CSC236 Week 01: Introduction and Basic Induction

Hisbaan Noorani

September 9 – 15, 2021

Contents

1 Why reason about computing

- You're not just hackers anymore

  Sometimes you need to analyze code before it runs. Sometimes it should never be run!
- Can you test everything?

  Infinitely many inputs: integers, strings, lists.
- Careful, you might get to like it... (?!*)

2 How to reason about computing

- It's messy...

  interesting problems fight back.

  You need to draft, re-draft, and re-re-draft.

You need to follow blind alleys until you find a solution.

You can also find a solution that isn't wrong, but could be better.

- It's art...

  Strive for correctness, clarity, surprise, humour, pathos, and others.

3 How to do well in this course

- Read the syllabus as a two-way promise
- Question, answer, record, synthesize. Try annotating blank slides.
- Collaborate with respect. You need computer science friends who are respectful and constructively critical.

4 Assume that you already know

- Chapter 0 material from *Introduction to Theory of Computation*.
- CSC110/111 material, especially proofs and big-$\mathcal{O}$.

5 By December you'll know

- Understand and use several flavours of induction. Some of these flavours will taste new.
- Formal languages, regular languages, regular expressions.
- Complexity and correctness of programs — both recursive and iterative.

6 Domino fates foretold

$$[P(0) \land (\forall n \in \mathbb{N}, P(n) \implies P(n+1))] \implies \forall n \in \mathbb{N}, P(n)$$

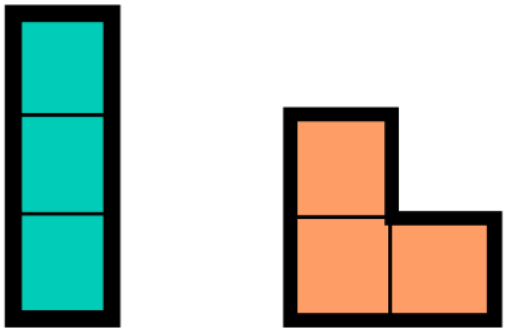
If the initial case works, and each case that works implies its successor works, then all cases work

7 Simple induction outline

- Inductive step: introduce n and inductive hypothesis $H(n)$
  - Derive conclusion $C(n)$: show that $C(n)$ follows from $H(n)$, indicating **where** you use $H(n)$ and why that is valid.
- Verify base case(s): verify that the claim is true for any cases not covered in the inductive step
- In simple induction $C(n)$ is just $H(n+1)$

8 Trominoes

See `https://en.wikipedia.org/wiki/Tromino`



Can an $n \times n$ square grid, with one subsquare removed, be tiled (covered without overlapping) by "chair" trominoes?

- 1×1: Yes.
- 2×2: Yes.
- 3×3: No. The remaining number of squares is not divisible by 3.
- 4×4: Yes.

Proof: $\forall n \in \mathbb{N}$, define the predicate $P(n)$ as a $2^n \times 2^n$ square grid, with one subsquare removed, can be tiled (covered without overlapping) by "chair" trominoes.

- Induction on n

  Let n be an arbitrary, fixed, natural number. (Let $n \in \mathbb{N}$).

  Assume $P(n)$, that is a $2^n \times 2^n$ grid, with one square removed can be tiled with "chairs."

  I will prove $P(n+1)$, that is a $2^{n+1} \times 2^{n+1}$ grid, with one square removed can be tiled by chairs.

  Let G be a $2^{n+1} \times 2^{n+1}$ grid with one square removed. Notice that G can be decomposed into four $2^n \times 2^n$ disjoint quadrant grids. We may assume, without loss of generality, that the missing square is in the upper-right quadrant, since otherwise just rotate it there, and rotate back when done. By $P(n)$ I can tile the upper-right quadrant, minus the missing square. By $P(n)$ 3 more times, I can tile the remaining 3 quadrants, omitting for a moment the 3 tiles nearest the centre of G, with chairs. The briefly omitted squares form a chair! So I complete the tiling by adding one more chair. Thus $P(n+1)$.

- Base Case

  A $2^0 \times 2^0$ grid, with one square removed, is just empty space! This can be tiled with 0 chairs. So $P(0)$ is true.

And thus $\forall n \in \mathbb{N}, P(n)$. ■

9 $3^n \geq n^3$?

9.1 Scratch Work

Check for a few values of n:

$$\begin{aligned} 3^0 = 1 \geq 0 = 0^3 & \quad \checkmark \\ 3^1 = 3 \geq 1 = 1^3 & \quad \checkmark \\ 3^2 = 9 \geq 8 = 2^3 & \quad \checkmark \\ 3^3 = 27 \geq 27 = 3^3 & \quad \checkmark \\ 3^4 = 81 \geq 64 = 4^3 & \quad \checkmark \\ 3^{-1} = \frac{1}{3} \geq -1 = -1^3 & \quad \checkmark \\ 3^{2.5} < 2.5^3 & \quad \times \end{aligned}$$

9.2 Simple Induction

Proof: $\forall n \in \mathbb{N}$, define the predicate $P(n)$ as $3^n \geq n^3$.

- Induction on n

  Let $n \in \mathbb{N}$. Assume $H(n) \ : \ 3^n \geq n^3$. I will prove $H(n+1)$ follows, that is $3^{n+1} \geq (n+1)^3$.

$$\begin{aligned} 3^{n+1} &= \ 3 \cdot 3^n \\ &\geq 3 \cdot n^3 \\ &= n^3 + n^3 + n^3 \\ &\geq n^3 + 3n^2 + 9n && \text{(since } n \geq 3\text{)} \\ &\geq n^3 + 3n^2 + 3n + 6n \\ &= n^3 + 3n^2 + 3n + 1 && \text{(since } 6n \geq 1\text{)} \\ &= (n+1)^3 \end{aligned}$$

  And thus we have shown that $\forall n \in \mathbb{N}$ s.t. $n \geq 3, H(n) \implies H(n+1)$.

- Base Case

  $3^3 \geq 3^3$ so $P(3)$ holds.
  $3^2 \geq 2^3$ so $P(2)$ holds.
  $3^1 \geq 1^3$ so $P(1)$ holds.
  $3^0 \geq 0^3$ so $P(0)$ holds.

And thus, we have shown $\forall n \in \mathbb{N}, 3^n \geq n^3$, as needed. ■

CSC236 Week 02: Complete Induction

Hisbaan Noorani

September 16 – 22, 2021

Contents

1 Complete Induction

- Every natural number greater than 1 has a prime factorization

$$
\begin{aligned}
2 &= 2 \\
3 &= 3 \\
4 &= 2 \times 2 \\
5 &= 5 \\
6 &= 2 \times 3 \\
7 &= 7 \\
8 &= 2 \times 2 \times 2 \\
9 &= 3 \times 3 \\
10 &= 2 \times 5
\end{aligned}
$$

- How does the factorization of 8 help with the factorization of 9?

  The fact that 8 can be expressed as a product of primes has nothing to do with 9 being a product of primes.

2 Notational Convenience

Sometimes you will see the following:

$$\bigwedge_{k=0}^{k=n-1} P(k)$$

... as equivalent to

$$\forall k \in \mathbb{N}, k < n \implies P(k)$$

3 More dominoes

$$\left(\forall n \in \mathbb{N}, \left[\bigwedge_{k=0}^{k=n-1} P(k)\right] \implies P(n)\right) \implies \forall n \in \mathbb{N}, P(n)$$

If all the previous cases always imply the current case then all cases are true.

4 Complete induction outline

- **Inductive step**: introduce n and state inductive hypothesis $H(n)$
  - **Derive conclusion** $C(n)$: show that $C(n)$ follows from $H(n)$, indicating where you use $H(n)$ and why that is valid.
- **Verify base case(s)**: verify that the claim is true for any cases not covered in the inductive step

This is the same outline as simple induction but we modify the inductive hypothesis, $H(n)$ so that it assumes the main claim for every natural number from the starting point up to $n-1$, and the conclusion, $C(n)$ is now the main claim for n.

5 Watch the base cases, part 1

$$f(n) = \begin{cases} 1 & n \leq 1 \\ [f(\lfloor\sqrt{n}\rfloor)]^2 + 2f(\lfloor\sqrt{n}\rfloor) & n > 1 \end{cases}$$

Check a few cases, and make a conjecture:

$$\begin{aligned} f(0) &= 1 \\ f(1) &= 1 \\ f(2) &= 3 \\ f(3) &= 3 \\ f(4) &= 15 \\ f(5 \ldots 15) &= 15 \\ f(16) &= 255 \end{aligned}$$

All of these things are divisible by 3. The square of something that is divisible by 3 is still divisible by 3 and the double of something that is divisible by 3 is still divisible by 3.

5.1 For all natural numbers $n > 1, f(n)$ is a multiple of 3?

Proof: $\forall n \in \mathbb{N}$, define the predicate $P(n)$ as $f(n)$ is a multiple of 3.

I will prove, using complete induction, that $\forall n > 1, P(n)$.

- Induction on n:

  Let $n \in N$. Assume $n > 1$. Also assume that $P(k)$ is true for all natural numbers k less than n, and greater than 1.

  Notice that the floor of the square root of n is greater than 1. Also, the square root of n is less than n (since $n > 1 \implies n^2 > n \implies n > \sqrt{n}$).

  Thus by the induction hypothesis, I have $P(\left\lfloor \sqrt{(n)} \right\rfloor)$, this number is a multiple of 3.

  Let $k \in \mathbb{N}$ s.t. $\lfloor \sqrt{n} \rfloor = 3k$, so $f(n) = (3k)^2 + 2(3k) = 3\left(3k^2 + 2k\right)$, a multiple of 3.

  So $P(n)$ follows in both possible cases.

- Base Cases:

  $P(2)$ claims that $f(2) = 3$ is a multiple of 3, which is true.

  $P(3)$ claims that $f(3) = 3$ is a multiple of 3, which is true. We have to prove $P(3)$ in the base case because the floor of the square root of three is not 2, but 1.

And thus $\forall n \in \mathbb{N}$ s.t. $n \geq 2, P(n)$. ■

Note: We have to include $P(3)$ as a base case because $\lfloor \sqrt{3} \rfloor$ is not 2, but 1.

6 Zero pair-free binary strings

Deonte by $zpfbs(n)$ the number of binary strings of length n That contain no pairs of adjacent zeros. What is $zpfbs(n)$ for the first few natural numbers n?

$$
\begin{aligned}
zpfbs(0) &= 1 \\
zpfbs(1) &= 2 \\
zpfbs(2) &= 3 \\
zpfbs(3) &= 5 \\
zpfbs(4) &= 8 \\
zpfbs(5) &= 13 \\
&\cdots \\
zpfbs(n) &= zpfbs(n-1) + zpfbs(n-2)
\end{aligned}
$$

$$
f(n) = \begin{cases} 1, & n = 0 \\ 2, & n = 1 \\ f(n-1) + f(n+2), & n > 1 \end{cases}
$$

$\forall n \in \mathbb{N}$, defined predicate $P(n)$ as: $f(n) = zpfbs(n)$

Prove by complete induction that for all natural numbers n, $P(n)$.

Proof: Let $n \in \mathbb{N}$. Assume that P is true for $0, \ldots, n-1$. I will show that $P(n)$ follows.

For the case $n \geq 2$: Partition the zero-pair-free binary strings of length n into those that end in 1 and those that end in 0. Those that end in 1 are simply those of length $n-1$. with a 1 appended, and by $P(n-1)$ (since $n-1 < n$ and $n-1 \geq 0$, $n \geq 1$), there are $f(n-1)$ of these. Those that end in 0 must actually end in 10 (otherwise they are a zero-pair), and by $P(n-2)$ (since $n-2 \leq n$ and $n-2 \geq 0$, $n \geq 2$), there are $f(n-2)$ of these. Altogether there are $f(n-1)+f(n-2)$ zero-pair-free binary strings of length n when $n \geq 2$, which is $P(n)$.

For the base case $n = 0$: There is one binary string (the empty one) of length 0, and it is zero-pair-free, and $f(0) = 1$ and $P(0)$ is true.

For the base case $n = 1$: There are two binary strings of length 1, and neither have pairs of zeros, and $f(1) = 2$ so $P(1)$ is true.

Thus in all possible cases, $P(n)$ follows. ■

7 Every natural number greater than 1 has a prime factorization

$\forall n \in \mathbb{N}$, define the predicate $P(n)$ as: n can be expressed as a product of primes.

Prime factorization: represent as product of 1 or more primes.

Prove by complete induction that for all natural numbers n, $P(n)$.

Proof: Let $n \in \mathbb{N}$ s.t. $n > 1$. Assume P is true for $2, \ldots, n-1$. I will show that $P(n)$ follows.

Case n is composite: By definition, n has a natural number factor f_1 such that $1 < f_1 < n$. By $P(f_1)$ (since $1 < f_1 < n$) we know f_1 can be expressed as a product of primes. Let $f_2 = \frac{n}{f_1}$, since $f_1 > 1$, we know that $\frac{n}{f_1} < n$, and also since $f_1 < n$, we know that $\frac{n}{f_1} > \frac{f_1}{f_1} = 1$. Since $f_1 > 1$, then $f_1 = \frac{n}{f_2} > 1$, so $n > f_2$. So, by $P(f_2)$, we know that f_2 can be expressed as a product of primes. Therefore since f_1 and f_2 are both products of primes, $n = f_1 \times f_2$ is a product of primes and $P(n)$ follows.

