Transducers, Building Bigger Turing Machines

CS154

Chris Pollett

Apr 11, 2007.

Outline

- Transducers
- Ways of Building Larger Turing Machines
- The Church-Turing Thesis

Transducers

- On Monday, we defined what it means for a TM to recognize and to decide a language.
- Another useful thing that we can do with a Turing Machine is to compute functions. We call these kinds of TMs **tranducers.**
- For example, we might need to compute a function to convert the problem of deciding membership for one language into the problem of deciding membership in another language.
- We say a function f with domain D is **computable**, if there is some TM M which when started on inputs w from D, halts with f(w) on the tape.
- We will give in a moment a TM which on w computes w_w. The book shows how + and * can be computed on a TM.

Ways of Building Larger Turing Machines

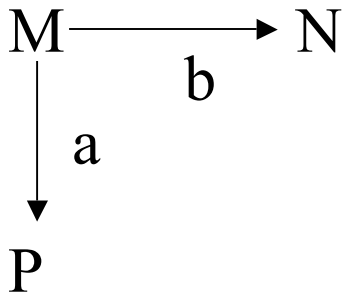
- We now turn our attention briefly to how to build larger Turing Machines from little Turing Machines.
- You can think of this process as very much like building larger and larger functions starting with little functions.

A Simple Turing Machine

- The transition function is the most important part of a TM's description.
- We will sometimes use a graphical notation to describe TM's and in particular this function.
- Given *a* in Σ define a machine $M_a = \{\{s,h\}, \Sigma, \Sigma \cup \{\_\}, \partial, s, \{h\}\}$, where for each *b* in $\Sigma \cup \{\_\}$, $\partial(s,b) = (h, a, R)$.
- That is, if *a* is a symbol, the only thing M_a does is write that symbol, move right, and halt.
- Similarly, if *a* is L or R we can build a machine such that the only thing M_a does is either move one square left or right.

Building Bigger TMs

- Given three TMs with a common alphabet: M, N, P, we can build a new machine M´ which operates as follows:
  - Start in the initial state of M; operate as M until M would halt, then
  - if the currently scanned symbol is an *b*, start N
  - if the currently scanned symbol is an *a*, start P.
  - halt otherwise.
- Diagrammatically we write:
- As an exercise you should work out
  what M´'s transition function would look like.
- This shows TMs can do if-then-else statements
- If one of N or P was instead a loop back to M we would have a while-loop. Hence, it should become clear we are starting to be able to simulate many programming language constructs.



More on Diagrams

- Similar to the if-else type diagram of the last slide we can have diagrams like:

  M ----> N

  Notice there is no label on the arrow. This means that if machine M is about to transition to its halt state h we instead have it transition to the start state of N.

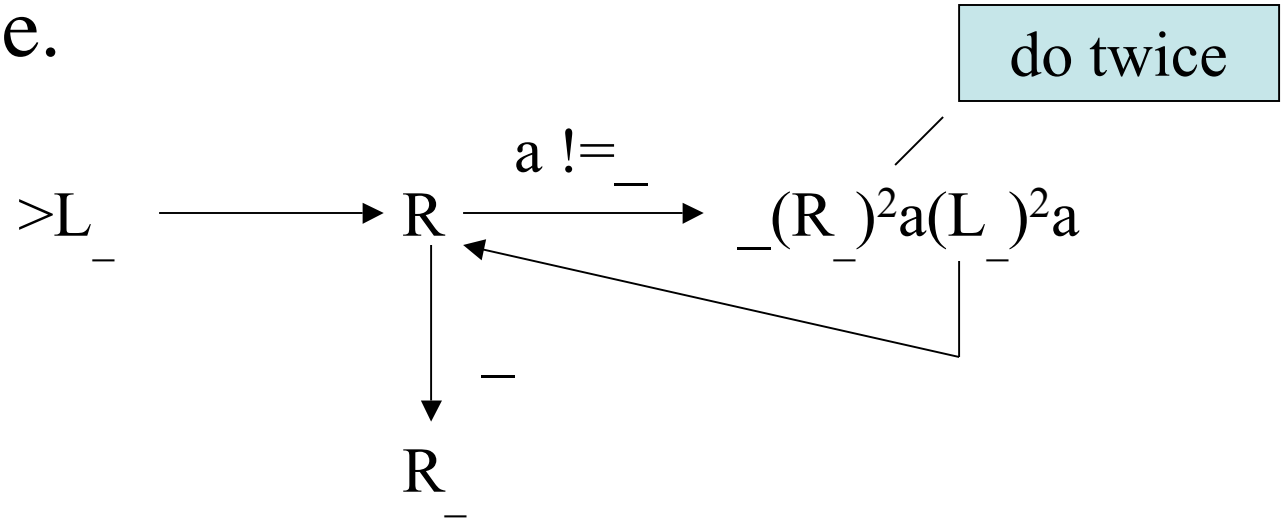
- We can also generalize the two branch construction of the previous slide to any fixed finite number of branches. This would allow us to simulate switch-case like commands.

