Introduction to the Theory of Computing

Mark Greenstreet, CpSc 421, Term 1, 2008/09

Lecture Outline

- Course Overview
- Languages
- Models of Computation

Schedule

Sept. 3 – 26: Regular languages and finite automata.

Sept. 29 – Oct. 17: Context-free languages and pushdown automata.

Oct. 8: Midterm 1

Oct. 20 – Nov. 14: Turing machines and decidability.

Nov. 5: Midterm 2

Nov. 17 – 28: To be determined.

Possible topics include:

cryptography, NP completeness, proving software correct, quantum computation, molecular computation, catching up because some topics took longer than I planned, . . .

Contact Info

- Instructor: Mark Greenstreet
  mrg@cs.ubc.ca, CICSR/CS 323
  Office hours: Monday 9-10am, Thursday 10-11am.
- TA: Brad Bingham
  binghamb@cs.ubc.ca, CICSR/CS 342
  Office hours: Let's vote!
- Course web-page: http://www.ugrad.cs.ubc.ca/ cs421.
- Course newsgroup: ubc.courses.cpsc.421
  Read it. Post questions. Claim bug-bounties!

Grading

Homework: 25%

- One assignment per week.

Midterms: 30%

- Two midterms, see schedule on slide 3.

Final: 45%

Plagiarism

Submitting the work of another person, whether that be another student, something from a book, or something off the web and representing it as your own is plagiarism and constitutes academic misconduct.

If the source is clearly cited, then it is not academic misconduct.

Bug Bounties

If I make a mistake, you get extra credit.

- If you find an error in a homework assignment or the lecture notes, post it to the course newsgroup.
- The first person to post the bug gets the bounty.
- If the error would prevent you from solving the problem, you get extra credit equal to the value of the problem.
- If it is a more minor error (or an error in the notes), I'll determine the number of extra credit points according to my sense of the severity of the error.

Lecture Outline

- Course Overview
- Languages
  - Human Languages
  - Programming Languages
  - Formal Languages
- Models of Computation

Human Languages

English, French, Danish, Hungarian, Urdu, Cantonese, . . .

Which sentences below are true, meaningful, grammatical?

- vgrlum qp#d*n aoiuiui brubrubrubru 3jc6r
- dog homework ate my. My
- Erpa shumblers groffed dulky brubrus.
- Iron is denser than styrofoam.
- The textbook for this class has exactly ten pages.
- Two is less than three.
- The loneliness sat for cast iron subtraction.
- George W. Bush is smarter than a dead slug.

Programming Languages

C, Java, Python, Prolog, Pascal, . . .

When is a program:

- syntactically correct?
- compilable?
- free from fatal exceptions at runtime?
- free from deadlock or infinite loops?
- a correct implementation of its specification?

Formal Languages

- An alphabet, Σ, is a finite set of symbols, e.g. $\{\clubsuit, \dagger, \oplus, \nabla\}$.
- A string is a sequence of zero or more symbols from Σ, e.g. $\clubsuit \oplus \oplus$ or $\dagger \dagger \dagger$.
- We'll write ϵ to denote the empty string (the string consisting of zero characters).
- We'll write Σ^* to denote all strings consisting of symbols from Σ.
- A language, L, is a subset of Σ^*.

Formal Language, example

- let $\Sigma = \{\mathsf{a}, \mathsf{b}, \wedge, \vee, \neg, \mathtt{(}, \mathtt{)}\}$.
- We could define L_0 to be the set of all strings that represent syntactically correct boolean formulas.
- We could define L_1 to be the set of all strings that represent boolean tautologies.
- Example strings:

  a ∧ b is in L_0 but not L_1.

  a ∨ ¬a is in L_0 and L_1.

  (a ∨ b ∨ (¬a ∧ ¬b)) is in L_0 and L_1.

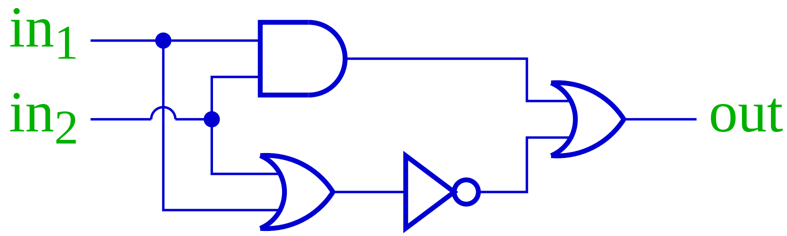
  (a ∨ ∧ b is not in L_0 and not in L_1.

