CS 3100 – Models of Computation – Fall 2010
Notes for Lecture 20 – 11/04/10

The book defines sets using $\{\langle x\rangle \ : \ p(x)\}$, while we define using $\{\langle x\rangle \ | \ p(x)\}$. The same thing.
Change of perspectives wrt languages

- Mostly we talked about languages having a syntactic structure.
- Now we are talking about languages that encode machine types. We are asking whether the encoding carries enough 'meaning' (*e.g.*, codes up a DFA whose language is empty)
- This is indeed a significant 'bump' that you should keep in mind
- The moment we consider the semantic import of a syntactic encoding, we (as human beings) may not be able to answer many questions; "we" (as the supreme computer type, or TM) may not be able to answer many questions also **in general**.
- For particular questions, we can of course answer
- The trick is to serve as a **uniform** algorithm for all such problems. That is an extreme view, and that is where the sharp jump from decidability to undecidability exists.
- The story is the same for P versus NP. The sharp jump seems to exist only if we consider the whole family of problems. Then the worst case asymptotic $O()$ upper bound seems to be exponential (unless P=NP).

Consider these questions:

- Consider $S_1 = \{\langle G, f\rangle \ | \ cond\}$ where *cond* is "G is a CFG encoding and f is an FA encoding." S_1 is recursive.
- Consider $S_{1.4} = \{\langle f\rangle \ | \ cond\}$ where *cond* is "f is an RE for the language $\{0^m1^n \ | \ n \geq 0\}$." Here it is possible to algorithmically decide $S_{1.4}$. How?
- Consider $S_{1.5} = \{\langle G\rangle \ | \ cond\}$ where *cond* is "G is a CFG encoding for the language $\{0^n1^n \ | \ n \geq 0\}$." It is plausible that $S_{1.5}$ is decidable. The way you would proceed is as follows.
  You would state one specific algorithm that checks, given any grammar G, whether it produces strings only of this form. It is not straightforward to prove such properties in general, but proving for the language $\{0^n1^n \ | \ n \geq 0\}$" seems to be within reach.
  In general it is undecidable. This is Greibach's theorem, very similar to Rice's theorem.
- Deciding non-trivial partitioning of CFGs (through PDA/CFG codes) is undecidable.
- Deciding non-trivial partitioning of TMs is undecidable (Rice's theorem for TMs).
- Consider $S_2 = \{\langle G, f\rangle \ | \ cond\}$ where *cond* is "G is a CFG encoding and f is an FA encoding" and $L(G)$ is non-empty. S_2 is recursive.
- Consider $S_3 = \{\langle G, f\rangle \ | \ cond\}$ where *cond* is "G is a CFG encoding and f is an FA encoding" and $L(G) \subseteq L(f)$ is non-empty. S_3 is recursive.
- Suppose we are given that $S_{univ} = \{\langle G\rangle \ | \ L(G) = \Sigma^*\}$ is not decidable.
  We are given $S_{nonuniv} = \{\langle G\rangle \ | \ L(G) \neq \Sigma^*\}$ is RE.
  Then can S_{univ} be RE?
- $S_3 = \{\langle G, f\rangle \ | \ cond\}$ where *cond* is "G is a CFG encoding and f is an FA encoding" and $L(G) \subseteq L(f)$ is non-empty. Show that S_3 is recursive.
- $S_4 = \{\langle G, f\rangle \ | \ cond\}$ where *cond* is "G is a CFG encoding and f is an FA encoding" and $L(G) = L(f)$. Show that S_4 is not RE.
- $S_5 = \{\langle G, f\rangle \ | \ cond\}$ where *cond* is "G is a CFG encoding and f is an FA encoding" and $L(G) \neq L(f)$. Show that S_5 is RE.

TM definitions

- Note that our book requires an 'accept' and a 'reject' state
- Most other books consider getting 'stuck' in a non-accept state as rejection (as does JFLAP)

- These are equivalent notions
- We can always convert a machine that has a collection F of final (accept) states and all other states being non-accept into a machine with a single h_a and h_r

So what is a digital computer?

- Has a stored program
- When fired up, the program acts on the main memory (to simplify things)
- The program itself is like a "FA" except it acts on the (potentially) infinite amount of memory
- A TM is similar. Its "FA" part is exactly like a C program
- Each step of a TM program accomplishes what amounts to setting or resetting a bit (a tiny amount of work)
- A PDA is a program where the "program part" looks similar to a C program, but the memory itself looks "funny". It is more like your HP calculator with a stack (remember those RPN calculators). Its stack is infinite
- An FA (DFA or NFA) is also like a C program except you've pulled out all the memory chips (no external memory). It remembers something "by being in a state"
- A PDA can remember things in a stack manner.
- A TM can remember "a whole lot" and look at info written any time in the past on the tape
- For fun (and completeness), introduce something called an LBA which is a TM that can write only on the portion of the input tape where the input was provided (it may read from outside of this region, but this info never changes, so re-reading things outside of the input zone does not "teach" the LBA anything)
- A TM can write anywhere on the tape and later read back its own writing. It can garner up an arbitrary amount of its own past writings before it decides to do one next thing.

The kinds of problems we want to study (not manually, but using a computer).

- Whether an FA can do something
- Whether a PDA can do something
- Whether an LBA can do something
- Whether a TM can do something

What is "something?" Not weight lifting, obviously. Now keep in mind that a TM is the most powerful type of "computer." So,

- If we have to study what an FA can do using a TM, it looks entirely feasible . Many of these studies can be done by looking at the structure of the FA.
- Thus telling whether an FA's language is empty is quite easy. We can minimize that FA and look and see if there is any path to a final state.
- Telling whether an FA can generate all strings with even positions containing a 1 is harder, but still does not need us to generate all the strings in the FA's language and check. We can algorithmically compare the equivalence of that FA with an RE which encodes "even positions being a 1."
- If we want to study what a PDA can do (using a TM) looks promising too.
- Even the halting problem of an LBA is answerable.
- But studying what a TM can do (by using another TM)? It surely has to run into trouble! Because there is no way we can build a "full state graph" of a TM and study it. We gotta run the TM being queried. So when a big TM runs a little TM, there is no guarantee that ihis run will ever come back.

Show that an infinite expansion of your cat will still ensure only a countably infinite number of hairs on its body, whereas the map of USA has an uncountably infinite number of points.

- An infinitely expanded cat's hide is still a 2D patch of pairs over Nat. This is isomorphic to Nat
- The map of USA has a rectangle inscribed in it, and inscribing it. The Cardinality Trap theorem says that the map has the same cardinality as either of these rectangles, which is uncountably infinite.