MACM 300
Formal Languages and Automata

Anoop Sarkar
`http://www.cs.sfu.ca/~anoop`

Applications of Finite-State Machines/Regular Languages

- There are many applications in computer science for finite-state machines
- We will focus here on a few canonical examples from:
  - Compilers
  - Natural language processing
  - Program verification
  - Connections with other areas of mathematics: like logic

Compilers

- A *compiler* takes program text and converts it into machine code which runs on hardware
- The first step in this conversion is to find the basic units of the program
- E.g. for the program text fragment

  `int counter = 20;`

  - We want to know that a variable `counter` has **type** integer and **value** 20

Compilers

- How can we use finite-state machines for this task?
- Let Σ = { a,…,z,A,…,Z,0,…,9}
  - Define regular expressions:

  ALPHA = a ∪ … ∪ z ∪ A ∪ … ∪ Z
  NUM = 0 ∪ … ∪ 9
  VARIABLE = ALPHA (ALPHA ∪ NUM)*
  TYPE = (int) ∪ (boolean) ∪ (real)
  INTEGER_CONSTANT = NUM (NUM)*
  EQ = '='
  END_STATEMENT = ';'
  WHITESPACE = (' ' ∪ '\n')*

Compilers

- First we convert these regular expressions to finite-state automata
- Each one recognizes a **token** in the program text
- We ask for the *longest match* for each FSA
- We remove the matched text from the input and continue until we have no more tokens
- The longest match requirement is important because we don't want to match i, then n, then t (as three variables); rather we want to match int as a type.

Compilers

- So for input text: `int counter=20;`
- When we ask for the longest match for each FSA, and continue until we have no more tokens, we get the following output:

TYPE	'int'
WHITESPACE	' '
VARIABLE	'counter'
EQ	'='
INTEGER_CONSTANT	20
END_STATEMENT	';'

- This is the first step in the conversion to machine code

Compilers

- The same techniques are used in a wide variety of other applications like text editors, search engines, etc.
- The key point is that using regular expressions we can separate the specification of a task from the implementation
- Regexp = specification
- FSA = implementation
- Automatic conversion between the two levels of representation

Natural Language Processing

- Finite-state machines are useful for NLP:
  - Speech recognition
  - Text to speech
    `http://www.naturalvoices.att.com/`
  - Machine transliteration (simple translations from one language to another)
    See handout for more information
  - Morphological analysis: walked = walk + PAST
    `http://www.fsmbook.com`

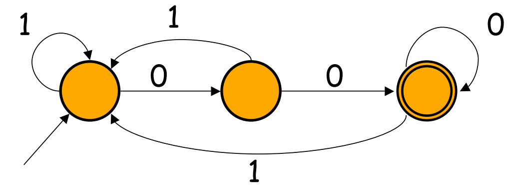
Program Verification

- Program verification is the checking of existing programs
- We want to show correctness of the program: that it will work correctly for any input
- We do this by representing all the states that the program can be in at any time as the states in a finite-state machine
- The inputs to the program are similarly taken to be the symbols of the alphabet for the FSA

Program Verification

- However, there is a problem: programs can run forever
- Also, programs can run forever *correctly*, producing the right answer for every input
- But also programs can run forever *incorrectly*: by going into an infinite loop
- We can model this in a finite-state machine by allowing it to accept *infinite strings*!

Program Verification



Consider the infinite string: 100101010101…
Similarly consider: 10010100100000000…
Acceptance is defined as reaching a final state *infinitely* often
These finite-state automata over infinite strings are called Buchi automata
For more see: `http://www.cs.rice.edu/~vardi/papers/ijcai03.ps.gz`

Connections with Logic

- This idea of automata that recognize infinite strings is also useful to prove results in logic
- Buchi automata have been used to show decidability for the logical theory of *n* successor functions
- For more see: `http://www.cs.sfu.ca/~anoop/papers/pdf/wpe2.pdf`