- We can write a computer program that determines whether or not an arbitrary string is in L_0 or in L_1.

Lecture Outline

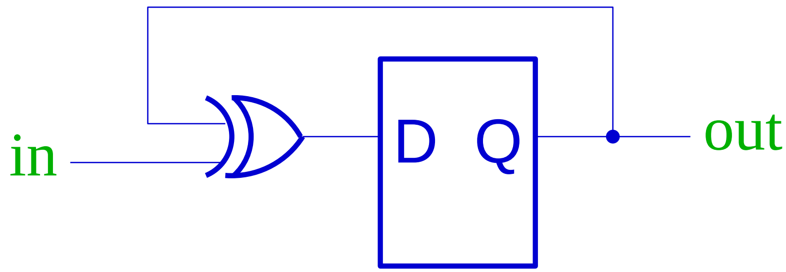
- Course Overview
- Languages
- Models of Computation
  - Logic gates
  - Finite automata
  - Push-down automata
  - Turing machines

Logic Gates



- “Language” is set of all inputs that produce a true output value.
- Any circuit only accepts fixed number of bits for input – not a true language in the sense described above.
- Note that two-input NAND gates are universal for logic circuits. Any boolean function can be constructed using only two-input NAND gates.

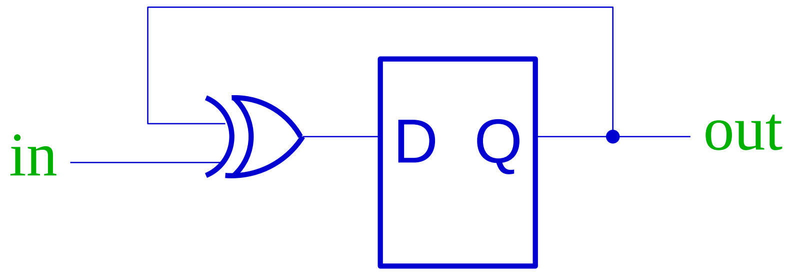
Finite Automata



Initially: out = 0

- Logic gates plus a fixed number of bits of storage.
- Can process an arbitrarily long strings.
  The example circuit accepts all strings with an odd number of ones.
- The languages that can be recognized by finite automata are very restricted.
  - For example, finite automaton can't recognize inputs that have more `1`'s than `0`'s or mathematical formulas where the parentheses balance properly.

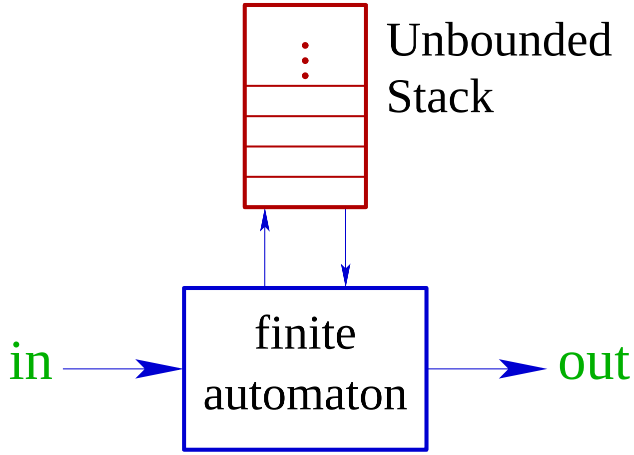
Finite Automata



Initially: out = 0

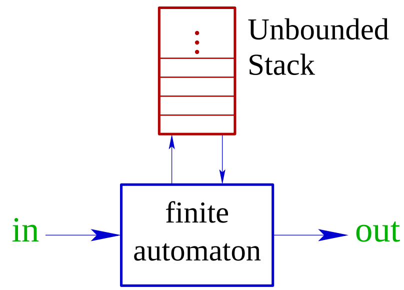
- Logic gates plus a fixed number of bits of storage.
- Can process an arbitrarily long strings.
  The example circuit accepts all strings with an odd number of ones.
- The languages that can be recognized by finite automata are very restricted.
  - For example, finite automaton can't recognize inputs that have more `1`'s than `0`'s or mathematical formulas where the parentheses balance properly.
  - Intuitvely, this is because a machine with a fixed number, k, bits of storage can only count to 2^k. After reading $2^k + 1$ `1`'s, the machine must be in a state that it was in before.

