Illinois Institute of Technology
Department of Computer Science

Lecture 25: April 24.tex

CS 330 Discrete Structures
Spring Semester, 2019

1 Languages that are not regular

We talked about pumping lemma which can help determine that a language is not regular. A classic example of these languages is $\{0^n1^n|n = 0, 1, 2, \cdots\}$. There is not a finite state machine to recognize this language since you need infinite many states to keep track of "Where am I" during the computation. This requires infinite memory. For languages like the one above, we can recognize them using a finite state machine with a stack (PDA: push down automata)+: the machine just pushes every 0 into the stack until it sees the first 1. From that point it pops a 0 everytime it sees a 1. If it sees a 0 again, the string is rejected. Otherwise, if the stack is empty when the input ends, the machine accepts the input string. If a language can be recognized by a PDA, we call it context free language. (The name "context free" comes from the property of grammars generating this kind of language).

Can PDA recognize every language? No. The language $\{0^n1^n2^n|n = 0, 1, 2, \cdots\}$ can not be recognized by any PDA. With a slightly advanced version of pumping lemma we can prove that this language is not context free.

How about a PDA with two stacks? It can recognize the above language. Actually a PDA with two stacks is equivalent to a Turing machine, which can recognize any language that is decidable and compute any problem that is computable.

Are there any language with is not decidable (that means, given the input string, no machine can tell you whether the input belongs to that language)? Yes. One example is that we can't have a machine which, given input strings encoding a student's program, test input and standard output, can tell you whether the students program generates the correct output, because there might be dead loops in the student's code and the machine never "dares" to cut off the running!

2 Graphs and finite state machines

There are many questions about FSMs and regular languages that can be solved by treating the FSM as a graph. For instance, given an FSM, does it accept anything?[1] Is a language infinite? Are two languages equal?

3 An application: string matching

One useful application of FSMs is in string matching. The basic idea is to construct an FSM that recognizes the string being searched for and to run it with the text as input.

Of course, this brute force approach is inefficient. In particular, as stated, it will only find the string if it occurs at the very beginning of the text. If the FSM rejects, we could then run it on the text beginning at the second character, and then beginning at the third character, and so on, but that would require $O(mn)$

[1] In terms of graphs, then, this question would be whether there are any paths from the start state to any of the accept state.

time, and we can do better. For instance, if the search string is `abababaca` and the text is `ababababacaba`, after reaching the third `b` in the text, rather than announcing failure and giving up, we can make use of the fact that the text, so far, still matches the beginning of the search string, as long as we begin looking two characters into the text.

How can this idea be encoded in an FSM? The idea is simple: *on failure, move to the latest state in the FSM that still matches some prefix of the text.* The algorithm is given in detail in section 32.4 of *Introduction to Algorithms*, 3nd edition, by Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein, MIT Press, 2009., and yields an $O(m+n)$ string matching algorithm. This algorithm and its variants are commonly used in text editors.