Nondeterministic Turing Machines

Foundations of Computer Science

February 21, 2016

Another variant of Turing machines that comes in handy are *nondeterministic Turing machines*, that bear a similar relationship to deterministic Turing machines that NFAs bear to DFAs. A nondeterministic Turing machine can have more than one transition available at a state for a given symbol at the tape head.

We can define nondeterministic Turing machines easily enough. A nondeterministic Turing machine is a tuple $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \Delta, s, acc, rej)$ defined exactly as a deterministic Turing machine, except that the transition relation Δ has the form

$$\Delta : Q \times \Gamma \longrightarrow \wp(Q \times \Gamma \times \{L, R\}).$$

Intuitively, $\Delta(p, a)$ returns a *set* of possible transitions, each of the form (q, b, d) for a state q, a symbol b to write on the tape at the tape head, and direction in which to move the tape. Again, just like NFAs: we don't just have a single transition, we have multiple possible transitions. As usual, we assume that the tape has a leftmost marker in its leftmost cell, and that the machine cannot move the tape head to the left when on the leftmost marker.

We define how nondeterministic Turing machines compute using configurations. We use the same configurations as for deterministic Turing machines, that is, tuples (u, q, v) where u, v are strings over Γ, and q is a state in Q.

The starting configuration of the machine with input w is $(, s, \vdash\ w)$, where s is the start state of the machine; a configuration is accepting (resp., rejecting) if the state is the accept (resp., reject) state of the machine.

The step relation $C \xrightarrow[M]{1} C'$ between configurations for nondeterministic Turing machines is where the main difference with deterministic Turing machines lies. The step relation is such that a configuration may lead to many next configurations, capturing the nondeterminism of the machine.

$$(u, p, av) \xrightarrow[M]{1} (ub, q, v) \qquad \text{if } (q, b, R) \in \Delta(p, a)$$

$$(uc, p, av) \xrightarrow[M]{1} (u, q, cbv) \qquad \text{if } (q, b, L) \in \Delta(p, a)$$

We define $C \xrightarrow[M]{*} C'$ if $C = C'$ or there exists $C_1, \ldots, C_k$ such that $C \xrightarrow[M]{1} C_1 \xrightarrow[M]{1} C_2 \xrightarrow[M]{1} \ldots \xrightarrow[M]{1} C_k \xrightarrow[M]{1} C'$.

We say nondeterministic Turing machine M *accepts* string w if $(\epsilon, s, \vdash\ w) \xrightarrow[M]{*} C_{acc}$ for some accepting configuration C_{acc}. We say nondeterministic Turing machine M *rejects* a string w if whenever $(\epsilon, s, \vdash\ w) \xrightarrow[M]{*} C$ and C is a halting configuration, then C is a rejecting configuration. A nondeterministic Turing machine is total if it always halts on all inputs (i.e., there is no infinite path in the configuration tree, for any input string). The language accepted by nondeterministic Turing machine M is the set of strings accepted by M.

We say a language is *enumerable by a nondeterministic Turing machine* if there is a nondeterministic Turing machine that accepts it. It is *decidable by a nondeterministic Turing machine* if there is a total nondeterministic Turing machine that accepts it.

Nondeterministic Turing machines also do not give us more enumerable or decidable languages.

Theorem: A language is enumerable (resp., decidable) by a nondeterministic Turing machine if and only if it is Turing-enumerable (resp., Turing-decidable).

The reverse direction is easy: deterministic Turing machines are just special cases of nondeterministic Turing machines, in which there is, for every state and every tape symbol, exactly one transition enabled. Thus, if A is enumerable, it is enumerable by a nondeterministic Turing machine. Similarly if A is decidable.

The forward direction is more interesting. Suppose a language A is enumerable by a nondeterministic Turing machine, say M_{nd}. To show it is Turing-enumerable, we need to show that there is a Turing machine M_A that accepts A. In fact, we are going to show that there is a deterministic Turing machine with 3 tapes that accepts A. (Since multitape Turing machines accept exactly the same class of languages as Turing machines, we get our result.) We build this 3-tape Turing machine M_A by essentially doing a breadth-first search through the configuration tree of M_{nd}. The first tape will hold the original input string. The second tape will hold the queue of configurations through which we search. The third tape will hold the current configuration, which we will use to simulate a step of M_{nd}.

Given M_{nd} with states Q and tape alphabet Γ, M_A has a tape alphabet that includes Γ and Q (so that states of M_{nd} are tape symbols of M_A, so we can write down a configuration onto the tape) as well as special symbols `#` and `/` and is defined as follows:

On input w:

1. Rewrite $\vdash a_1 \ldots a_k$ into $\vdash$ `//`s`/`$a_1 \ldots a_k$ on the first tape, where s is the start state of M_{nd} (`/` separates configuration components)
2. Copy the first tape to the second tape
3. If the second tape is empty, reject
4. Move the last string on the second tape to the third tape, up to the `#` symbol, erasing the string and the `#` symbol
5. If the string on the third tape is an accepting configuration, accept
6. If the string on the third tape is a rejecting configuration, go to step 8
7. Simulate a move of M_{nd}:
   a. Scan third tape from left to right, finding the state in the configuration
   b. For every possible nondeterministic transition of M_{nd} in the configuration on the third tape, write the result of taking the transition as a configuration at the beginning of the second tape, shifting the tape to the right to make room, and write a `#` after the configuration
8. Erase the third tape
9. Go to step 3

Observe once again that if the original M_{nd} is total, then M_A is total. Thus, this construction also shows that if A is decidable by a nondeterministic Turing machine, it is decidable.