CSC236 Notes

Jenci Wei

Fall 2021

Contents

1 Simple Induction

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

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

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 $H(n)$ is used and why that is valid
  - In simple induction, $C(n)$ is $H(n+1)$
- Verify *base cases*: verify that the claim is true for any cases not covered in the inductive step

2 Complete Induction

Notation:

$$\bigwedge_{k=0}^{k=n-1} P(k) := \forall k \in \mathbb{N}, k < n \implies P(k)$$

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

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

Complete induction outline

- *Inductive step*: state inductive hypothesis $H(n)$
  - Derive conclusion $C(n)$: show that $C(n)$ follows from $H(n)$, indicating where $H(n)$ is used and why that is valid
  - $H(n)$ assumes the main claim for every natural number from the starting point up to $n-1$
  - $C(n)$ is the main claim for n
- Verify *base cases*: verify that the claim is true for any cases not covered in the inductive step

3 Structural Induction

Recursively defined function: example

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

Inductively defined set: example

- $\mathbb{N}$ is the smallest set such that

  1. $0 \in \mathbb{N}$ (basis)
  2. $n \in \mathbb{N} \implies n + 1 \in \mathbb{N}$ (inductive step)

  - *Smallest*: no proper subsets satisfy these conditions

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

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

Structural induction

- Define $P(e) : vr(e) = op(e) + 1$ where
  - vr is the variable-counting function
  - op is the operator-counting function
- Verify *base cases*: show that the property is true for the simplest members, $\{x, y, z\}$; i.e. show
  - $P(x)$
  - $P(y)$
  - $P(z)$
- *Inductive step*: let $e_1, e_2 \in \mathcal{E}$. Assume $H(\{e_1, e_2\})$, i.e. $P(e_1)$ and $P(e_2)$.
  - Show that $C(\{e_1, e_2\})$ follows: all possible combinations of e_1 and e_2 have the property, i.e.
    * $P((e_1 + e_2))$
    * $P((e_1 - e_2))$
    * $P((e_1 \times e_2))$
    * $P((e_1 \div e_2))$

4 Principle of Well-Ordering

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

Proving a claim using Principle of Well-Ordering

- Assume the negation for a contradiction
- Let S be some set
- Show that S is nonempty and $S \subseteq \mathbb{N}$
- By the Principle of Well-Ordering S has a smallest element, call it s'
- From this, show that there exists an element s less than s'
- This is a contradiction, and so our assumption is false

5 Languages

Definitions

- **Alphabet**: finite, non-empty set of symbols
  - Conventionally denoted Σ
  - E.g. $\{a, b\}$, $\{0, 1, -1\}$
- **String**: finite (including empty) sequence of symbols over an alphabet
  - ϵ is the empty string
  - Σ^* is the set of all strings over Σ
  - E.g. $abba$ is a string over $\{a, b\}$
- **Language:** subset of Σ^* for some alphabet Σ. Possibly empty, possibly infinite subset
  - E.g. $\{\}, \{aa, aaa, aaaa, \ldots\}$
  - $\{\} \neq \{\epsilon\}$ since one has length 0 and the other has length 1

String operations:

- $|s|$: string length, number of symbols in s
  - E.g. $|bba| = 3$
- $s = t$: iff $|s| = |t|$ and $s_i = t_i$ for $0 \leq i < |s|$
- s^R: reversal of s, 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 in t
  - $bba \circ bb = bbabb$
- s^k: s concatenated with itself k times
  - $ab^3 = ababab$, $101^0 = \epsilon$
- Σ^n: all strings of length n over Σ
- Σ^*: all strings over Σ

Language operations:

- $\overline{L}$: complement of L, i.e. $\Sigma^* - L$
  - E.g. if L is 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
  - E.g. $\{0, 00, 000\} - \{10, 01, 0\} = \{00, 000\}$
- $Rev(L)$: $\left\{s^R : s \in L\right\}$
- LL' or $L \circ L'$: concatenation, $\{rt : r \in L, t \in L'\}$

  - E.g. $L\{\epsilon\} = L = \{\epsilon\}L$, $L\{\} = \{\} = \{\}L$
- L^k: exponentiation, concatenation of L k times
  - E.g. $L^0 = \{\epsilon\}$, even when $L = \{\}$
- L^*: Kleene star, $L^0 \cup L^1 \cup L^2 \cup \cdots$

6 Regular Expressions

The **regular expressions** over alphabet Σ is the *smallest* set such that

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

Regular expression to language

- The $L(R)$, then language denoted 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 $L(R)$:
    - $L(\varnothing) = \varnothing$ (the empty language, no strings)
    - $L(\epsilon) = \{\epsilon\}$ (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 reular expression by the induction step of the definition, then define $L(R)$:
    - $L((S + T)) = L(S) \cup L(T)$
    - $L((ST)) = L(S)L(T)$
    - $L(T^*) = L(T)^*$