Case n is prime: Then n is its own primes factorization, and $P(n)$ follows.

In all possible cases, $P(n)$ follows. ■

CSC236 Week 03: Recurrences, Structural Induction

Hisbaan Noorani

September 23 – 29, 2021

Contents

1 Recursively defined function

Recall:

$$f(n) = \begin{cases} 1 & n = 0 \\ 2 & n = 1 \\ f(n-1) + f(n-2) & n > 1 \end{cases}$$

For comparison, let

$$r_1 = \frac{1+\sqrt{5}}{2},\ r_2 = \frac{1-\sqrt{5}}{2},\ a = \frac{\sqrt{5}+3}{2\sqrt{5}},\ b = \frac{\sqrt{5}-3}{2\sqrt{5}}$$

... and define

$$f'(n) = ar_1^n + br_2^n$$

Compare the first few values of $f(n)$ to the first few values of $f'(n)$

$$\begin{aligned} f(0) &= 1, & f'(0) &= 1 \\ f(1) &= 2, & f'(1) &= 2 \\ f(2) &= 5, & f'(2) &= 5 \\ f(3) &= 8, & f'(3) &= 8 \\ f(4) &= 13, & f'(4) &= 13 \end{aligned}$$

2 Prove that $\forall n \in \mathbb{N}, f(n) = f'(n)$

$\forall n \in \mathbb{N}$, define $P(n)$ by $f(n) = f'(n)$. Prove by complete induction that $\forall n \in \mathbb{N}, P(n)$. It helps to verify $1 + r_1 = r_1^2$ and $1 + r_2 = r_2^2$.

Proof: Let $n \in \mathbb{N}$. Assume the induction hypothesis, $\forall k \in \mathbb{N}$ s.t. $k < n$ has $f(k) = f'(k)$.

- Case $n > 1$:

$$\begin{aligned}
f(n) &= f(n-1) + f(n-2) && \text{(by definition)} \\
&= ar_1^{n-1} + br_2^{n-1} + ar_1^{n-2} + ar_2^{n-2} && \text{(by IH, since } 0 \leq n-2, n-1 < n\text{)} \\
&= a\left(r_1^{n-2} + r_1^{n-1}\right) + b\left(r_2^{n-2} + r_2^{n-1}\right) \\
&= a\left(r_1^{n-2}\right)(1 + r_1) + b\left(r_2^{n-2}\right)(1 + r_2) \\
&= a\left(r_1^{n-2}\right)\left(r_1^2\right) + b\left(r_2^{n-2}\right)\left(r_2^2\right) \\
&= ar_1^n + br_2^n \\
&= f'(n)
\end{aligned}$$

  And thus $\forall n > 1, P(n)$ holds.

- Case $n = 0$: $f(0) = 1 = f'(0)$, by evaluating $f'(0)$ so $P(0)$ holds.
- Case $n = 1$: $f(1) = 2 = f'(1)$, by evaluating $f'(1)$ so $P(1)$ holds.

Thus, in all possible cases, $P(n)$ holds. ■

3 Define sets inductively

One way to define the natural numbers:

$\mathbb{N}$: The smallest set such that

1. $0 \in \mathbb{N}$
2. $n \in \mathbb{N} \implies n + 1 \in \mathbb{N}$

By **smallest** we mean $\mathbb{N}$ has no proper subsets that satisfy these two conditions. If we leave out **smallest**, what other sets satisfy the definition? $\mathbb{Q}, \mathbb{R}, \mathbb{C}, \mathbb{Z},$ etc.

4 What can you do with it?

The definition in §3 defined the simplest natural number (0) and the rule to produce new natural numbers from old ones (add 1). Proof using Mathematical Induction work by showing that 0 has some property, and then that the rule to produce natural numbers preserves the property, that is

1. Show that $P(0)$ is true for basis, 0.
2. Prove that $\forall n \in \mathbb{N}, P(n) \implies P(n+1)$.

5 Other structurally-defined sets

Define $\mathcal{E}$: The smallest set such that

1. $x, y, z \in \mathcal{E}$

2. $e_1, e_2 \in \mathcal{E} \implies (e_1 + e_2), (e_1 - e_2), (e_1 \times e_2), (e_1 \div e_2) \in \mathcal{E}$

Form some expressions in $\mathcal{E}$. Count the number of variables (symbols from $\{x, y, z\}$) and the number of operators symbols from $\{+, -, \times, \div\}$. Make a conjecture:

$$(x + y), (x \times y) \in \mathcal{E}$$

$$((x + y) \times (x \times y)) \in \mathcal{E}$$

Let $\text{vr}(e)$ count the number of variables in e and let $\text{op}(e)$ count the number of operators in e.

$$\forall e \in \mathcal{E}, \text{vr}(e) = \text{op}(e) + 1$$

6 Structural induction

To prove that a property is true for all $e \in \mathcal{E}$, parallel the recursive set definition:

- **Verify the base case(s)**: Show that the property is true for the simplest members, $\{x, y, z\}$, that is show $P(x), P(y),$ and $P(z)$.
- **Inductive step**: Let e_1 and e_2 be arbitrary elements of $\mathcal{E}$. Assume $H(\{e_1, e_2\})$: $P(e_1)$ and $P(e_2)$, that is e_1 and e_2 have the property.
  - **Show that $C(\{e_1, e_2\})$ follows:**

    All possible combinations of e_1 and e_2 have the property, that is $P((e_1 + e_2)), P((e$

7 Structural induction proofs

7.1 Prove $\forall e \in \mathcal{E}, \text{vr}(e) = \text{op}(e) + 1$

Proof: For every $e \in \mathcal{E}$, define $P(e)$ as $\text{vr}(e) = \text{op}(e) + 1$. Verify the basis: Let $t \in \{x, y, z\}$. The $\text{vr}(t) = 1 = 0 + 1 = \text{op}(t) + 1$, so $P(t)$ holds for every element of the basis.

Let $e_1, e_2 \in \mathcal{E}$, and assume $P(e_1)$ and $P(e_2)$. Want to prove that $P((e_1 \odot e_2))$ where $\odot \in \{+, -, \times, \div\}$, that is $\text{vr}((e_1 \odot e_2)) = \text{op}((e_1 \odot e_2)) + 1$.

$$\begin{aligned}
\text{vr}((e_1 \odot e_2)) &= \text{vr}(e_1) + \text{vr}(e_2) \\
&= \text{op}(e_1) + 1 + \text{op}(e_2) + 1 && \text{(by } P(e_1) \text{ and } P(e_2)\text{)} \\
&= [\text{op}(e_1) + \text{op}(e_2)] + 1 \\
&= \text{op}(e_1 \odot e_2) + 1
\end{aligned}$$

So $P((e_1 \odot e_2))$ follows. ■

7.2 Prove $\forall e \in \mathcal{E}, \text{vr}(e) \leq 2^{h(3)}$

Define the heights, $h(x) = h(y) = h(z) = 0$, and $h((e_1 \odot e_2))$ as $1 + \max(h(e_1), h(e_2))$ if $e_1, e_2 \in \mathcal{E}$ and $\odot \in \{+, -, \times, \div\}$. What's the connection between the number of variables and the height?

$$\begin{aligned}
h(x) = 0, \text{vr}(x) &= 1 \\
h((x + y)) = 1, \text{vr}((x + y)) &= 2 \\
h(((x + y) - z)) = 2, \text{vr}(((x + y) - z)) & \\
h(((x + y) - (z \times x))) = 2, \text{vr}(((x + y) - (z \times x))) &= 4 \\
h((((x + y) - (z \times x)) \div ((x + y) - (z \times x)))) = 3, \text{vr}((((x + y) - (z \times x)) \div ((x + y) - (z \times x)))) &= 8
\end{aligned}$$

Proof: For every $e \in \mathcal{E}$ define $P(e)$ as $\mathrm{vr}\,(e) \leq 2^{h(e)}$. Let $a \in \{x, y, z\}$. Then a has one variable (itself), and no operators so $\mathrm{vr}\,(a) = 1 = 2^0 = 2^{h(a)}$ since h of any single variable is 0. So $P(a)$ holds for $a \in \{x, y, z\}$ (the basis).

Let $e_1, e_2 \in \mathcal{E}$. Assume $P(e_1), P(e_2)$ i.e. $\mathrm{vr}\,(e_1) \leq 2^{h(e_1)}$ and $\mathrm{vr}\,(e_2) \leq 2^{h(e_2)}$. We will show that $P(e_1 \odot e_2)$ is true, where $\odot \in \{+, -, \times, \div\}$.

$$\begin{aligned}
\mathrm{vr}\,(e_1 \odot e_2) &= \mathrm{vr}\,(e_1) + \mathrm{vr}\,(e_2) \\
&\leq 2^{h(e_1)} + 2^{h(e_2)} && \text{(by } P(e_1) \text{ and } P(e_2)\text{)} \\
&\leq 2 \cdot 2^{\max(h(e_1), h(e_2))} \\
&= 2^{1+\max(h(e_1), h(e_2))} \\
&= 2^{h((e_1 \odot e_2))} && \text{(by definition of } h\text{)}
\end{aligned}$$

So $P((e_1 \odot e_2))$ follows. ■

CSC236 Week 04: Well-Ordering, Induction Pitfalls

Hisbaan Noorani

September 30 – October 6, 2021

Contents

1 Principle of well-ordering

Every non-empty subset of $\mathbb{N}$ has a smallest element

- How would you prove this for some $S \subseteq \mathbb{N}$?

  Since $\mathbb{N}$ is bounded below by 0 and $0 \in \mathbb{N}$, we know that any subset S of $\mathbb{N}$ is also bounded below by 0. Since it is bounded below, we can say that there is a smallest element.

- Is there something similar for $\mathbb{Q}$ or $\mathbb{R}$?

  No. For example, $(0, 1) \subseteq \mathbb{Q}, \left\{\frac{1}{n} : n \in \mathbb{N}^+\right\}$.

- Here is the main part of proving the existence of a unique quotient and remainder

  $$\forall m \in \mathbb{N}, \forall n \in \mathbb{N}^+, \exists q, r \in \mathbb{N} \text{ s.t. } m = qn + r \wedge 0 \leq r < n$$

  The course notes use Mathematical Induction. Well-ordering seems shorter and clearer.

2 Well-ordering example

$\forall m \in \mathbb{N}, \forall n \in \mathbb{N}^i, R(m,n) = \{r \in \mathbb{N} : \exists q \in \mathbb{N} \text{ s.t. } m = qn + r\}$ has a smallest element.

For a given pair natural numbers $m, n \neq 0$ does the set $R(m,n)$ satisfy the conditions for well-ordering?

$$R(m,n) = \{r \in \mathbb{N} : \exists q \in \mathbb{N} \text{ s.t. } m = qn + r\}$$

If so, we still need to be sure that the smallest element, r' has

1. $0 \leq r' < n$
2. That q' and r' are unique — no other pair of natural numbers would work

... in order to have

$$\forall m \in \mathbb{N}, \forall n \in \mathbb{N}^+, \exists! q', r' \text{ s.t. } m = q'n + r' \wedge 0 \leq r < n$$

Proof. Let $m \in \mathbb{N}, n \in \mathbb{N}^+$. The $R(m,n)$ is a non-empty set of natural numbers. Then there is a smallest element of $R(m,n)$, let it be r', with corresponding q' such that $m = q'n + r'$.

- Case $r' < n$: This automatically satisfies our requirements.
- Case $r' \geq n$: Then we can show that there is another r'' such that $r'' < r'$ and r' is not the smallest element of the set.

$$\begin{aligned} m &= q'n + r' && \text{(by choice of } r') \\ &= \left(q' + 1\right) n + \left(r' - n\right) \end{aligned}$$

Let $r'' = r' - n, q'' = q' - 1$. Then $r'' \in R(m,n)$, since $m = q''n + r''$

Then $r'' < r'$ which is a contradiction, and thus the initial assumption that $r' \geq n$ is false.

So we have a unique pair of natural numbers (r', q') such that $m = q'n + r'$ and $0 \leq r' < n$. ■

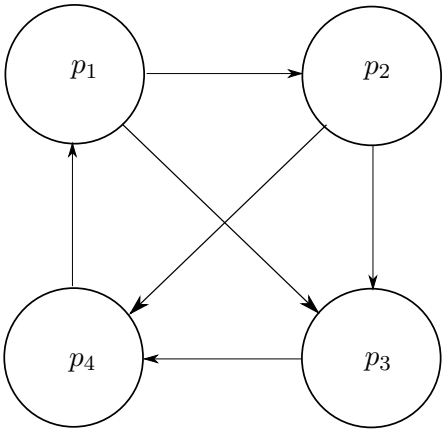
3 $P(n)$: Every round-robin tournament with n players with a cycle has a 3-cycle.

Use: every non-empty subset of $\mathbb{N}$ has a smallest element

Think of a round-robin as a complete oriented graph $G = (V, E)$, where $V = \{p_1, \ldots, p_n\}$ and every pair of distinct vertices has a uniquely directed edge between them.

If there is a cycle $p_1 \to p_2 \to p_3 \to \cdots \to p_n \to p_1$, can you find a shorter one? Can you add another directed edge without creating a 3-cycle? No, you can't.

Claim: $\forall n \in \mathbb{N}^{\geq 3}, P(n)$



Proof: Let T be a tournament with $n \geq 3$ contestants. Assume T has at least one cycle. Since there are no 1-cycles or 2-cycles (think about why), T has a cycle of length at least 3.