Push-Down Automaton



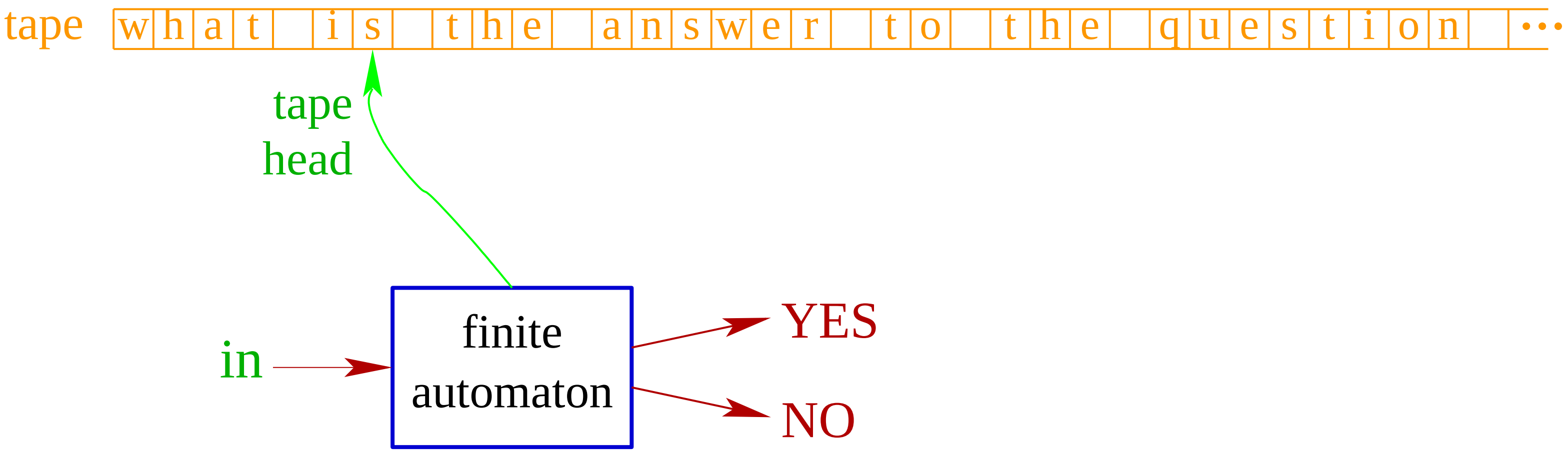
- A finite automaton with an unbounded stack.
- Can recognize properly balanced parantheses and other languages with nesting structures.
- Most programming languages have syntaxes with this kind of nesting structure.
- More general than finite automata, but still limited.
  - Cannot recognize the language of all strings whose lengths are prime numbers.

Push-Down Automaton



- A finite automaton with an unbounded stack.
- Can recognize properly balanced parantheses and other languages with nesting structures.
- Most programming languages have syntaxes with this kind of nesting structure.
- More general than finite automata, but still limited.
  - Cannot recognize the language of all strings whose lengths are prime numbers.
  - Intuitvely, a machine with a stack can only recognize lanuages with tree-like syntaxes, and we can “graft” copies of a subtree into an existing tree. If a language doesn’t have subtrees like this, then it can’t be recognized by a pushdown automaton.

Turing Machines

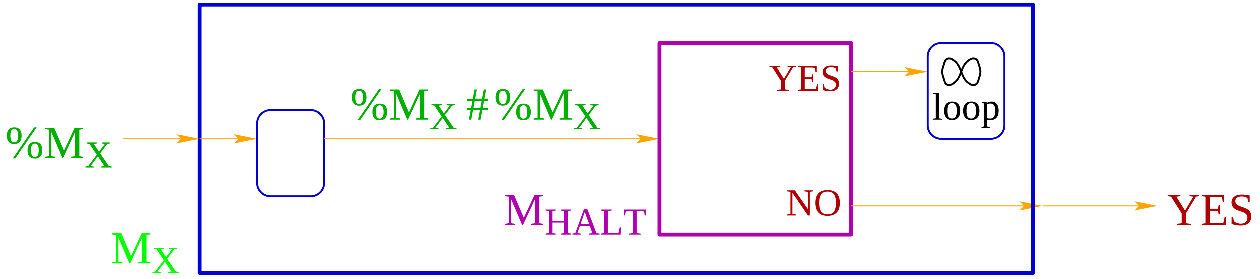


- A finite automaton with an unbounded read/write tape.
- Can recognize any language that is recognizable by ANY computer!
- Yet, there are problems that a Turing machine cannot solve.

