

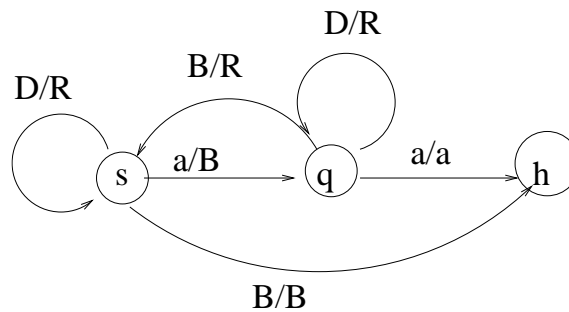
Study guide on Turing Machines, CS301, Spring '09

Modified 5/7 9:00

The final exam will be on May 12, 1:30-3:30 in the usual classroom. It will be comprehensive, focusing primarily on problems that are similar to ones that you have worked in the homeworks. It will be similar in format to the second midterm. Only a few easy questions will concern Turing machines, and two extra-credit problems will concern NP-completeness. See the posting on NP-completeness to see how to pick up the extra-credit points.

1. Look at the formal definition of Turing machines give in Definition 4.1.1. Why, according to this definition, is the Turing machine deterministic? *Hint: Look at the definition of δ and recall what we mean by a function, looking up function in the index to find the definition, if necessary.*
2. As in the case of PDA's, we will obtain a graphical representation of a Turing machine labeling transitions with what determines which transition the machine can take (the letter at the current position on the tape) and what action it takes if it does.

The following depicts the Turing machine of Example 4.1.1. Because my graphics utility doesn't allow special characters, I will denote a right move by R , a left move by L , and a blank character by B .

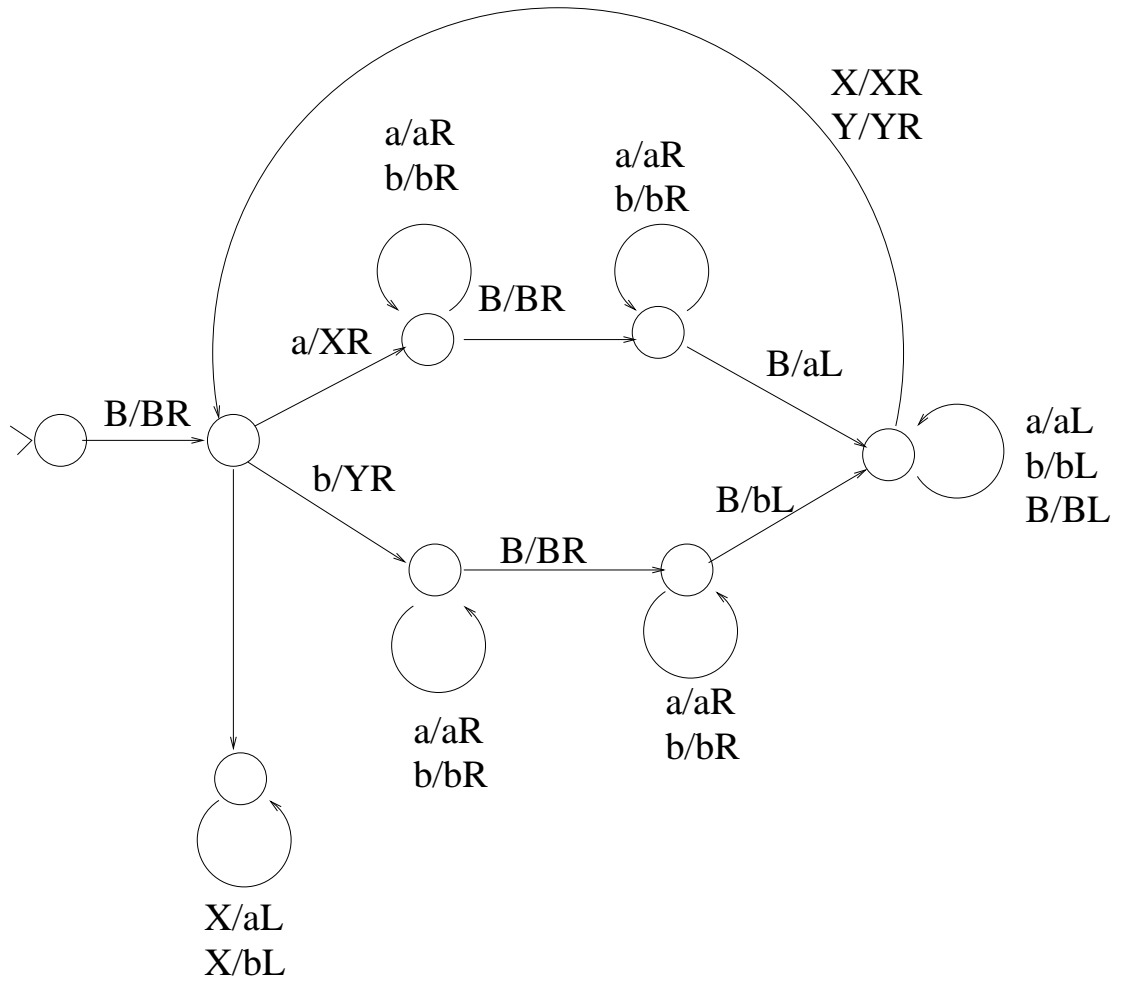


(Clarification: confusion arose in class when I miscopied the B . on the transition (s, q) as a small b , rather than a capital B .)