Let $C = \left\{c \in \mathbb{N}^{\geq 3} : c \text{ is the length of a cycle in } T\right\}$. So C is a non-empty set of natural numbers, so it must have a smallest element c'. Then, either $c' = 3$, in which case we are done, or $c' > 3$.

If $c' > 3$, there are some sub-cases to consider:

We want to show that this cycle cannot be: $p_1 \to p_2 \to p_3 \to \ldots p_{c'} \to p_1$. To do this, we can find a smaller 3-cycle that contradicts this is the smallest cycle.

- Case $p_3 \to p_1$: Then there is a shorter cycle, $p_1 \to p_2 \to p_3 \to p_1$, contradicting c' being the shortest cycle length.
- Case $p_1 \to p_3$: Then there is a shorter cycle, $p_1 \to p_3 \to \cdots \to p_{c'} \to p_1$ has length $c' - 1$, contradicting c' being the shortest cycle length.

In both cases, assuming $c' > 3$ leads to a contradiction, so this assumption is false. ■

4 No basis?

What if we try to prove $\forall n \in \mathbb{N}, \sum_{i=0}^{n} 2^i = 2^{n+1}$ (!?)

Pseudo Proof: Let $n \in \mathbb{N}$. Assume $\sum_{i=0}^{n} 2^i = 2^{n+1}$ (IH). I will show that $\sum_{i=0}^{n+1} 2^i = 2^{n+2}$.

$$\begin{aligned}
\sum_{i=0}^{n+1} 2^i &= \left[\sum_{i=0}^{n} 2^i\right] + 2^{n+1} \\
&= 2^{n+1} + 2^{n+1} && \text{(by the IH)} \\
&= 2 \cdot 2^{n+1} \\
&= 2^{n+2}
\end{aligned}$$

However, there is no verified base case!! This means that while we can prove the induction step, there is no basis to start the "infinite loop" of induction. For example, Let $n = 0, \sum_{i=0}^{n} 2^i = \sum_{i=0}^{0} 2^i = 2^0 = 1 \neq 2^1 = 2^n$.

5 Zero pair free binary strings, again but with well-ordering

$$f(n) = \begin{cases} 1, & n = 0 \\ 2, & n = 1 \\ f(n-1) + f(n+2), & n > 1 \end{cases}$$

$P(n)$ can follow simply from n being a natural number. You do not need to assume to $P(n-1)$ and $P(n-2)$ in order to prove this. Since the function is defined recursively, it is tempting to use induction, but you do not always need to.

6 How not to do simple induction

Does a graph $G = (V, E)$ with $|V| > 0$ have $|E| = |V| - 1$?

- Base case: Easy
- Inductive step: What happens if you try to extend an arbitrary graph with $|V| = n$ to one with $|V| = n + 1$? The claim breaks.
- Variations: What happens if you restrict the claim to connected graphs? If we make the graph cyclic, then the statement breaks again. What about acyclic connected graphs? In this final case, it becomes true.
- Take-home: Decompose rather than construct... except in structural induction. You never want to take an object of size n and extend it to size $n + 1$. Instead, you want to take an object of size $n + 1$, and find something related to n (your inductive hypothesis) inside of it.

CSC236 Week 05: Languages: Definitions

Hisbaan Noorani

October 7 – October 13, 2021

Contents

1 Some definitions

- **Alphabet:** Finite, non-empty set of symbols, e.g. $\{a, b\}$ or $\{0, 1, -1\}$. Conventionally denotes Σ.
- **String:** Finite (including empty) sequence of symbols over an alphabet: abba is a string over $\{a, b\}$. Convention: ε is the empty string, never an allowed symbol, Σ^* is set of all strings over Σ.
- **Language:** Subset of Σ^* for some alphabet Σ. Possible empty, possibly empty, possibly infinite subset. E.e. $\{\}, \{aa, aaa, aaaa, \dots\}$.

N.B.: $\{\} \neq \{\varepsilon\}$. *Proof:* $|\{\}| = 0 \neq 1 = |\{\varepsilon\}| \implies \{\} \neq \{\varepsilon\}$ ■

Many problems can be reduced to languages: logical formulas, identifiers from compilation, natural language processing. The key question is recognition:

Given language L and string s, is $s \in L$?

2 More notation — string operations

- **String length:** denotes $|s|$, is the number of symbols in s, e.g. $|bba| = 3$.
- $s = t$**:** if and only if $|s| = |t|$, and $s_i = s_t$, for $0 \leq i < |s|$.
- s^R**:** reversal of s is obtained by reversing symbols of s, e.g. $1011^R = 1101$.

- st **or** $s \circ t$**:** concatenation of s and t — all characters of s followed by all those of t, e.g. $bba \circ bb = bbabb$.
- s^k**:** denotes s concatenated with itself k times, e.g. $ab^3 = ababab, 101^0 = \varepsilon$.
- Σ^n**:** all strings of length n over Σ, Σ^* denotes all strings over Σ.

3 Language operations

- $\overline{L}$**:** Complement of L, i.e. $\Sigma^* - L$. If L is a language of strings over $\{0, 1\}$ that start with 0, then $\overline{L}$ is the language of strings that begin with 1 plus the empty string.
- $L \cup L'$**:** Union.
- $L \cap L'$**:** Intersection.
- $L - L'$**:** Difference.
- $\text{Rev}(L)$**:** $= \left\{s^R : s \in L\right\}$.
- **concatenation:** LL' or $L \circ L' = \{rt : r \in L, r \in L'\}$.

  There are some special cases, $L\{\varepsilon\} = L = \{\varepsilon\}L,$ and $L\{\} = \{\} = \{\}L$.
- **exponentiation:** L^k is concatenation of L, k times.

  There is a special case, $L^0 = \{\varepsilon\}$, including $L = \{\}$.
- **Kleene star:** $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$

4 Another way to define languages

In addition to the set description $L = \{\ldots\}$.

Definition: The regular expressions (regexps or REs) over alphabet Σ is the smallest set such that

- $\emptyset, \varepsilon,$ and x, for every $x \in \Sigma$ are REs over Σ.
- If T and S are REs are over Σ, then so are:
  - $(T + S)$ (union) — lowest precedence operator
  - (TS) (concatenation) — middle precedence operator
  - T^* (star) — highest precedence

5 Regular expression to languages:

The $L(R)$, the language denoted (or described) by R is defined by structural induction.

- Basis; If R is a regular expression by the basis of the definition of regular expressions, then define $L(R)$:
  - $L(\emptyset) = \emptyset$ (the empty language – no strings!)
  - $L(\varepsilon) = \{\varepsilon\}$ (the language consisting of just the empty string)
  - $L(x) = \{x\}$ (the language consisting of the one-symbol string)

- Induction step: If R is a regular expression by the induction step of the definition, then define $L(R)$:
  - $L((T + S)) = L(S) \cup L(T)$
  - $L((TS)) = L(S)L(T)$
  - $L(T^*) = L(T)*$

We are assuming about $(S + T)$ and (ST) above?

6 Regexp examples

- $L(0 + 1) = L(0) \cup L(1) = \{0, 1\}$
- $L((0 + 1)^*)$ All binary strings over $\{0, 1\}$
- $L((01)^*) = \{\varepsilon, 01, 0101, 010101, \dots\}$
- $L(0^*1^*)$ 0 or more 0s followed by 0 or more 1s
- $L(0^* + 1^*)$ 0 or more 0s or 0 or more 1s
- $L((0 + 1)\,(0 + 1)^*)$ Non-empty binary strings over $\{0, 1\}$

CSC236 Week 06: Automata and Languages

Hisbaan Noorani

October 14 – October 20, 2021

Contents

1 $L = \{x \in \{0,1\}^* : x$ begins and ends with a different bit$\}$

The language $L'((0\,(0+1)^*\,1) \cup (1\,(1+0)^*\,0))$ should be the same language as the one listed above.

If we want to prove this, we can show that $\forall x \in L, x \in L' \wedge \forall x \in L', x \in L$. This is the same as showing that $L \subseteq L' \wedge L' \subseteq L$, which is equivalent to $L = L'$, what we are trying to show.

Proof: First we show that $L' \subseteq L$: Let $x \in L'$. Then either $x = 1y0$ where $y \in \{0,1\}^*$ or $x = 0w1$, where $w \in \{1,0\}^*$. Without loss of generality, assume $x = 1y0$, otherwise just replace 1 with 0, 0 with 1, and y with w.

Then $1 \in L(1), 0 \in L(0),$ and $y \in L(0+1)^*$, since it is the concatenation of 0 or more strings from $L(0+1)$. So $x \in L(1)L(0+1)^*L(0)$, so it begins with 1 and ends with 0, which are different, so $x \in L$.

Next we show that $L \subseteq L$: this is left as an exercise to the reader

So since we have shown that $L \subseteq L' \wedge L' \subseteq L$, $L = L'$. ■

2 RE identities

Some of these follow from set properties, others require some proof

- Commutativity of union: $R + S \equiv S + R$
- Associativity of union: $(R + S) + T \equiv R + (S + T)$

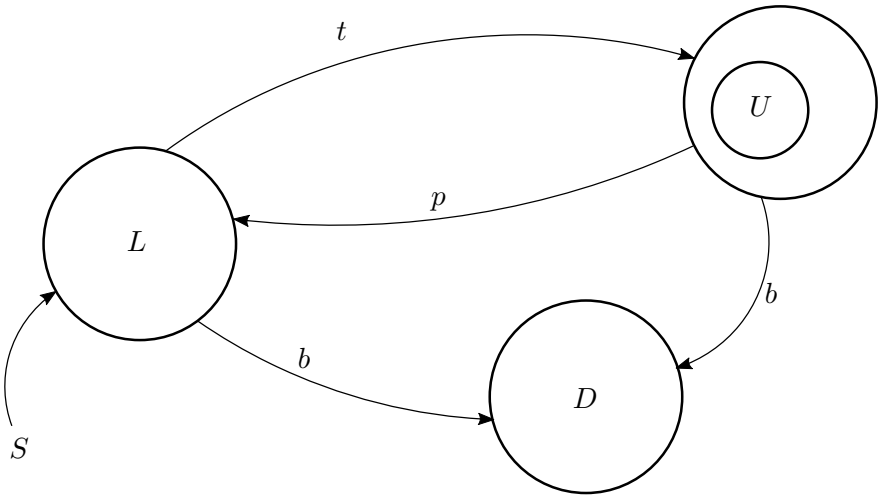
- Associativity of concatenation: $(RS)T \equiv R(ST)$
- Left distributivity: $R(S+T) \equiv RS + RT$
- Right distributivity: $(S+T)R \equiv SR + TR$
- Identity for union: $R + \emptyset \equiv R$
- Identity for concatenation: $R\varepsilon \equiv R \equiv \varepsilon R$
- Annihilator for concatenation: $\emptyset R \equiv \emptyset \equiv R\emptyset$
- Idempotence of Kleene star: $(R^*)^* \equiv R^*$

3 Turnstile finite-state machine

Let out alphabet be $\{t, p, b\}$, where p denotes push, t denotes tap or token, and b denotes bicycle.

We will study three different states: $Q = \{U, L, D\}$. U denotes unlocked, L denotes locked, D stands for dead (or jammed or deactivated).

Σ^* is all strings over $\{t, p, b\} = \{t, p, b\}^*$



Is *tptppt* accepted? We start at L, go to U with the first t, go back to L with p and so on. When we have a p but we are at L, then nothing happens as pushing on a locked turnstile will do nothing. Similarly, when we have a t when we are at a U, then nothing will happen as using a tap/token on an unlocked turnstile will also do nothing. Following these rules, we arrive at the following:

$$L \xrightarrow{t} U \xrightarrow{p} L \xrightarrow{t} U \xrightarrow{p} L \xrightarrow{p} L \xrightarrow{t} U$$

And thus the combination is accepted.

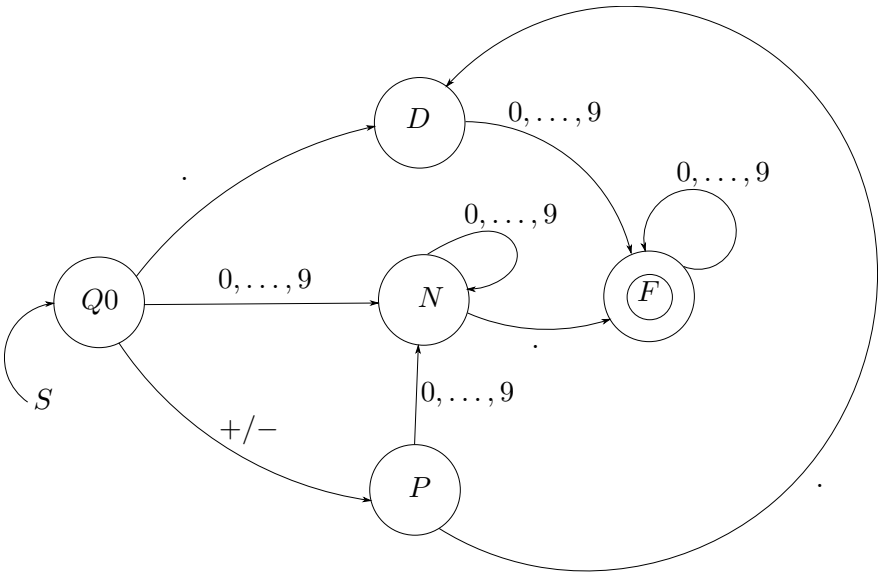
4 Float machine

Which strings are floats in Python?

