tuple. The only difference is how δ is defined compared to DFA.

$\delta : Q \times (\Sigma \cup \{\lambda\}) \to P(Q)$, range of δ is $P(Q) = \{s \mid S \subseteq Q\}$ in staed of Q.

$\delta(q, \lambda)$ - transition without reading a letter



$\delta(q_2, 0) = \varnothing = \delta(q_3, 1) = \delta(q_2, \lambda)$

$\delta(q_1, 0) = \{q_0, q_2\}$

$\delta(q_1, 1) = \{q_2\}$

Let $E : Q \to P(Q)$ denote the set of states reachale from q by following (including 0 and at most $|\mathbb{Q}| - 1$ steps) any number of λ-transitions (that is, $\delta(q, \lambda)$ is a λ-transition)

$E(q_0) = \{q_0, q_2\}$

$E(q_1) = \{q_0, q_1, q_2\}$

$E(q_2) = \{q_2\}$

2.1 Extended Transition Function

The extended transition function $\delta^* : Q \times \Sigma^* \to P(Q)$ can be defined recursively as follows:

Base Case: $\delta^*(q, \lambda) = E(Q)$ for all $q \in Q$.

Constructor Case: For all $q \in Q, a \in \Sigma, x \in \Sigma^*$, let

$$\delta^*(q, xa) = \bigcup \{E(q'') \mid q'' \in \delta(q', a)\} \text{ for some } q' \in \delta^*(q, x).$$

Then, one can prove by induction that $\forall x \in \Sigma^* . \forall y \in \S^* . [\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)]$.

For example,

$$\begin{aligned}
\delta^*(q_0, \lambda) &= E(q_0) = \{q_0, q_2\} \\
\delta^*(q_0, 1) &= E(\delta(q_0, 1)) \cup E(\delta(q_2, 1)) \\
&= E(\{q_1\}) \cup E(\varnothing) \\
&= \{q_0, q_1, q_2\}
\end{aligned}$$

Above we defined $\delta^*(q, x) =$ “ set of all states that can be reached from q when the letters of x are read.”

Now, let $L(N) = \{x \in \Sigma^* \mid \delta^*(q_0, x) \cap F \neq \varnothing\}$

N accepts x if and only if there is a sequence of lucky guesses that the machine can bring from the initial state to the final state.

In other words N accepts x if there is a sequence of generated N can make to go from q_0 to a final state while reading x.