Regular expression identities

- 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: $R + \varnothing = R$
- Identity for concatenation: $R\epsilon \equiv R = \epsilon R$
- Annihilator for concatenation: $\varnothing R \equiv \varnothing \equiv R\varnothing$
- Idempotence of Kleene star: $(R^*)^* \equiv R^*$

7 Deterministic Finite State Machine

Build an automaton with formalities

- Quintuple: $(Q, \Sigma, q_0, F, \delta)$
- Q is the set of states
- Σ is finite, non-empty alphabet
- q_0 is start state
- F is set of accepting states
- $\delta : Q \times \Sigma \to Q$ is transition function

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

$$\delta^* : Q \times \Sigma^* \to Q \text{ defined by } \delta^*(q, s) = \begin{cases} q, & \text{if } s = \epsilon \\ \delta(\delta^*(q, s'), a), & \text{if } s' \in \Sigma^*, a \in \Sigma, s = s'a \end{cases}$$

String s is accepted iff $\delta^*(q_0, s) \in F$, and rejected otherwise

Product construction

- $Q = Q_1 \times Q_2$
- Σ does not change
- $q_0 = \left(q_0^{(1)}, q_0^{(2)}\right)$
- $F = F_1 \times F_2$ for intersection or $\{(q_1, q_2) \in Q : q_1 \in F_1 \vee q_2 \in F_2\}$ for union
- $\delta\left((q_1, q_2), c\right) = (\delta_1(q_1, c), \delta_2(q_2, c))$

8 Non-deterministic Finite State Machine

Difference from DFSA

- δ can have multiple outputs

Convert NFSA to DFSA - **subset construction**

- E.g. $\Sigma = \{0, 1\}$
- Start at the start state combined with any states reachable from the start with ϵ-transitions
- If there are any 1-transitions from this new combined start state, combine them into a new state
- If there are any 0-transitions from this new combined start state, combine them into a new state
- Repeat for every state reachable from the start

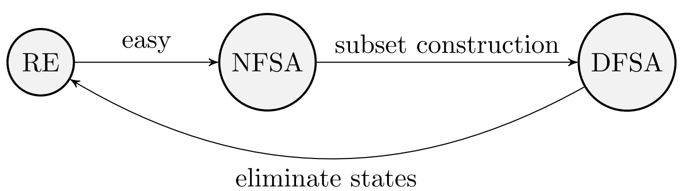
Equivalence between machines and expressions

$$\begin{aligned} & L = L(M) \text{ for some DFSA } M \\ \iff & L = L(M') \text{ for some NFSA } M' \\ \iff & L = L(R) \text{ for some regular expression } R \end{aligned}$$

Convert DFSA to regular expression - *eliminate states*

1. $s_1, \ldots, s_m$ are states with transitions *to* q, with labels $S_1, \ldots, S_m$
2. $t_1, \ldots, t_n$ are states with transitions *from* q, with labels $T_1, \ldots, T_n$
3. Q is any self-loop on q
4. Eliminate q, and union transition label $S_i Q^* T_j$ from s_i to t_j
    - Start from s_i, S_i to the former q, then Q any number of times, then T_j to the destination t_j

Summary



9 Regularity of Languages

Regular languages closure

- L regular $\implies \overline{L}$ regular
- L regular $\implies Rev(L)$ regular
- If $|L|$ is finite, then L is regular
- If L is a language in which every string has length $\leq k$ for some $k \in \mathbb{N}$, then L is regular

Pumping Lemma

- If $L \subseteq \Sigma^*$ is a regular language, then there is some $n_L \in \mathbb{N}$ such that if $x \in L$ and $|x| \geq n_L$, then
  - $\exists u, v, w \in \Sigma^*$ such that $x = uvw$ (x is a sandwich)
  - $|v| > 0$ (sandwich filling is not empty)
  - $|uv| \leq n_L$ (first two layers not bigger than n_L)
  - $\forall k \in \mathbb{N}, uv^k w \in L$ (filling can be "pumped")

Proof of irregularity using Pumping Lemma

- Assume for contradiction that L is regular
- Let $m > 0$
- Let $x = ... \in L$, satisfying $|x| \geq m$
- By Pumping Lemma, $x = uvw$, where $|uv| \leq m$, and $|v| > 0$, and for all $k \in \mathbb{N}$, $uv^k w \in L$.
- Then $uvvw \in L$, however it is not in L
- Which is a contradiction, and so the assumption is false. Therefore L is not regular.