$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., -, +\}$

Examples of accepted floats: $\{1.25, -0.3, .3, 125.\}$

Examples of rejected floats: $125, 1..2, 1.2.5, -., 1.25-$



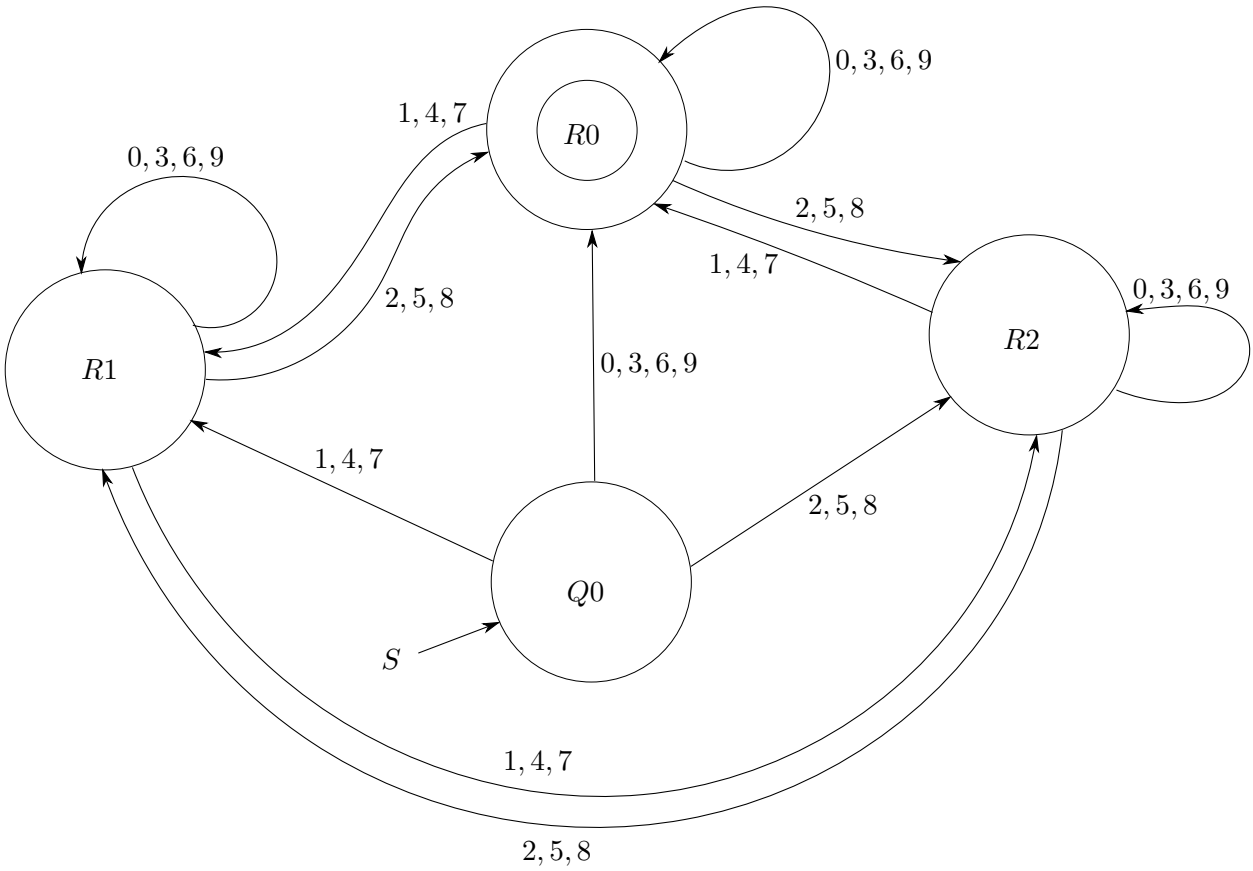
5 States needed to classify a string

What state is a stingy vending machine in, based on coins? Accepts only nickels, dimes, and quarters, no change given, and everything costs 30 cents.

δ	0	5	10	15	20	25	≥ 30
n	5	10	15	20	25	≥ 30	≥ 30
d	10	15	20	25	≥ 30	≥ 30	≥ 30
q	25	≥ 30	≥ 30	≥ 30	≥ 30	≥ 30	≥ 30

6 Integer multiples of 3

If an integer is divided by 3, it falls into one of three groups. Remainder 0, remainder 1, and remainder 2. Concatenating a number onto the end of another number can have interesting effects depicted in the diagram below.



7 Build an automaton with formalities

The idea motivating this is to be able to describe the complicated systems above without drawing out messy and time consuming diagrams.

A quintiple is a set comprised of five components: $(Q, \Sigma, q_0, F, \delta)$

Q is a set of states, Σ is a finite and non-empty alphabet, q_0 is the start state, F is the set of accepting states, and $\delta : Q \times \Sigma \mapsto Q$ is a transition function.

We can extend $\delta : Q \times \Sigma \mapsto Q$ to a transition function that tells us what state a string s takes the automaton to:

$$\delta^* : Q \times \Sigma^* \mapsto Q \text{ such that } \delta^*(q, s) = \begin{cases} q & \text{if } s = \varepsilon \\ \delta(\delta^*(q, s'), a) & \text{if } s' \in \Sigma^*, a \in \Sigma, s = s'a \end{cases}$$

Any sqquad tring s is accepted if and only if $\delta^*(q, s) \in F$, and it is rejected otherwise.

δ^* and δ are not in fact the same thing. δ takes a single valid character $\in \Sigma$, whereas δ^* takes any valid strings of characters $\in \Sigma^*$. This is why $\delta : Q \times \Sigma \mapsto Q$ and $\delta^* : Q \times \Sigma^* \mapsto Q$.

CSC236 Week 07: Autoamata and Languages

Hisbaan Noorani

October 21 – October 27, 2021

Contents

1 Example — an odd machine

A machine that accepts strings over $\{0, 1\}$ with an odd number of 0s.

We will formally prove that this description can be represented by:

$$\delta^*(E, s) = \begin{cases} E & \text{only if } s \text{ has even number of 0s} \\ O & \text{only if } s \text{ has odd number of 0s} \end{cases}$$

Define Σ^* as the smallest set of strings over Σ such that:

- $\varepsilon \in \Sigma^*$
- $s \in \Sigma^* \implies s0, s1 \in \Sigma^*$

Define $P(s)$ as $\delta^*(E, s)$ correctly defines the machine.

Proof: We will show that $\forall s \in \Sigma^*, P(s)$.

- Base Case: The following implications hold vacuously:

  $$\delta^*(E, \varepsilon) = E \implies \varepsilon \text{ has an even number of 0s}$$

  $$\delta^*(E, \varepsilon) = O \implies \varepsilon \text{ has an odd number of 0s}$$

  And thus for all members of the basis, $P(s)$.
- Inductive step: Let $s \in \Sigma^*$ and assume $P(s)$. We want to show that $P(s0)$ and $P(s1)$ hold.

$P(s0)$: If $\delta^*(E, s) = E$

$$\begin{aligned}\delta^*(E, s0) &= \delta(\delta^*(E, s), 0) \\ &= \delta(E, 0) && \text{(by the current case)} \\ &= O\end{aligned}$$

So $s0$ has an odd number of 0s and $P(s0)$ follows in this case.

If $\delta^*(E, s) = O$

$$\begin{aligned}\delta^*(E, s0) &= \delta(\delta^*(E, s), 0) \\ &= \delta(O, 0) && \text{(by the current case)} \\ &= E\end{aligned}$$

So $s0$ has an even number of 0s and $P(s0)$ follows in this case.

So $P(s0)$ holds.

$P(s1)$: If $\delta^*(E, s) = E$

$$\begin{aligned}\delta^*(E, s1) &= \delta(\delta^*(E, s), 1) \\ &= \delta(E, 1) && \text{(by the current case)} \\ &= E\end{aligned}$$

So $s1$ has an even number of 0s and $P(s1)$ follows in this case.

If $\delta^*(E, s) = O$

$$\begin{aligned}\delta^*(E, s1) &= \delta(\delta^*(E, s), 1) \\ &= \delta(O, 1) && \text{(by the current case)} \\ &= O\end{aligned}$$

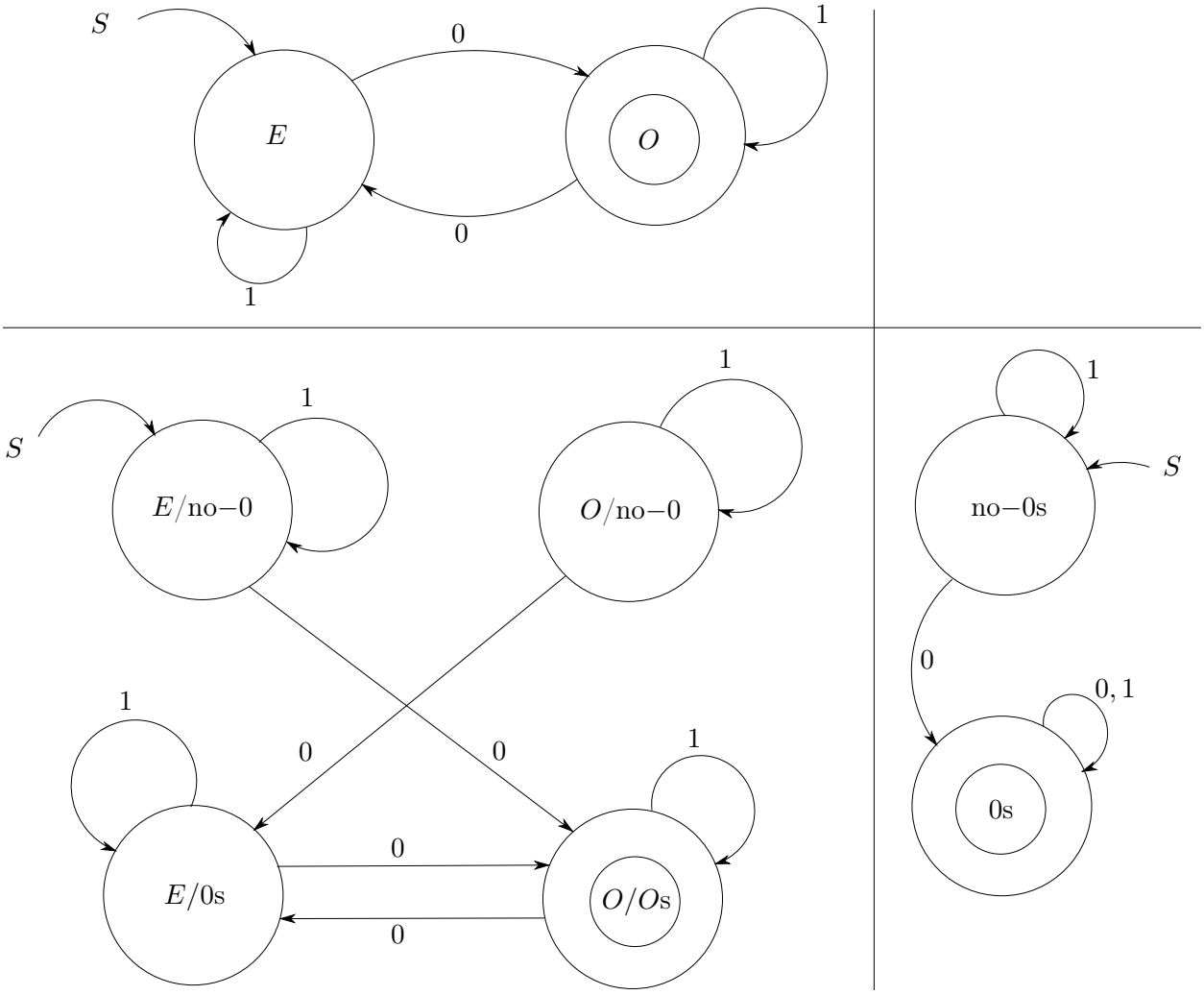
So $s1$ has an odd number of 0s and $P(s1)$ follows in this case.

So $P(s1)$ holds.