3. According to our book, an action consists of either writing a letter to the current position, moving right one position, or moving left one position. Let us allow an action to consist of to writing a letter to the current position *and/or* moving left or right one position.

By now, you know enough proof techniques to show that our new model is no more powerful than the book's. Make sure you can come up with an argument.

Here's a variant of the book's Example 4.1.3, which makes a copy of a string u . The input tape starts with a blank, followed by a string u , followed by blanks. It ends with a blank, followed by u , followed by another blank, followed by another copy of u .



4. Here's a Turing machine that recognizes even-length palindromes. The string is initially stored immediately following the left-end marker of the tape.

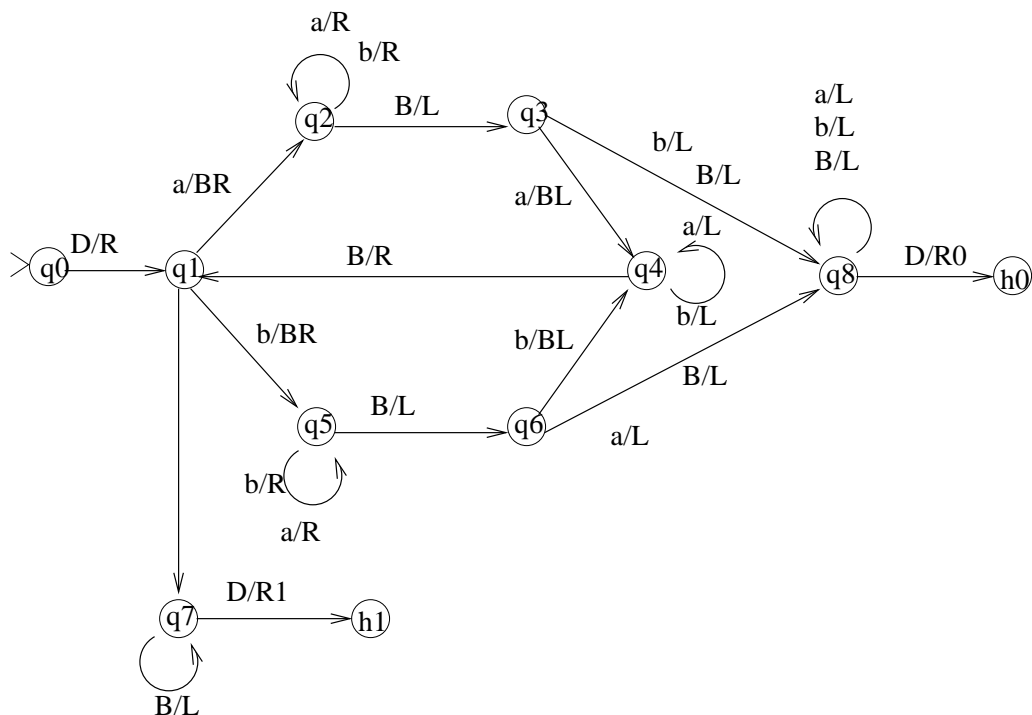
Turing machine for recognizing even-length palindromes

D: marks left end of tape

B: blank

L: move left

R: move right



The machine starts out on the left-end marker, D , in state q_0 . It works by traveling back and forth, iteratively blanking out the first and last character if they match, reducing the string from both ends. If it discovers a mismatch, it goes to state q_8 , searches for the beginning-of-tape marker, moves right one position, writes a 0, and halts in state h_0 . If the string is empty or it blanks out the string without discovering a mismatch, it moves to q_7 , searches for the beginning of tape marker, moves right one position, and writes a 1.

5. A *configuration* consists of the string currently on the tape, the current state, and the current position. We can represent this by underlining the current position and labeling it with the current state. A transition moves from one configuration to the next. As in the case of pushdown automata, we can represent a computation by the sequence of configurations that occur each time we take a transition.

An example of a sequence of configurations giving a computation by the first Turing machine above is given on page 126 of our text.

6. Any deterministic Turing machine, operating on a given input, can be simulated on our RAM model of a computer, which has infinite random-access memory and a CPU equipped with an instruction set consisting of basic arithmetic operations,

including boolean ones, random-access retrieval and storage, and the ability to execute a program consisting of these instructions, including conditional branching. You could prove this by writing a computer program in Java that inputs a specification of a Turing machine and a tape input and simulates its computation. The existence of a compiler that converts your program to machine code that essentially gives the RAM-model proves this. **It also implies that Turing machines are no more powerful than Java programs.** This is a compelling argument, but a complete proof would have to include details of your program and the compiler, so we will resort to this bit of handwaving.

According to the book, a modern computer equipped with random access memory and arithmetic operations, including boolean operations, is **no more** powerful than a Turing machine. The strategy of the proof is one that you are familiar with: they give an algorithm that converts any program, written in typical machine code, into a Turing machine that performs the same computation. As in the argument above, a complete proof is lengthy, and the book's proof is lengthy, yet uses some similar handwaving. I will not ask about this proof on the exam.

The main point we will take away from this is that we can prove things about what can be computed by deterministic Turing machines by proving things about what can be computed by our RAM model or by a Java program. This will save us a lot of grief, since it is even more tedious to make up Turing machines for interesting problems than to write significant programs in machine code. We will use this in the next posting, a guide to reading the handout on NP-completeness, to sidestep Turing machines in discussing complicated algorithms.

7. A *nondeterministic Turing machine* is one that can allow a choice of moves in a given configuration. How would you modify Definition 4.1.1 so that it allowed nondeterminism?