Harvard CS121 and CSCI E-207

Lecture 2: Strings, Languages, and Induction

Salil Vadhan

September 6, 2012

Reading: Sipser, Ch. 0 and §1.1

Strings and Languages

- **Symbol** a, b, $\ldots$
- **Alphabet** A finite, nonempty set of symbols

  usually denoted by Σ
- **String** (informal) Finite number of symbols “put together”

  e.g. $abba$, b, bb

  Empty string denoted by ε
- $\Sigma^* =$ set of all strings over alphabet Σ

  e.g. $\{a, b\}^* = \{\varepsilon, a, b, aa, ab, \ldots\}$

More on Strings

- **Length** of a string x is written $|x|$

$$|abba| = 4$$

$$|a| = 1$$

$$|\varepsilon| = 0$$

The set of strings of length n is denoted Σ^n.

Concatenation

- **Concatenation** of strings

  Written as $x \cdot y$, or just xy

  Just follow the symbols of x by the symbols of y

  $x = abba$, $y = b \Rightarrow xy = abbab$

  $x\varepsilon = \varepsilon x = x$ for any x

- The **reversal** x^R of a string x is x written backwards.

  If $x = x_1 x_2 \cdots x_n$, then $x^R = x_n x_{n-1} \cdots x_1$.

Formal Inductive Definitions

- Like recursive data structures and recursive procedures when programming.

- **Strings** and their **length**:

  Base Case: ε is a string of length 0.

  Induction: If x is a string of length n and $\sigma \in \Sigma$, then $x\sigma$ is a string of length $n+1$.

  (i.e. start with ε and add one symbol at a time, like $\varepsilon aaba$, but we don't write the initial ε unless the string is empty)

- Like how one would program a string type, eg in OCaml:

```
type string = Epsilon | Append of string*char
```

Inductive definitions of string operations

- The **concatenation** of x and y, defined by induction on $|y|$.

  $[\,|y| = 0\,]$ $\quad x \cdot \varepsilon = x$

  $[\,|y| = n+1\,]$ write $y = z\sigma$ for some $|z| = n$, $\sigma \in \Sigma$
  define $x \cdot (z\sigma) = (x \cdot z)\sigma$,

- Like how one might program concatenation, eg in OCaml:

```
let rec concatenate (a:string) (b:string) : string =
  match b with
    | Epsilon -> a
    | Append(s, c) -> Append(concatenate a s, c)
```

- Such definitions are formally justified using the same Principle of Mathematical Induction used in proofs by induction.

Inductive definitions of string operations

- **Facts:** For all strings x, y, z,

  1. $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
     $\Rightarrow$ we can drop parentheses and write xyz.

  2. $\varepsilon \cdot x = x$,

- The **reversal** of x, defined by induction on $|x|$:

- Like recursive procedures to compute these operations.

Structural Induction

When doing proofs about inductively defined objects, it is often useful to perform induction on the size of the object.

Proposition: $(xy)^R = y^R x^R$ for every $x, y \in \Sigma^*$

Proof by induction on $|y|$:

Base Case: $|y| = 0$. Then $y = \varepsilon$

Induction Hypothesis: Assume $(uv)^R = v^R u^R$ for all u, v such that $|v| \leq n$

Proof, continued

Induction Step: Let $|y| = n + 1$, and say $y = z\sigma$, where $|z| = n$, $\sigma \in \Sigma$. Then:

Proofs by Induction

To prove $P(n)$ for all $n \in \mathcal{N}$:

1. "Base Case": Prove $P(0)$.

2. "Induction Hypothesis": Assume that $P(k)$ holds for all $k \leq n$ (where n is fixed but arbitrary)

3. "Induction Step": Given induction hypothesis, prove that $P(n+1)$ holds.

If we prove the Base Case and the Induction Step, then we have proved that $P(n)$ holds for $n = 0, 1, 2, \ldots$ (i.e., for all $n \in \mathcal{N}$)

Proofs vs. Programs

- There is a close parallel between formal mathematical proofs and computer programs (so doing proofs should make you a better programmer).

- BUT we generally write proofs to be read by *people*, not computers. Thus we use English prose and omit some low-level formalism when not needed to express our reasoning clearly.

- If it were just one step in a more complex proof, it would usually be OK to justify $(xy)^R = y^R x^R$ by writing

$$\begin{aligned}
&(x_1 x_2 \cdots x_{n-1} x_n y_1 y_2 \cdots y_{m-1} y_m)^R \\
&= \quad y_m y_{m-1} \cdots y_2 y_1 x_n x_{n-1} \cdots x_2 x_1 \\
&= \quad y^R x^R.
\end{aligned}$$

Detail and Formalism

You can omit some formal details (only) when:

- You are making a clear and correct claim,
- They are not the main point of what you're proving,
- You (and your reader) would be able to fill in the details if asked.

Languages

A **language** L over alphabet Σ is a set of strings over Σ (i.e. $L \subseteq \Sigma^*$)

Computational problem: given $x \in \Sigma^*$, is $x \in L$?

Every YES/NO problem can be cast as a language.

Examples of simple languages:

- All words in the *American Heritage Dictionary* $\{a, aah, aardvark, \ldots, zyzzva\}$.
- $\emptyset$
- Σ^*
- Σ
- $\{x \in \Sigma^* : |x| = 3\} = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

More complicated languages

- The set of strings $x \in \{a, b\}^*$ such that x has more a's than b's.
- The set of strings $x \in \{0, 1\}^*$ such that x is the binary representation of a prime number.
- All 'C' programs that do not go into an infinite loop.
- $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$ if L_1 and L_2 are languages.

⋮

The highly abstract and metaphorical term “language”

- A language can be either finite or infinite

- A language need not have any “internal structure”

Be careful to distinguish

ε The empty string (a string)

$\emptyset$ The empty set (a set, possibly a language)

$\{\varepsilon\}$ The set containing one element, which is the empty string (a language)

$\{\emptyset\}$ The set containing one element, which is the empty set (a set of sets, maybe a set of languages)

(Deterministic) Finite Automata

Example: Home Stereo

- P = power button (ON/OFF)
- S = source button (CD/Radio/TV), only works when stereo is ON, but source remembered when stereo is OFF.
- Starts OFF, in CD mode.
- A computational problem: does a given a sequence of button presses $w \in \{P, S\}^*$ leave the system with the radio on?

The Home Stereo DFA