Since $P(s0)$ and $P(s1)$ both hold, we have shown that $s \in \Sigma^* \wedge P(s) \implies P(s0) \wedge P(s1)$. So $\forall s \in \Sigma^*, P(s)$, as needed. ■

2 More odd/even: intersection

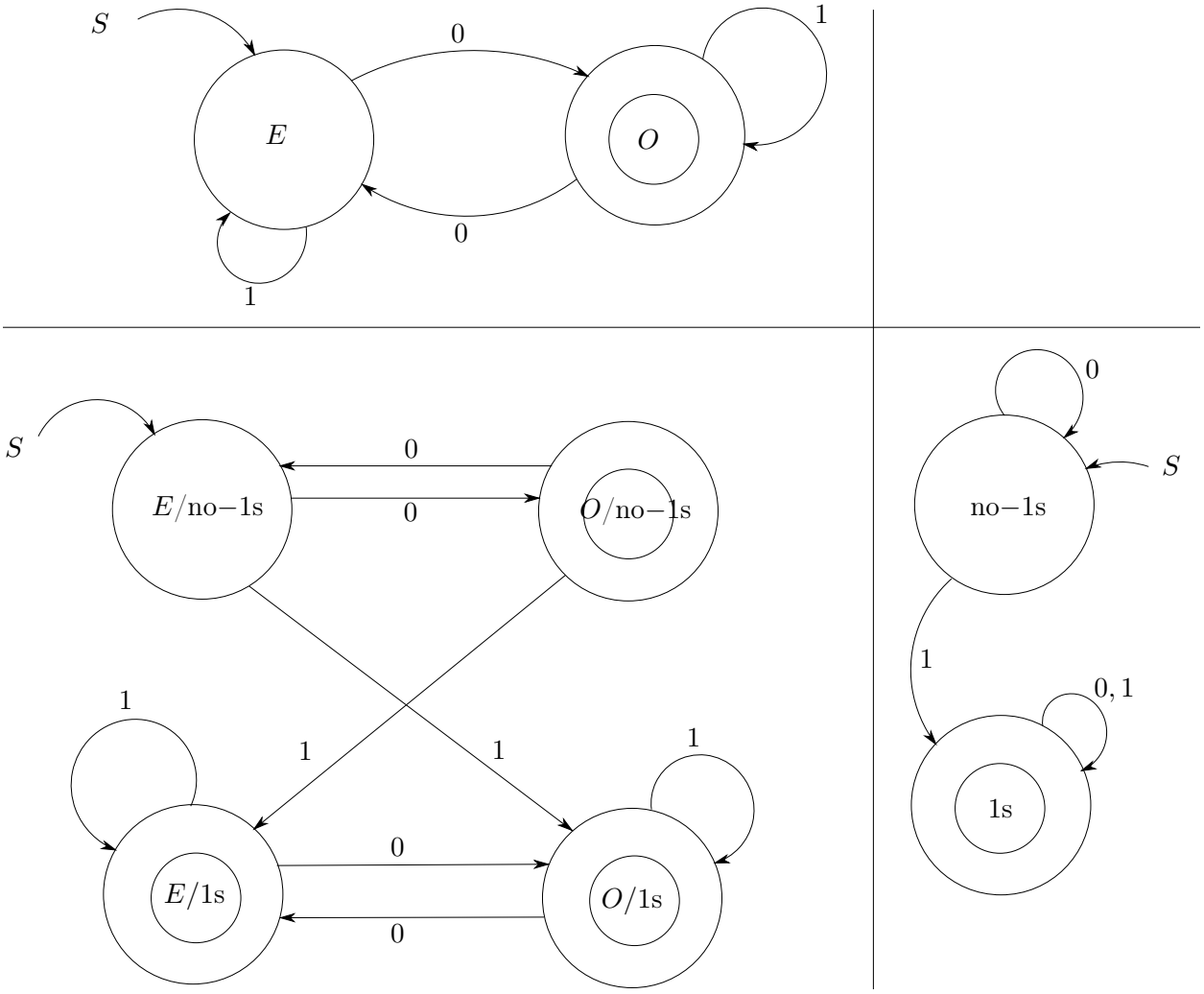
L is the langauge of binary strings with an odd number of 0s, and at least one 0. We will devise a machine for L using product construction.



If there is an odd number of 0s, we don't need to check if we have 0s or not. This means that we do not need to check for the upper right hand state. We could simply leave this state out since there are no arrows going into it.

3 More odd/even: union

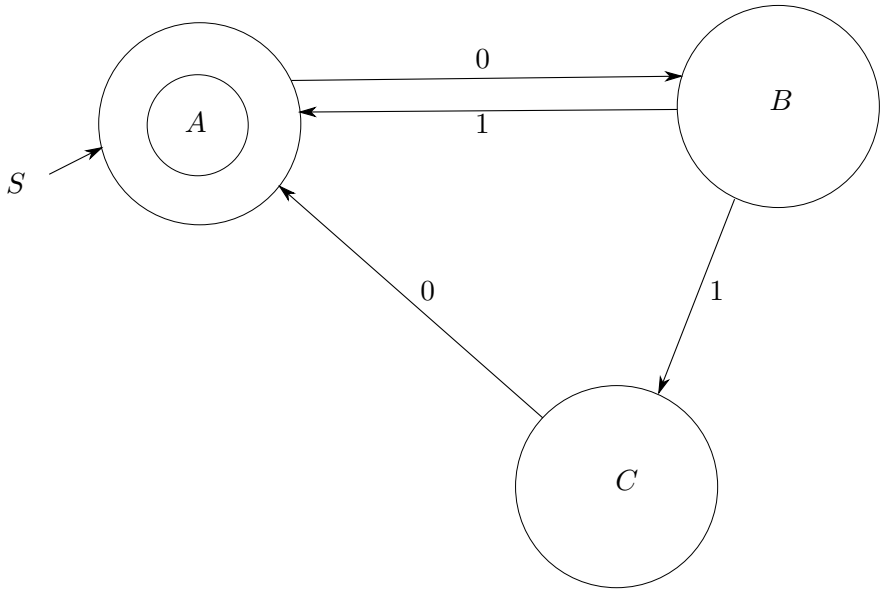
L is the langauge of binary strings with an odd number of 0s, or at least one 1. We will devise a machine for L using product construction.



4 Non-deterministic FSA (NFSA) example

FSA that accepts $L((010 + 01)^*)$. With NFSA, we allow more than one transition from a state for the same character. There is no "definite" transition for a given character, there can be different "paths" that a string would take since there can be more than one transition for a given character.

- Accepts: $\varepsilon, 01001, 0101, 01, \ldots$
- Rejects: $1, 110, 10, 01011, \ldots$



A string is accepted if there is some path that accepts the string (goes from the starting state to the accepted state).

5 NFSAs are real... you can always convert them to DFSAs

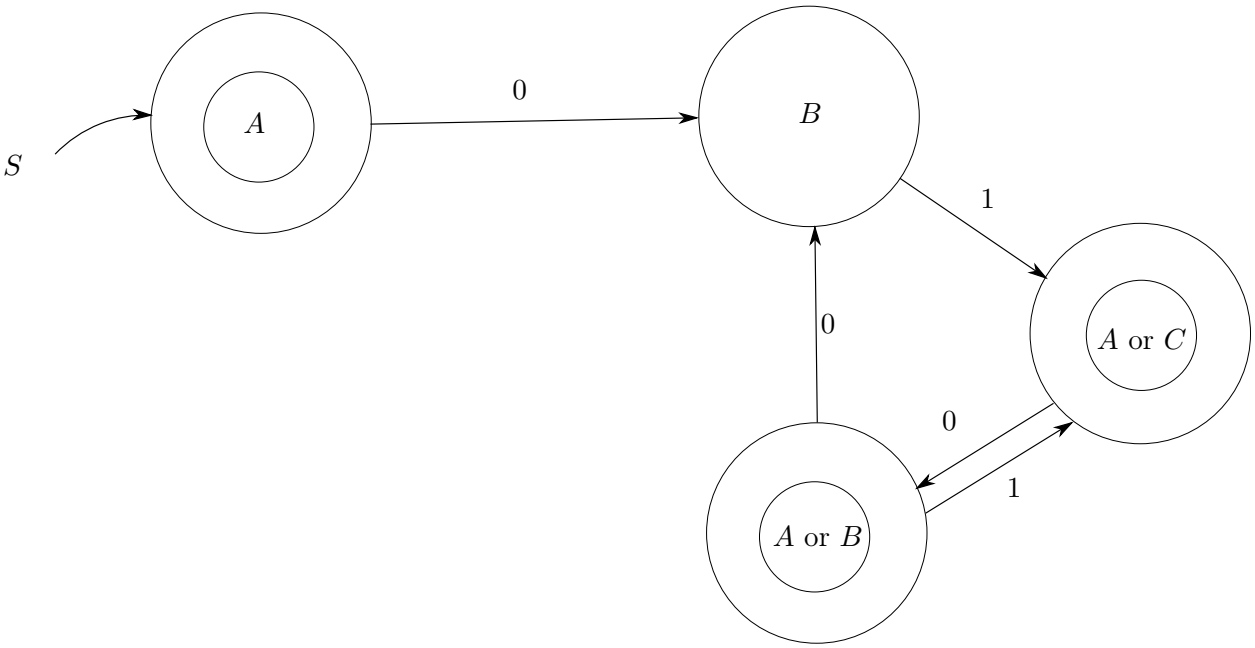
Use subset construction, more details can found in Vassos' notes page 219, but the construction is, roughly:

- Start at the start state combined with any states reachable from start with ε-transitions.
- If there are ant 1–transition from this new combined start state, combine them into a new state.
- There are any 0–translations from this new combined start, combine them into a new state.
- Repeat for every state reachable from the start.

This system lets us deal with the creation of deterministic finite state automata for regular expressions much more easily by making it a non-deterministic finite state automata and then converting it into a deterministic one.

6 Deterministic FSA (DFSA) from NFSA

We will follow the algorithm to construct a DFSA from the previous NFSA.



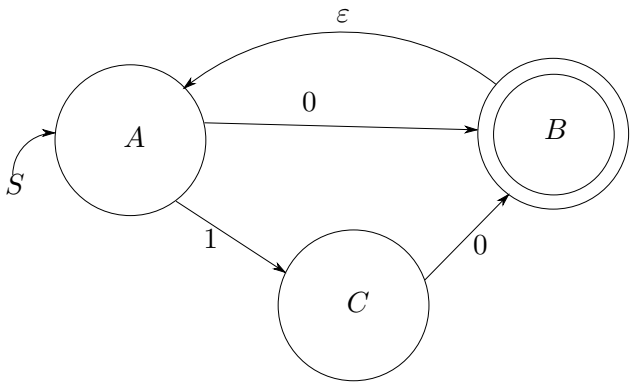
CSC236 Week 08: Machines, Expressions: Equivalence

Hisbaan Noorani

October 28 – November 3, 2021

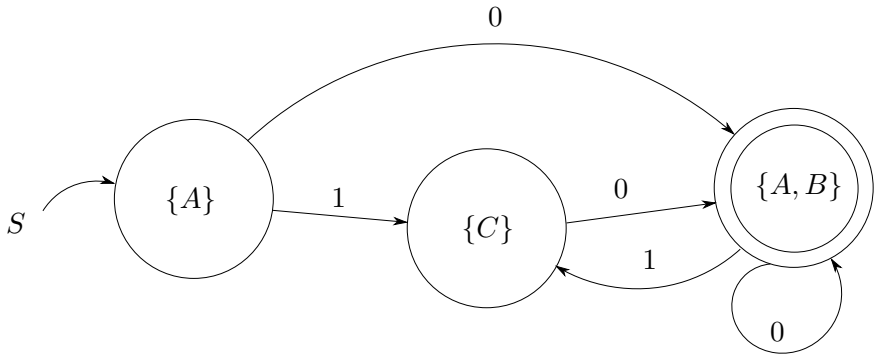
Contents

1 NFSA that accepts $L((0+10)(0+10)^*)$

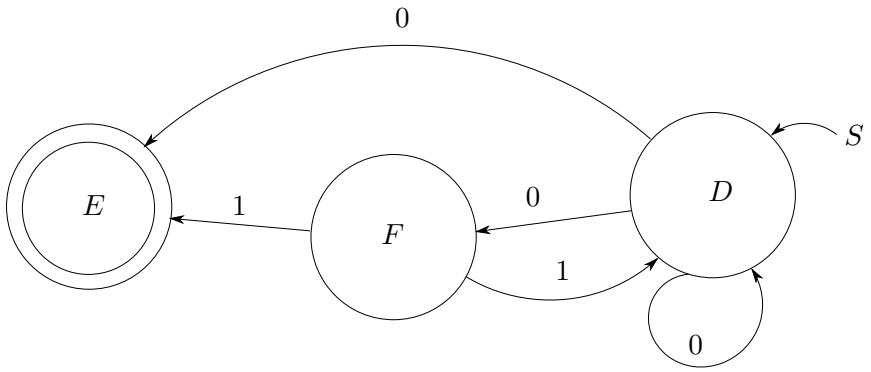


The ε transition makes it non deterministic. $A \xrightarrow{0} A \cup B$ and $C \xrightarrow{0} A \cup B$.

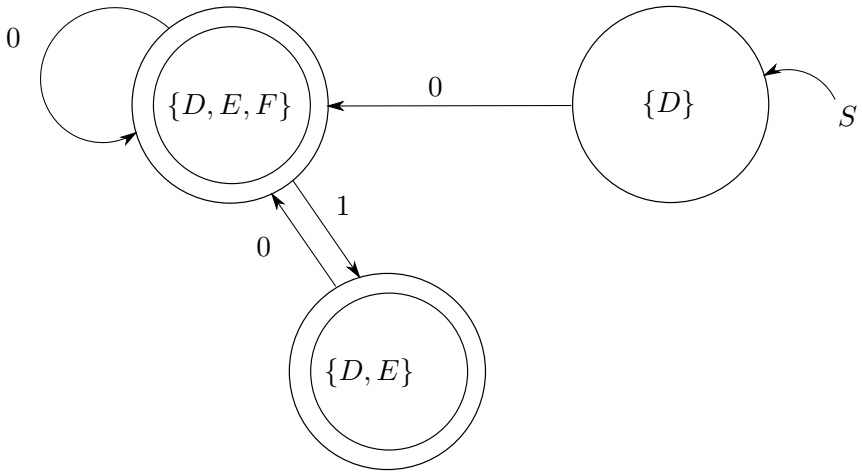
The corresponding DFSA is as follows:



2 NFSA that accepts $\mathrm{Rev}(L((0+10)(0+10)^*))$



The corresponding DFSA is as follows:



3 FSAs and regexes are equivalent.

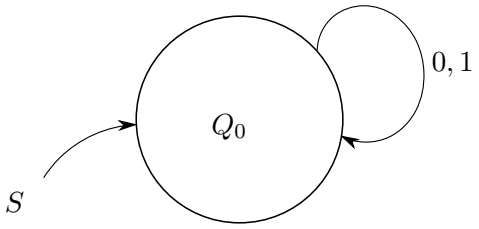
$L = L(M)$ for some DFSA $M \iff L = L(M')$ for some NFSA $M' \iff L = L(R)$ for some regular expression R.