Examples

- We sometimes abbreviate M_R as R and M_a as a. We might also make abbreviations like Ra for the machine which does M_R, then reading any symbol writes an a. Similarly, we might have RR or La.
- Let !*a* denote all the symbols in ∑ except *a*.
- Here is a machine $R_{\_}$ that scans right to the first space 
- Here is a machine $L_{\_}$ that scans left to the first space 

More Examples

- Here is a machine which when started with a string w on the tape halts with w_w on the tape.



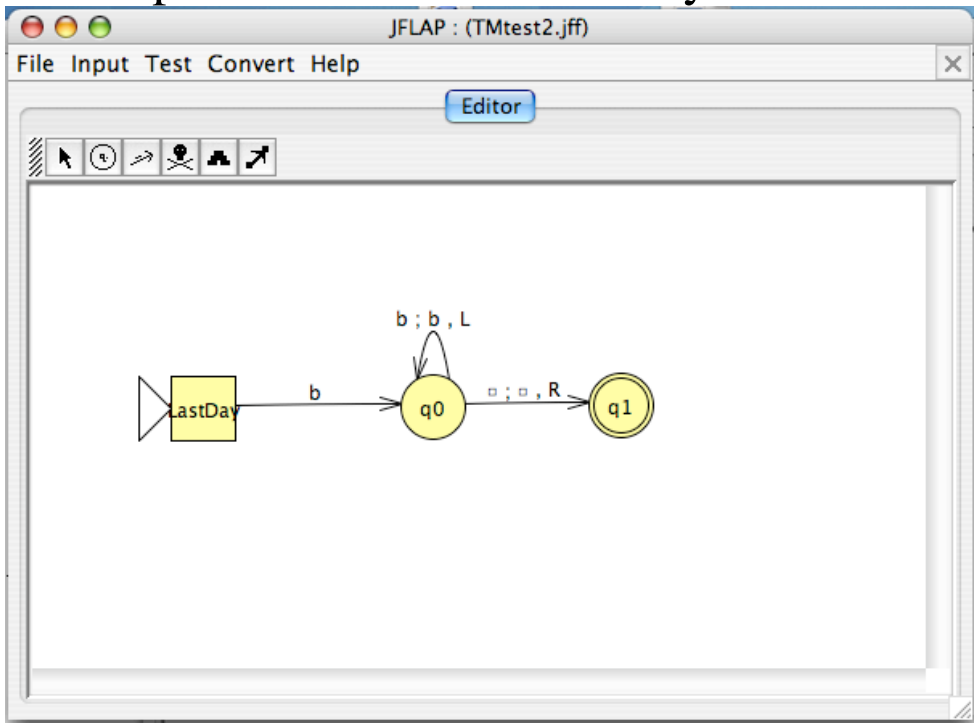
JFLAP

- JFLAP supports building larger machines out of smaller machines.
- If one clicks on the Turing Machine button in JFLAP, one can start building a Turing Machine.
- The two rightmost buttons:

Allow one to respectively import Turing Machines you've already created as building blocks, and allow you to set up transitions between Turing Machines.

JFLAP Example

- For example, recall that on Monday we built a machine in JFLAP that converts strings over a's and b's into one over just b's.
- It finishes with the tape head on the rightmost symbol.
- We can combine this machine with a machine to "rewind the tape" to finish with the tape head on the leftmost symbol:



The Church Turing Thesis

- This thesis is that any computational process that can be effectively carried out on a real-world computational device can be simulated by a Turing Machine.
- Since you can always come up with new computational devices, it is not something you can prove.
- However, so far no one since it was proposed in the 1940s has come up with a model which is both physically implementable which cannot be simulated by a Turing Machine.
- Over the next week, we will look at some of the models that have been considered.
- These include: multi-tape variants of Turing Machines, multi-stack PDAs, RAMs (computers with machine code instructions), cellular automata, lambda calculus (based on functional programming approaches like in LISP or scheme), rewrite systems, classical and quantum physics based processes, etc.