The Halting Problem

- Let `halt(p, in)` be a function that is true iff program p halts when run with input in.
- Note that both `p` and `in` can be strings.
  If # is a symbol that cannot be in p or in, we can write both as a single string: $p\#in$.
- The set of all strings $p\#in$ such that program p halts when run with input in is a language, HALT.
- If there were a Turing machine that recognizes HALT, we could call such a machine M_{HALT}.
- We could now build a Turing machine, M_X that when run on input string s:
  - Creates string $s\#s$.
  - Runs M_{Halt} on $s\#s$.
  - If M_{Halt} recognizes $s\#s$
    then M_X goes into an infinite loop.
    else M_X halts.
- What happens if we run M_X with a string describing M_X as its input?

The Halting Problem (cont.)



- If M_{HALT} accepts $\%M_X\#\%M_X$,
  - That means that M_X will halt when run with its own description, $\%M_X$ as input.
  - But, M_X will go into an infinite loop if M_{HALT} accepts.
- If M_{HALT} rejects $\%M_X\#\%M_X$,
  - That means that M_X will run forever when run with its own description, $\%M_X$ as input.
  - But, M_X will go exit immediately loop if M_{HALT} rejects.
- M_{HALT} cannot give a correct answer.
- Note that M_X was constructed using a proposed M_{HALT}. We get a different M_X for each proposed solution, M_{HALT}, but this shows that no solution to the halting problem exists.

What's the "Theory of Computing"?

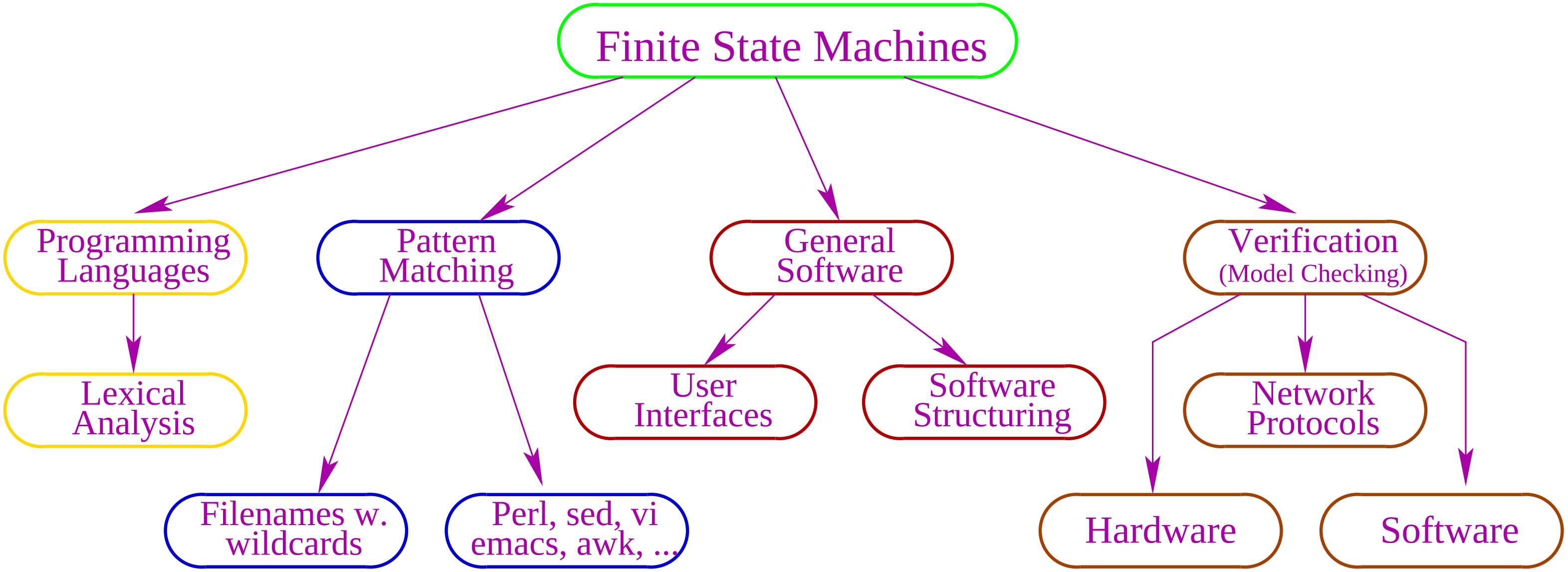
Here's the kinds of questions we consider:

- **1.** What problems are possible/impossible to solve with a computer?
- **2.** What problems are easy/hard to solve with a computer?
- **3.** What is a computer?
- **4.** Do do the answers to 1 and 2 depend on the answer to 3?

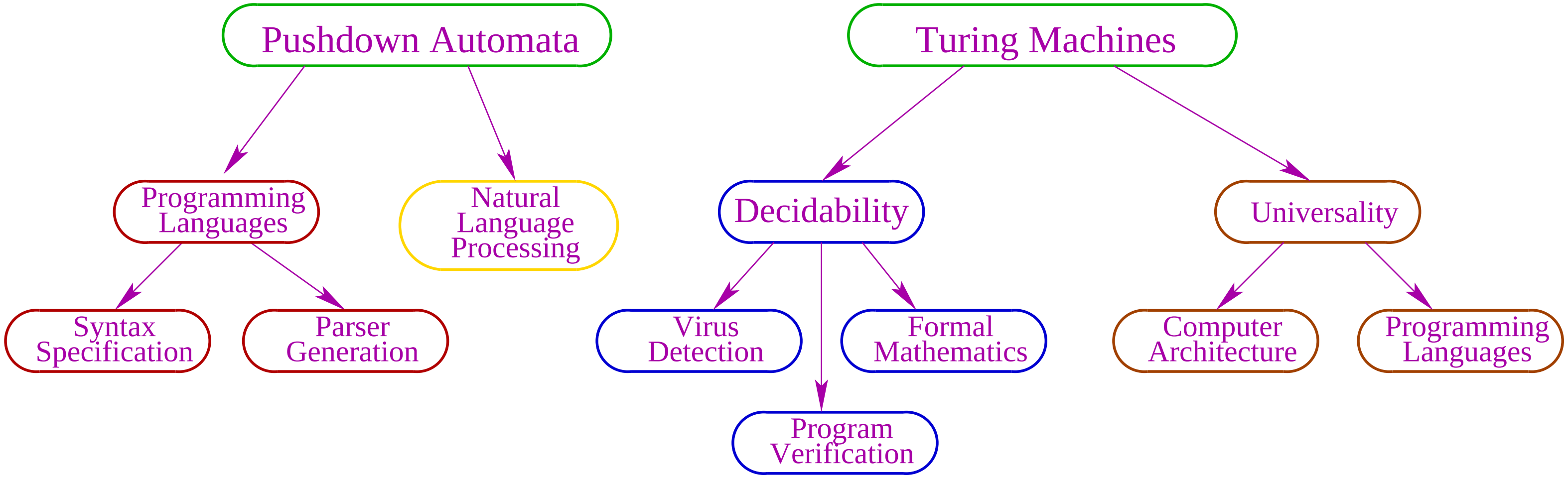
What is a computer?

- Finite state machines:
  A fixed amount of memory.
- Pushdown automata:
  An infinite amount of memory, arranged as a stack.
- Turing machines:
  An infinite amount of memory, arranged as a tape with a “head” that can read, write, and move left or right.
  A Turing machine is very simple but can perform any computation that a conventional comptuter can do. In fact, we don’t know of anything that can compute something that a Turing machine cannot.

Connections



Connections



Summary

- You've now seen everything in this course:
  - Finite Automata, regular languages, and their limitations,
  - Push-Down Automata, context-free languages, and their limitations,
  - Turing machines, general languages, and their limitations
- What's left to cover:
  - Go over the material at a reasonable pace so everyone can understand it.
  - Make it mathematically precise.
  - Look at practical implications and applications.