3.1 Step 1.0: convert $L(R)$ to $L(M')$.

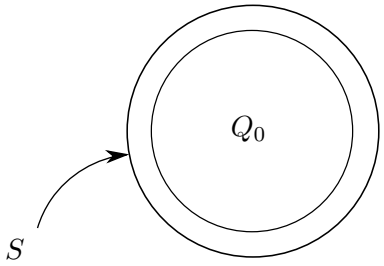
Start with $\emptyset, \varepsilon, a \in \Sigma$.

- Base case: Let $s\ in\ \{\emptyset, \varepsilon, a\}$ for some $a \in \Sigma$.

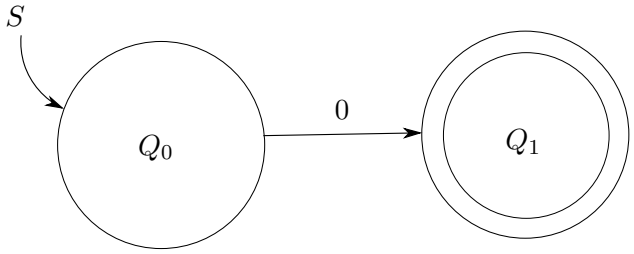
  $L(\emptyset) = L(M)$, where M is:



  $L(\varepsilon) = L(M)$, where M is:

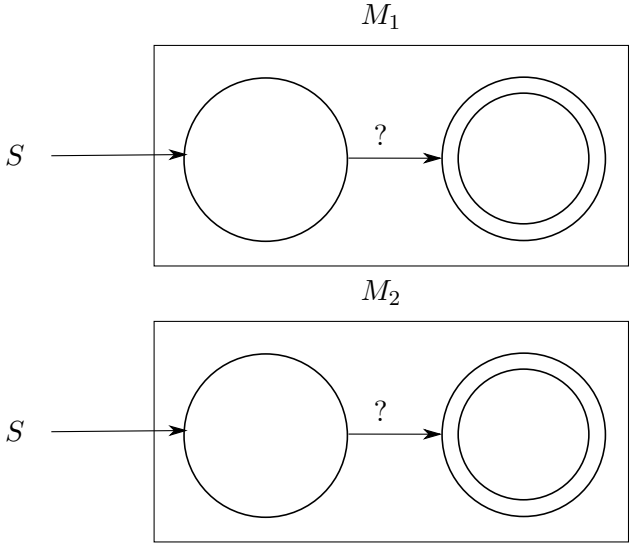


  $L(0) = L(M)$, where M is:



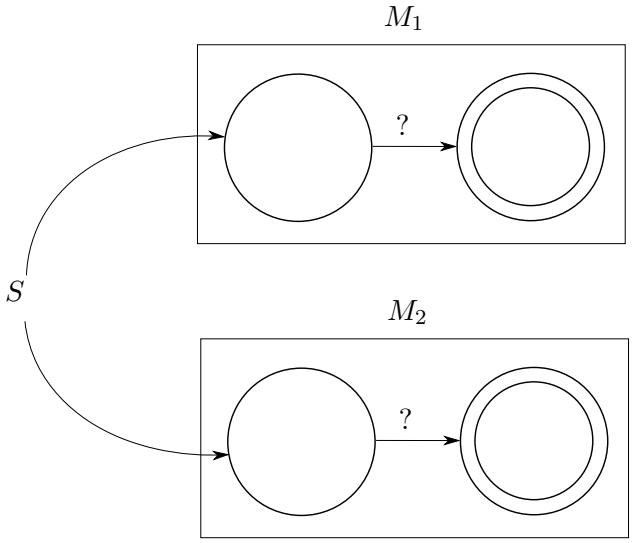
3.2 Step 1.5: Convert $L(R)$ to $L(M')$.

Suppose r_1 and r_2 denote languages accepted by M_1 and M_2 respectively.

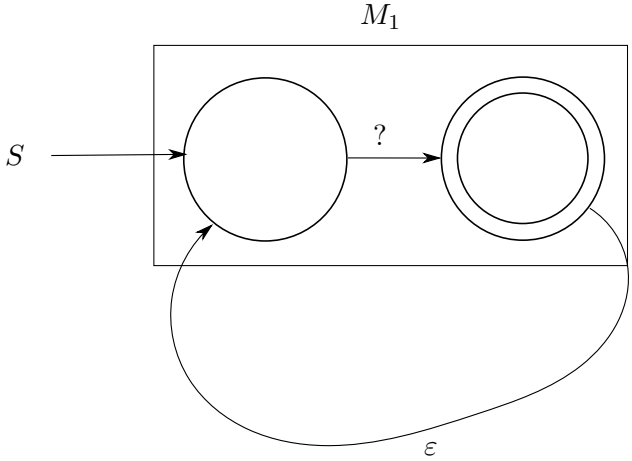


$L(r_1 + r_2) = L(r_1) \cup L(r_2)$. We could build the corresponding machine using product construction

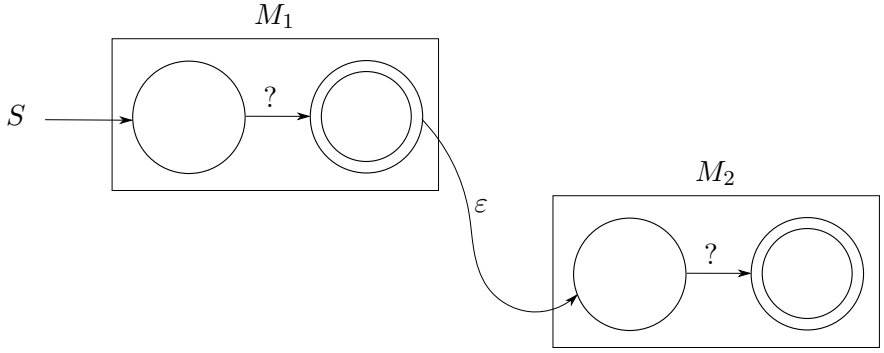
on M_1 and M_2 or we could use a NFSA.



$L(r_1^*) = L(r_1)^*$. We want to transform M_1 into a machine that accepts the Kleene star of the language accepted by M_1.



$L(r_1 r_2) = L(r_1)L(r_2)$. How do we combine M_1 and M_2 to accept this concatenated language? We concatenate the two machines!



Not that all three techniques use non-determinism but we can use subset construction to create an equivalent deterministic machine so these answers are no less valid.

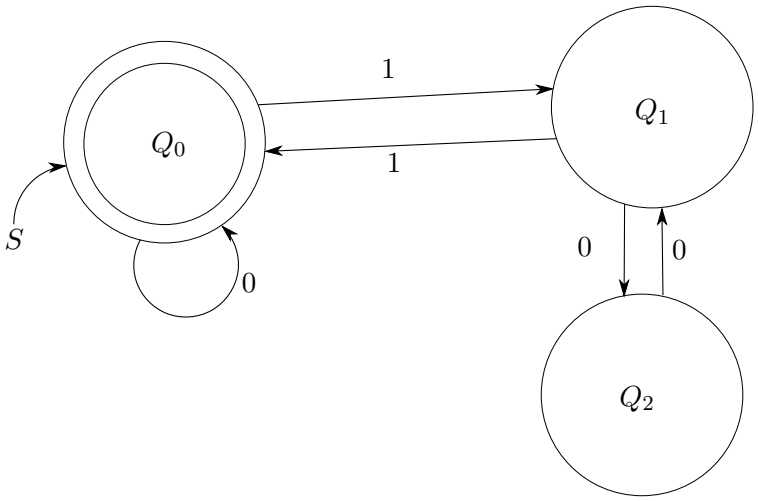
4 State elimination recipe for state q

1. $s_1 \dots s_n$ are states with transition to q, with labels $S_1 \dots S_n$.
2. $t_1 \dots t_n$ are states with transition from q, with labels $T_1 \dots T_n$.
3. Q is any self-loop state on q.
4. Eliminate q, and add (union) transition label $S_i Q^* T_j$ from s_i to t_j.

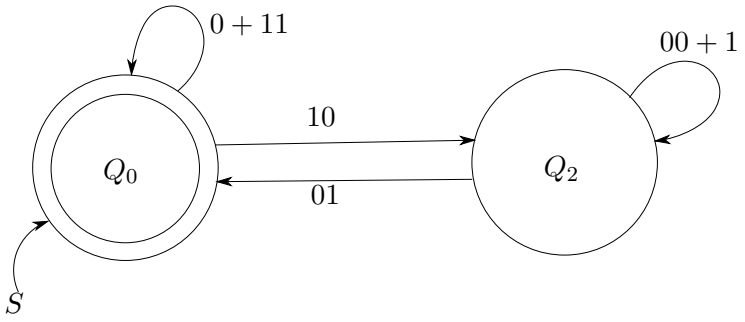
5 FSAs and regexes are equivalent:

$L = L(M)$ for some DFSA $M \iff L = L(M')$ for some NFSA $M' \iff L = L(R)$ for some regular expression R.

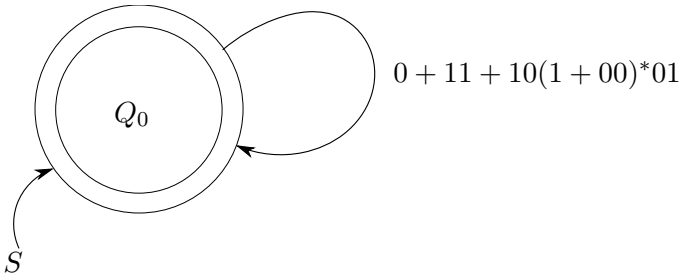
5.1 Step 3: convert $L(M)$ to $L(R)$ and eliminate states.



Then we eliminate Q_1:



Then we eliminate Q_2!



So our regular expression is $(0 + 11 + 10(1 + 00)^*01)^*$.

CSC236 Week 09: Languages: The Last Words

Hisbaan Noorani

November 4 – November 17, 2021

Contents

1 Regular languages closure

Regular languages are those that can be denoted by a regular expression or accept by an FSA. In addition:

- L regular $\implies$ $\overline{L}$ regular.
- L regular $\implies$ Rev(L) regular.

If L has a finite number of strings, then L is regular.

2 Pumping Lemma

If $L \subseteq \Sigma^*$ is a regular language, then there is some $n_L \in \mathbb{N}$ (n_L is the number of states in some FSA that accepts L) such that if $x \in L$ and $|x| \geq n_L$ then:

- $\exists u, v, w \in \Sigma^*, x = uvw$ — x is a sandwich
- $|v| > 0$ — the middle of the sandwich is non-empty
- $|uv| \leq n_L$ — first two slices no longer than n_L
- $\forall k \in \mathbb{N}, uv^k w \in L$

The general idea is if a machine $M(L)$ has $|Q| = n_L, x \in L \wedge |x| \geq n_L$, denote $q_i = \delta^*(q_0, x[:i])$, so x "visits" $q_0, q_1, \ldots, q_{n_L-1}$ with the first n_L prefixes of x (including ε)... so there is at least one state that x "visits" twice (pigeonhole principle, and x has at least $n_L + 1$ prefixes).

3 Consequences of regularity

How about $L = \{1^n 0^n : n \in \mathbb{N}\}$?

Proof: Assume, for the sake of contradiction, that L is regular. Then, there must be a machine M_L that accepts L. So M_L has $|Q| = m > 0$ states. Consider the string $1^m 0^m$. By the pumping lemma, $x = uvw$, where $|uv| \leq m$ and $|v| > 0$, and $\forall k \in \mathbb{N}, uv^k w \in L$. But, then $uvvw \in L$, so $m + |v|$ 1s followed by just m 0s. *This is a contradiction.* Elements of L must have the same number of 1s as zeros, but $m + |v| > m$. ■

4 Another approach... Myhill-Nerode

Consider how many different states $1^k \in \text{Prefix}(L)$ and end up in... for various k

Scratch work: Could 1, 11, 111 each take the machine to the same state?

