Models of Computation

Ganesh Gopalakrishnan

Aug 23, 2011
Table of contents

General Announcements

More Python, and Functional Programming

LANGUAGES
Recap

- General Motivations for studying Automata Theory
- Characters and strings in Python; `ord` and `chr`
- Lists and Sequences (or tuples): two similarities / differences?
- Lists and Sets: two similarities / differences?
- Sets and Sequences: two similarities / differences?
- Definition using *comprehensions* (“set of all x such that,” etc.)
- Function `range`
Strings and Substrings

Let $s="abcd"$. Then what do these denote

- $s[0], s[1]$
- $s[1:], s[1::], s[0:]$
- $s[:]
- $s[:2], s[1::2], s[:1], s[::-1]$
- $s+s[::2]$
- \url{http://docs.python.org/release/2.5.2/lib/string-methods.html}
- \url{http://docs.python.org/library/stdtypes.html}
Let \( L = [1, 2, 3, 4] \). Then what do these denote

- \( L[0] \)
- \( L[0:3:2], L[1::2] \)
- \( L.\text{reverse}() \) does destructive reversal

Let \( L = [1, 2, 3, 4, 5] \)

- \( L1 = L \)
- \( L[::-1] \), then print \( L \), \( L1 \)
- \( L.\text{reverse}() \), then print \( L \), \( L1 \)
Lambdas

- Anonymous functions
- Function Literals (like 1993 is a number)
- In constructions such as `def fred(x, y): .., fred is redundant!`
  - What about if `fred` is recursive?
  - Still redundant!
- `lambda x: x+1`
- `lambda x, y: x+y`
- `lambda x, y=4: x+y # Overloaded use, default y=4`
- `f = lambda x: x+1`
- `def something():.... return lambda x: x+1...`
Map, Filter, Reduce

- Maps functions on lists, sets, etc.
- `list(map(lambda x: x+1, [1, 2, 3]))`
- `def f(): ... then later list(map(f, [1,2,3]))` is OK too

- `filter(lambda x: x%2 == 1, [0,1,2,3,4,5])`

- To use reduce, do from functools import *
- Given an associative operator, does tree-reduction
  - `reduce(lambda x, y: x+y, range(11))`
  - `reduce(lambda x, y: x*y, range(6))`
  - `reduce(lambda x, y: x*y, range(1,6))`
Dicts

- D = 'a': 1, 'b': 2
- D.keys()
- D.values()
- D.items()
- set(D.items())
- D.update((‘aa’: 11, ‘bb’: 22))
Languages, and Operations

- Sets of strings
  - Almost always (in this class): infinite sets
  - Always (in this class): infinite sets containing finite strings
  - Name one infinite set of strings
  - Name one infinite set of numbers
  - Name one infinite set of sets
  - Name one finite set of finite strings
  - Name one finite set of infinite strings
  - Name one infinite set of infinite strings
Language Operations

- **Empty Language (or Zero Language):** $\emptyset$ or $\{}$ We call it the “zero” language because it is like the 0-element for concatenation.

- **Unit Language:** $\{\varepsilon\}$ We call it the “unit” language because it is like the unit element for concatenation.
Language Operations

- **Concatenation**: \(L_1 L_2 = \{xy \mid x \in L_1 \land y \in L_2\}\)
- **Exponentiation**: \(L^0 = \{\varepsilon\}\) and \(L^n = LL^{n-1}\)
Language Operations

- **Union:** $L_1 \cup L_2 = \{ x \mid x \in L_1 \lor x \in L_2 \}$
- **Star:** $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$
Language Operations

- **Reverse**: \( rev(L) = \{ rev(s) \mid s \in L \} \)

- **Complementation**: Complementation of any set is with respect to a “universe” (or universal set). For language complementation, the universe is \( \Sigma^* \). Now define the complementation of a language \( L \) with respect to that universe:

\[
\overline{L} = \{ x \mid x \in \Sigma^* \setminus L \}.
\]

Again, language complements can be (and usually are) infinitary. For “simulating it in Python,” we need to bound complements:
Language Operations

- **Homomorphism on a string**: Given a string belonging to $\Sigma^*$ (a "string over $\Sigma^*$"), a function $h$ from domain $\Sigma^*$ to range $\Gamma^*$ is called a homomorphism if it respects two conditions:
  - $h(\varepsilon) = \varepsilon$
  - $h(xy) = h(x)h(y)$

- **Homomorphism on a language**: Given a homomorphism from $\Sigma^*$ to range $\Gamma^*$, it can be applied to a language $L \subseteq \Sigma^*$ to produce a language $G \subseteq \Gamma^*$, and is defined in the obvious manner:
  $h(L) = \{ h(x) \mid x \in L \}$
Language Operations

- **Intersection**: $L_1 \cap L_2 = \{ x \mid x \in L_1 \land x \in L_2 \}$
- **Language Subtraction**: $L_1 \setminus L_2 = \{ x \mid x \in L_1 \land x \notin L_2 \}$
- **Symmetric difference**: $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$