Myhill-Nerode

- If 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" $q_0, q_1, \ldots, q_{n_L}$ with the $n_L + 1$ prefixes of x (including ϵ), so there is at least one state that x visits twice (by pigeonhole principle, and x has $n_L + 1$ prefixes)

Proof of irregularity using Myhill-Nerode

- Assume for contradiction that L is regular
- There is some FSA M that accepts L, where M has $|Q| = m > 0$
- Consider the prefixes $x^0, x^1, \ldots, x^m$, which are valid prefixes of...
- Since ther are $m + 1$ prefixes and m states, there are at least 2 prefixes that drive M to the same state, so there are $0 \leq h < i \leq m$ such that x^h and x^i drive M to the same state
- So, since $x^h y$ is accepted, $x^i y$ must also be accepted
- But $x^i y$ is not accepted
- This is a contradiction, and so the assumption is false. Therefore L is not regular

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

Linear bounded automata

- Finite states
- Read/write a tape of memory proportional to input size
- Tape moves on one position from left to right
- Most realistic model of our current computing capability

Turing machine

- Finite states
- Read/write an infinite tape of memory
- Tape moves on one position from left to right
- Model that we usually use to say what is computable

Each machine has a corresponding **grammar**

- E.g. FSAs use regexes

10 Recurrences

Recursive definition example: Fibonacci patterns

$$F(n) = \begin{cases} n, & \text{if } n < 2 \\ F(n-2) + F(n-1), & \text{if } n \geq 2 \end{cases} \qquad \text{For a natural number } n$$

Closed form for $F(n)$:

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

Mergesort complexity

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

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

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

$$T(2^k) = 2^k T(1) + k2^k = c'n + n \log n$$

   Conjecture: $T \in \Theta(n \log n)$

3. Prove T is non-decreasing
4. Prove $T \in \mathcal{O}(n \log n)$ and $T \in \Omega(n \log n)$

Divide-and-conquer general case

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

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: number of pieces we divide the problem into
- a: number of recursive calls of T
- f: cost of splitting and later re-combining the problem input

Master Theorem: if $f \in \Theta(n^d)$, then

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

- d: degree of polynomial expressing splitting/recombining costs

Master Theorem examples

- Binary search: $b = 2, d = 0, a = 1$, so the complexity is $\Theta(\log n)$
- Mergesort: $b = 2, d = 1, a = 2$, so the complexity is $\Theta(n \log n)$

To prove the Master Theorem:

1. Unwind the recurrence, and prove a result for $n = b^k$

2. Prove that T is non-decreasing
3. Extend to all n

Binary multiplication

- Want to multiply bits but they do not fit into a machine instruction
- Cut down each multiplier in $x \times y$ in the middle, resulting $x_1x_0 \times y_1y_0$

$$xy = (2^{n/2}x_1 + x_0)(2^{n/2}y_1 + y_0) = 2^n x_1y_1 + 2^{n/2}(x_1y_0 + y_1x_0) + x_0y_0$$

  - Divide each factor (roughly) in half – $b = 2$
  - Recursively multiply the halves – $a = 4$
  - Combine the products with shifts and adds – $d = 1$
  - Complexity: $\Theta(n^2)$
- Gauss's trick

$$xy = 2^n x_1y_1 + 2^{n/2}x_1y_1 + 2^{n/2}((x_1 - x_0)(y_0 - y_1) + x_0y_0) + x_0y_0$$

  - a becomes 3 since we only recursively multiplicate 3 times
  - Complexity: $\Theta(n^{\log_2 3})$

11 Recursive Correctness

Want to prove: precondition $\implies$ termination and postcondition

Proof example: by induction on n

- *Base case*: $n = \ldots$
  - Terminates because there are no loops or further calls
  - Returns $\ldots$, so postcondition satisfied
- *Induction step*: Assume $n > \ldots$ and that the postcondition is satisfied for inputs of size $1 \leq k < \ldots$, and the function terminates on such inputs.
  - Show that IH applies to the recursive call
  - Translate the postcondition to the recursive call
  - Show that the original call satisfies postcondition

12 Iterative Correctness

Loop invariant

- Come up with a loop invariant
- Prove by induction

Prove termination

- Associate a decreasing sequence in $\mathbb{N}$ with loop iterations
- By the Principle of Well-Ordering, there must be a smallest, and hence last, element of the sequence, which is linked to the last iteration
- Could add a loop invariant to do so

Prove partial correctness

- precondition $\wedge$ execution $\wedge$ termination $\implies$ postcondition
- Assume loop terminates after iteration f
- By loop condition $\ldots$, we have $\ldots$, which is the postcondition

Putting everything together, we have iterative correctness