Proof: Assume, for the sake of contradiction, that L (previous section) is regular. Then some machine M that accepts L has some number of states $|Q| = m$. Consider the prefixes $1^0, 1^1, \ldots, 1^m$. Since there are $m+1$ such prefixes, at least two drive M to the same state, so there are $0 \leq h < i \leq m$ such that 1^h and 1^i drive M to the same state but then $1^h 0^h$ drive the machine to an accepting state. But so does $1^i 0^h$! But $1^i 0^h$ (since $i \neq h$) sohuld not be accepted. *This is a contradiction.* By assuming that L was regular, we had to conclude there was a mchine that accepted L, which lead to a contradiction. So that assumption is false and L is not regular. ■

5 "Real life" consequences

- The proof of irregularity of $L = \{1^n 0^n : n \in \mathbb{N}\}$ suggest a proof of irregularity of $L' = \{x \in \{0,1\}^* : x \text{ has an equal number of 1s and 0s}\}$ (explain... consider $L' \cap L(1^*0^*)$)
- A similar argument implies irregularity of $L'' = \{x \in \Sigma^* : x \text{ has an equal number of}\}$

6 How about $L = \{w \in \Sigma^* : |w| = p$ is prime$\}$

Non regular. We can always find a prime that is at least as big as a given machine since there are an infinite amount of primes. There is always some bigger prime. We will prove this by the pumping lemma.

Proof: Assume for the sake of contradiction, that L is regular. So there is some machine M with $|Q| = m$ states, that accepts L. Let p be a prime number that is no smaller then m. Such a p exists since there are an infinite number of primes. Then the regular expression 1^p has length $\geq m$. This regular expression $1^p = uvw$ where $|v| > 0$, $|uv| < m$, and $uv^{kw} \in L$ for all natural numbers k. We know that $|uvw| = p$, but $\left|uv^{1+p}w\right| = p + p \cdot |v| = p \cdot (1 + |v|)$, a composite number. *This is a contradiction* since L consists of only strings of prime length. ■

7 A humble admission... (from Prof. Heap)

- At any point in time, my computer, and yours, are DFSAs

  You machine has a finite number of states, and is driven to new states by strings processed by its instruction set...

- Do the arithmetic...

  Prof. Heap's laptop has 66108489728 bits of RAM and 843585945600 bits of disk secondary storage, leading to $2^{66108489728+843585945600}$ possible different states. This is and always will be a finite number.

- However, we could dynamically add/access increasing stores of memory

  He could run down to Spadina/College to get more RAM (or storage) for bigger jobs.

- If we try and calculate this, we will find that it is impossible. We cannot calculate how many states there are in a given machine using that same machine — we will run out of ram.

8 Push Down Automata (PDA)

- DFSA plus an infinite stack with finite set of stack symbols. Each transition depends on the state, (optionally) the input symbol, (optionally) a pop from stack.
- Each transition results in a state, (optional) push onto stack.

Design a PDA that accepts $L = \{1^n 0^n : n \in \mathbb{N}\}$:

Corresponds to a context-free grammar (production rules) that denotes this language

- $S \to 1S0$
- $S \to \varepsilon$

9 Yet more power

- (informally) linear bounded automata: finite states, read/write a tape of memory proportional to input size, tape move are one position L-to-R.

  Many computer scientists think this is an accurate model of contemporary computers.

- (informally) Turing machine: finite states, read/write an infinite tape of memory, tape moves are one position L-to-R.

  This is the "ultimate" model of what computers can do.

Each machine has a corresponding **grammar** (e.g. FSAs $\leftrightarrow$ regexes (right-linear grammar))

CSC236 Week 10: Recurrences...

Hisbaan Noorani

November 18 – November 24, 2021

Contents

1 Recursive definition

This sequence comes up in applied rabbit breeding, the height of AVL tress, and the complexity of Euclid's algorithm for the GCD, and an astonishing number of other places: Define for n a natural number,

$$F(n) = \begin{cases} n & n < 2 \\ F(n-2) + F(n-1) & n \geq 2 \end{cases}$$

1.1 Fibonacci patterns

What are Fibonacci numbers multiples of? Which Fibonacci numbers are multiples of 5?

Proof: We define the predicate $P(n)$ for all natural numbers n to be: $\exists k \in \mathbb{N}$ s.t. $F(5n) = 5k$. We will prove, using simple induction that $\forall n \in \mathbb{N}, P(n)$ (Note: for every $n, F(5n+2), F(5n+3)$ are recursively defined).

- Base Case ($n = 0$): $F(5 \cdot 0) = F(0) = 05 \cdot 0$, so $P(0)$ holds,
- Inducvite step: Let $n \in \mathbb{N}$. Assume $P(n)$ i.e. $\exists k \in \mathbb{N}$ s.t. $F(5n) = 5k$. It suffices to prove that $\exists k' \in \mathbb{N}$ s.t. $F(5(n+1)) = 5k'$

Let $k' = (3k + F(5n+1))$. Then,

$$\begin{aligned}
F(5(n+1)) &= F(5n+5) \\
&= F(5n+3) + F(5n+4) \\
&= F(5n+2) + F(5n+3) + F(5n+3) \\
&= F(5n+1) + F(5n+1) + F(5n+2) + F(5n+2) + F(5n+2) \\
&= 3F(5n) + 5F(5n+1) \\
&= 3 \cdot 5k + 5F(5n+1) && \text{(By } P(n)\text{)} \\
&= 5 \cdot (3k + F(5n+1)) \\
&= 5k'
\end{aligned}$$

So $P(n+1)$ follows from $P(n)$

So $\forall \in \mathbb{N}, P(n)$. ■

1.2 Closed form for $F(n)$?

The course notes present a proof by induction that

$$F(n) = \frac{\phi^n - \left(\hat{\phi}\right)^n}{\sqrt{5}}, \phi = \frac{1+\sqrt{5}}{2}, \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

You can, and should, be able to work through the proof. The question remains, **how** did somebody ever think of ϕ and $\hat{\phi}$?

2 Binary search

The binary search function below is represented by the function $T(n)$.

Since we are not proving correctness, we don't care whether A is sorted. $|A| = n = e - b + 1$, where n is the length of the string/list/tree, b is the beginning of the search, and e is the end of the search.

```
1  def recBinSearch(x, A, b, e):
2      if b == e:              #
3          if x <= A[b]:       #
4              return b        # has time complexity c
5          else:               #
6              return e + 1    #
7      else:
8          m = (b + e) // 2 # midpoint  # has time complexity 1
9          if x <= A[m]:
10             return recBinSearch(x, A, b, m)      # has time complexity
11         else:                                    # max{T(⌊n/2⌋), T(⌈n/2⌉)}
12             return recBinSearch(x, A, m + 1, e)  #
```

Left as an exercise to the reader: Show $\lceil \frac{n}{2} \rceil = m - b + 1$. $\lfloor \frac{n}{2} \rfloor = e - m$

2.1 Guess the bound of $T(n)$ by "unwinding it"

Suppose (thoughtful wishing...) that $n = 2^k$ for some natural number k.

$$\begin{aligned}
T(n) &= T(2^k) \\
&= 1 + T(2^{k-1}) \\
&= 1 + 1 + T(2^{k-2}) \\
&= 1 + 1 + 1 + T(2^{k-3}) \\
&\quad \cdots \\
&= k + T(1) \\
&= \log_2 n + c && (T(1) = C \text{ and } 2^k = n \text{ so, } \log_2 n = k)
\end{aligned}$$

This is a guess! Even for powers of 2, there's an induction (on k maybe?) here.

2.2 Prove lower bound on $T(n)$

Proof: Let $B \in \mathbb{R}^+ = 2$. Let $c \in \mathbb{R}^+ = 1$. Let $n \in \mathbb{N}$. Assume $n \geq B$. Assume $\forall i$ s.t. $B \leq i < n, T(i) \geq c \cdot \log_2(i)$.

- Case $n \geq 3$:

$$\begin{aligned}
T(n) &= 1 + \max\left\{T\left(\left\lceil \frac{n}{2} \right\rceil\right), T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right\} \\
&\geq 1 + c\log_2\left(\left\lceil \frac{n}{2} \right\rceil\right) && (\text{By IH, since } n > \left\lceil \frac{n}{2} \right\rceil \geq B \text{ since } B \geq 2) \\
&\geq 1 + c\log_2\left(\frac{n}{2}\right) && (\text{since } \frac{n}{2} \leq \left\lceil \frac{n}{2} \right\rceil) \\
&= 1 + c\left(\log_2(n) - \log_2(2)\right) \\
&= 1 + c\left(\log_2(n) - 1\right) \\
&= (1 - c) + c\log_2(n) \\
&\geq c\log_2(n) && (\text{since } c \leq 1)
\end{aligned}$$

- Case $n = 2$: $T(2) = 1 + T(1) = 1 + c' \geq 1 \cdot \log_2(2) = c\log_2(2)$
- Case $n = 1$: $T(1) = c' \geq 1 \cdot \log_2(1) = 0 = c \cdot 0 = c\log_2(1)$

So the lower bound of $T(n)$ is $c\log_2(n)$. ■

2.3 Prove upper bound on $T(n)$

Proof: Let $B \in \mathbb{R}^+ = \ldots$. Let $c \in \mathbb{R}^+ = \ldots$ Let $n \in \mathbb{N}$. Assume $n \geq B$ and that $T(i) \leq c \cdot \log_2 i$ for all $B \leq i < n$.

It suffices to show that $T(n) \leq c\log_2(n)$

- Case $n = \ldots$:

$$\begin{aligned}
T(n) &= 1 + \max\left\{T\left(\left\lceil \frac{n}{2} \right\rceil\right), T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right\} \\
&\leq 1 + c\log_2\left(\left\lceil \frac{n}{2} \right\rceil\right) && \text{(since } B \leq \left\lceil \frac{n}{2} \right\rceil < n \text{, since } n \geq 2\text{)} \\
&\leq 1 + c\log_2\left(\frac{n+1}{1}\right) \\
&= 1 + c(\log_2(n+1) - \log_2(2)) \\
&= 1 + c(\log_2(n+1) - 1) \\
&= (1-c) + c\log_2(n+1)
\end{aligned}$$

  Oops! We want $T(n) \leq c\log_2(n)$, not just $T(n) \leq c\log_2(n+1)$. We could fiddle aroudn and make this work, by strenghtening the claim to:

$$T(n) \leq c\log_2(n-1) \leq c\log_2(n)$$

  This strengthens the IH, and it should work. Howerver, will it generalize to other recursive algorithms that use the "divide and conquer" approach?

There are other strategies that will work to prove this.

3 Reucrrence for MergeSort

```
1  MergeSort(A, b, e) -> None:
2      if b == e: return None  # c_1
3      m = (b + e) / 2
4
5      MergeSort(A, b, m)      #
6      MergeSort(A, m + 1, e)  #
7
8      # merge sorted A[b..m] and A[m + 1..e] back into A[b..e]
9      B = A[:] # copy A  # linear in n              #
10     c = b                                          #
11     d = m + 1                                      #
12     for i in [b,...,e]:                            #
13         if d > e or (c <= m and B[c] < B[d]):      #
14             A[i] = B[c]                            # linear in n
15             c = c + 1                              #
16         else: # d <= e and (c > m or B[c] >= B[d]) #
17             A[i] = B[d]                            #
18             d = d + 1                              #
```

3.1 Unwind (repeated substitution)

$T(n) = 2T\left(\frac{n}{2}\right) + n$

Assume $n = 2^k$ for some positive integer k.

$$
\begin{aligned}
T(n) &= T(2^k) \\
&= T\left(\left\lceil 2^{k-1} \right\rceil\right) + T\left(\left\lfloor 2^{k-1} \right\rfloor\right) + n && \text{(by definition)} \\
&= T\left(2^{k-1}\right) + T\left(2^{k-1}\right) + 2^k && \text{(since } 2^k \text{ is always whole)} \\
&= 2T\left(2^{k-1}\right) + 2^k \\
&\quad \ldots && \text{(repeat } k-1 \text{ more times)} \\
&= 2^k T(1) + k\left(2^k\right) \\
&= n \cdot c + \log_2(n) \cdot n && \text{(since } n = 2^k \text{ and } T(1) = c) \\
&= cn + n\log_2(n)
\end{aligned}
$$

So for the case $n = 2^k, T(n) = cn + n\log_2(n)$, which can be simplified to $T(2^k) = 2^k c + 2^k k$ (only for this specific case).

CSC236 Week 11: Recurrences...

Hisbaan Noorani

November 25 – November 1, 2021

Contents

1 Merge sort complexity, a sketch

1. Derive a recurrence to express worst-case run times in terms of $n = |A|$:

$$T(n) = \begin{cases} c' & \text{if } n = 1 \\ T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor\right) + n & \text{if } n > 1 \end{cases}$$

2. Repeated substitution/unwinding in special case where $n = 2^k$ for some natural number k leads to:

$$T(2^k) = 2^k T(1) k 2^k = c'n + n \log_2(n)$$

   We can neglect the $c'n$ term since it is of a lower order so we make the conjecture, $T(2^k) \in \Theta(n \log_2(n))$.

3. Prove T is non-decreasing (see Course Notes Lemma 3.6)

4. Prove $T \in \mathcal{O}(n \log_2(n))$ and $T \in \Omega(n \log_2(n))$

2 T is non-decreasing, see Course Notes Lemma 3.6

Exercise: prove the recurrence for binary search is non-decreasing...

Let $\hat{n} = 2^{\lceil \log_2(n) \rceil}$. We want to sandwich $T(n)$ between successive powers of 2.

$$\begin{aligned} \lceil \log_2(n) \rceil - 1 < \log_2(n) &\leq \lceil \log_2(n) \rceil \\ 2^{\lceil \log_2(n) \rceil - 1} < n &\leq 2^{\lceil \log_2(n) \rceil} \\ \frac{\hat{n}}{2} < n &\leq \hat{n} \end{aligned}$$

As an aside, try working out for $n \in \{1, 2, 3, 4, 5, 6, 7, 8\}$.

The remainder of this proof can be found in the Vassos' notes, *Lemma 3.6.*

3 Prove $T \in \mathcal{O}(n \log_2(n)$ for the general case

Let $d \in \mathbb{R}^+ = 2(2 + c)$. Let $B \in \mathbb{R}^+ = 2$. Let $n \in \mathbb{N}$ no smaller than B.

$$\begin{aligned} T(n) &\leq T(\hat{n}) && \text{(since } T \text{ is non-decreasing and } n \leq \hat{n}) \\ &= \hat{n} \log_2(\hat{n}) + c\hat{n} && \text{(by the unwinding)} \\ &< 2n \log_2(2n) + c \cdot 2n && \text{(since } \hat{n} < 2n \text{ and } \log_2 \text{ is non-decreasing)} \\ &= 2n(\log_2(2) + \log_2(n)) + 2cn \\ &= 2n((1 + c) \log_2(n) + \log_2(n)) && (\log_2(n) \geq \log_2(2) = 1, \text{ since } n \geq B) \\ &= 2n(\log_2(n))(1 + 1 + c) \\ &= 2n(\log_2(n))(2 + c) \\ &= 2n \log_2(n) + 2n(1 + c) \\ &= dn \log_2(n) && \text{(since d = 2 (2 + c))} \end{aligned}$$

This proves that T is bounded above by some constant times $n \log_2(n)$. ■

Note: in proving Ω, you will want to use the fact that $\frac{n}{2} \leq \frac{\hat{n}}{2}$.

4 Divide-and-conquer general case

Divide-and-conquer algorithms: partition a problem into b roughly equal sub-problems, solve, and recombine.

$$T(n) = \begin{cases} k & \text{if } n \leq B \\ a_1 T\left(\lceil \frac{n}{b} \rceil\right) + a_2 T\left(\lfloor \frac{n}{b} \rfloor\right) + f(n) & \text{if } n > B \end{cases}$$

Where $b, k > 0, a_1, a_2 \geq 0,$ and $a = a_1 + a_2 > 0$. $f(n)$ is the cost of splitting and recombining.

b is the number of pieces we divide the problem into.

a is the number of recursive calls.

f is the cost of splitting and recombining, we hope $f \in \Theta(n^d)$.

4.1 Divide-and conquer Master Theorem

If f from the previous slide has $f \in \Theta(n^d)$

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log_b n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Note: the complexity is sensitive to a (the number of recursive calls) and d (the degree of polynomial for splitting and recombining).

There are three steps to the proof of the Master Theorem. They are exactly parallel to the Merge Sort proof.

1. Unwind the recurrence, and prove a result for $n = b^k$. The unwinding is only valid for special values of n.
2. Prove that T is non-decreasing. See the course notes for details on how to do this.
3. Extend to all n, similar to Merge Sort. This is the easiest step, recall the technique with $\hat{n}$.

4.2 Apply the master theorem

4.2.1 Merge sort

$a = b = 2,\ d = 1$. So the complexity is $\Theta(n^1 \log_2 n)$.

4.2.2 Binary search

$a = 1,\ b = 2,\ d = 1$. So the complexity is $\Theta(n^0 \log_2 n)$.

5 Multiply lots of bits

Machines are usually able to multiply able to process integers of machine size (64-bit, 32-bit, etc.) in constant time. But what if they don't fit into machine instruction? This process takes a longer time:

$$\begin{array}{cccccccc} & & & 1 & 1 & 0 & 1 \\ & & \times & 1 & 0 & 1 & 1 \\ \hline & & & 1 & 1 & 0 & 1 \\ & & 1 & 1 & 0 & 1 & \\ & 0 & 0 & 0 & 0 & & \\ 1 & 1 & 0 & 1 & & & \\ \hline 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}$$

Let n be the number of bits of the numbers we are multiplying. We make n copies, and have n additions of 2 $n-$bit numbers. So the complexity of this "algorithm" is $\Theta(n^2)$.

5.1 Divide and recombine

Recursively, $2^n = n$ left-shifts, and addition/subtractions are $\Theta(n)$.

$$\begin{array}{rcc} & 11 & 01 \\ \times & 10 & 11 \end{array}$$

Let x_0 be the top left of this table, x_1 be the top right, y_0 be the bottom left, y_1 be the bottom right.

$$\begin{aligned} xy &= (2^{\frac{n}{2}}x_1 + x_0)(2^{\frac{n}{2}}y_1 + y_0) \\ &= 2^n x_1 y_1 + 2^{\frac{n}{2}}(x_1 y_0 + x_0 y_1) + x_0 y_0 \end{aligned}$$

The time complexity of this can be broken down as follows:

1. Divide each factor (roughly) in half ($b = 2$).
2. Multiply the halves (recursively, if they're too big) ($a = 4$).
3. Combine he products with shifts and adds (linear in number of bits).

According to the master theorem, since we have $a = 4,\ b = 2,\ d = 1$, the time complexity of this algorithm is $\Theta(n^{log_2 4})) = \Theta(n^2)$.

5.2 Gauss's Trick

Gauss rewrote the multiplication of x and y as follows, to reduce the number of individual calculations by making some of the multiplications repeat themselves.

$$xy = 2^n x_1 y_1 + 2^{\frac{n}{2}} x_1 y_1 + 2^{\frac{n}{2}}((x_1 - x_0)(y_0 - y_1) + x_0 y_0) + x_0 y_0$$

Repeated products can be stored after the first calculation, so they don't need to be calculated multiple times. So, not we have just 3 multiplications, at the cost of 2 more subtractions and one more addition. Since addition and subtractions are linear in the number of bits, this greatly reduces the complexity.

So now, using the master theorem, since $a = 3\ , b = 2,\ d = 1$, the time complexity of this algorithm is $\Theta(n^{\log_2 3}) = \Theta(n^{1.5894})$. This is, in fact, better that $\Theta(n^2)$, even if not by much.

CSC236 Week 12: Correct, Before & After

Hisbaan Noorani

December 2 – December 8, 2021

Contents

fancyverb

1 Recursive Binary Search

We define correctness of a program in terms of it running, terminating, and fullfilling what it sought out to do. Purpose: find position where either x is, or should be inserted.

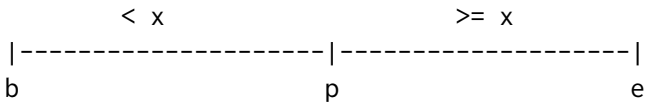
A: list, non-decreasing, comparable.

x: value to search for, must be comparable.

b: beginning index of search.

e: ending index of our search.

```
1  def recBinSearch(x, A, b, e):
2      if b == e:              #
3          if x <= A[b]:       #                1. b ≤ p ≤ e + 1
4              return b        # p return       2. b < p ⟹ A[p − 1] < x
5          else:               #                3. p < e + 1 ⟹ A[p] ≥ x
6              return e + 1    #
7      else:
8          m = (b + e) // 2 # midpoint  # ⌊(b+e)/2⌋
9          if x <= A[m]:
10             return recBinSearch(x, A, b, m)      #
11         else:                                    # ..
12             return recBinSearch(x, A, m + 1, e)  #
```



2 Conditions, pre- and post-

- x and elements of A are comparable.
- e and b are valid indices, $0 \leq b \leq e < |A|$.
- $A[b..e]$ is sorted non-decreasing.

`RecBinSearch(x, A, b, e)` terminates and also returns index p.

- $b \leq p \leq e + 1$
- $b < p \implies A[p-1] < x$
- $p \leq e \implies x \leq A[p]$

(except for boundaries, returns p so that $A[p-1] < x \leq A[p]$)

Prove that postcondition(s) follow from preconditions by induction on $n = e - b + 1$ (which is the size of the list).

3 Precondition $\implies$ termination and postcondition

Proof:

- Base Case, $n = 1$: Terminates because ther eare no loops or futher calls, returns $p = b = e \iff x \leq A[b = p]$ or $p = b + 1 = e + 1 \iff x > A[b = p - 1]$, so the postcondition is satisfied. Notice that the choice forces if-and-only-if.
- Induction step: Assume $n > 1$ and that the postcondition is satisfied for inputs of size $1 \leq k < n$ that satisfiy the precondition, and the `RecBinSearch(A, x, b e)` when $n = e - b + 1 > 1$. Since $b < e$ in this case, the check on line 2 fails, and line 8 executes.
- Exercise: $b \leq m < e$ in this case. there are two cases, according to whether $x \leq A[m]$ or $x > A[m]$.
  - Case 1: $x \leq A[m]$.
    - Show that IH applies to `RecurtiveBinarySearch(x, A, b, m)`.

      $$n = e - b + 1 > m - b + 1 \geq 1$$

    - Translate the postcondition to `RecurtiveBinarySearch(x, A, b, m)` These are now are IH:
      1. $b \leq p \leq m + 1$
      2. $b < p \implies A[p-1] < x$
      3. $p \leq m \implies A[p] \geq x$
    - Show that `RecurtiveBinarySearch(x, A, b, e)` satisfies postcondition

1. The first precondition:

$$\begin{aligned} b &\leq m+1 && \text{(by the IH)} \\ &\leq e+1 \end{aligned}$$

2. The second precondition:

$$b > p \implies A[p-1] < x \qquad \text{(by the IH)}$$

3. The third precondition:

$$p \leq e \implies p \leq m$$

Since $p = m+1 \implies A[p-1] = A[m] \implies A[m] < x$, which is a contradiction, so $p \neq m+1$, so we must have $p \leq m$.

– Case 2: $x > A[m]$

* Show that IH applies to `RecBinarySearch(A, x , m + 1, e)`. We must show that

$$\begin{aligned} n &= e+b+1 \\ &> e-(m+1)+1 \\ &\cdots \\ &\geq 1 \end{aligned}$$

* Translate postcondition to `RecBinarySearch(x, A, m + 1, e)`. These are now our IH:
  1. $m+1 \leq p \leq e+1$
  2. $m+1 < p \implies A[p-1] < x$
  3. $p \leq e \implies A[p] \geq x$
* Show that `RecBinarySearch(x, A, b, e)`
  1. The first precondition:

$$\begin{aligned} p &\leq e+1 && \text{(by the IH)} \\ b &\leq m+1 \leq p && \text{(since } b \leq m \text{ by exercise)} \end{aligned}$$

  2. The second precondition:

$$b < p \implies p = m+1 \text{ or } p > m+1 \qquad \text{(by the IH)}$$

In the case where $p > m+1$, we can say that $A[p-1] < x$ by IH #2.

In the case where $p = m+1$, we can say that $A[p-1] = A[m] < x$ by the last else statement.

  3. The third precondition:

$$p \leq e \implies A[p] \geq x$$

4 Correctbess by design

Draw bictures of before, during, and after

- Precondition: A sorted, comparable with x.
- Postcondition: $0 \leq b \leq n$ and $A[0:b] < x \leq A[p:b-1]$

4.1 "Derive" conditions from pictures

We need some notation for mutation.

- e_i will be e at the end of the i^{th} loop iteration.
- b_i will be b at the end of the i^{th} loop iteration.
- m_i will be m at the end of the i^{th} loop iteration.

Precondition: A is a sorted list comparable to x elements, $n = |A| > 0$. $0 = b \leq e = n - 1$

Postcondition: $0 \leq b \leq n$ and all($[j < xforjinA[0:b]]$) and all($[k \geq xforkinA[b:n]]$)

For all natural numbers i, define $P(i)$: At the end of hte loop iteration i (if it occurs), $0 \leq b_i \leq e_i + 1 \leq n$ and $b_i, e_i\mathbb{N}$. And, $all([j < xforjinA[0:b_i]])$ and $all([k \geq xforkinA[e_i + 1:n]])$

Prove for all $i \in \mathbb{N}$, $P(i)$ using simple induction:

5 Prove termination

Associate a decreasing seqence in $\mathbb{N}$ with loop iterations. It helps to add claims to the loop invariant.

We are tempted to "prove" that "eventaully" $b_i > e_i$. DO NOT EVER DO THIS. A more successful approach is to divise an expression linked to loop iteration i that is (1) a natural number, and (2) strictly decreases with each loop iteration. A decreasing sequence of natiral numbers must, by definition, be finite by the property of well ordering.

A good candidate for such a sequence is the "distance" of A being searched, i.e. $e_i - b_i + 1$. We want to prove that this expression is a natural number and is strictly decreasing. The expression beinga natrual number follows directly from $P(i)$. The expression being strictly decreasing has two cases.

- Case $A[m] < x$: So, $e_{i+1} = e_i$ and $b_{i+1} = m_{i+1} + 1$. Then:

$$\begin{aligned} e_{i+1} + 1 - b_{i+1} &= e_i + 1 - m_{i+1} - 1 \\ &= e_i + m_{i+1} \\ &< e_i + 1 - m_{i+1} \\ &\leq e_i + 1 - b_i \end{aligned}$$

- Case $A[m] \geq x$: So, $e_{i+1} = m_{i+1} - 1$ and $b_{i+1} = b_i$

$$\begin{aligned} e_{i+1} + 1 - b_{i+1} &= m_{i+1} - 1 + 1 - b_i \\ &= m_{i+1} - b_i \\ &< m_{i+1} + 1 - b_i \\ &\leq e_i + 1 + b_i \end{aligned}$$

In both cases, we have a decreasing sequence of natural numbers corresponding to the loop iteration. ■