

CS385-Theory of Computation

Final Exam

December 18, 2006

1 Floating-point Constants in Java

In the C programming language and its descendants (C++, Java) decimal floating-point constants are strings over the alphabet $\{0, 1, \dots, 9, E, +, -, .\}$ having the following form: An optional $+$ or $-$ at the beginning, followed by a string of digits, then a decimal point, then another string of digits, followed by an optional E . If the E is present, it is followed by an optional $+$ or $-$, then a string of digits.

There are several restrictions: Either of the two strings of digits surrounding the decimal point can be empty, but they cannot both be empty. The decimal point is not optional! (A string of digits without a decimal point is an integer constant, not a floating point constant.) If the E is present, then the string of digits at the end cannot be empty.

For example,

+1.3E28

1.E+28

.3E+28

-2.76

-.76

are all legitimate decimal floating point constants.

(a) Draw the state-transition diagram of an NFA that recognizes this set of strings. Since all the digits $0, \dots, 9$ induce the same state transitions, you may use a single letter d to denote “digit”. (In truth, there is hardly any need for nondeterminism here, and an equivalent DFA is almost as simple. However, it helps to be able to use an occasional ϵ -transition and to not have to write all the transitions to a dead state.)

(b) Write a regular expression for this set of strings. You could do this by applying the algorithm for transforming automata to regular expressions to the automaton you constructed in part (a), but it is surely easier to write down the regular expression directly.

2 An Operation Preserving Regularity

Let Σ be a finite alphabet, and let $\#$ be a symbol that is not an element of Σ . If $u \in (\Sigma \cup \{\#\})^*$, then we denote by $k(u)$ the string in Σ^* that results by deleting all occurrences of $\#$ from u .

Let $L \subseteq \Sigma^*$, and set

$$L^\# = \{u \in (\Sigma \cup \{\#\})^* : k(u) \in L\}.$$

In other words, $L^\#$ is obtained from L by inserting zero or more occurrences of the symbol $\#$ in all the words of L . For instance, if L consists of the single word ab , then $L^\#$ is the infinite language $\#^*a\#^*b\#^*$.

(a) Show that if L is a regular language, then so is $L^\#$. You can do this by showing either how to alter an automaton for L or a regular expression for L so as to obtain one for $L^\#$. Both alterations are extremely simple to describe!

(b) Let $L^{1-\#}$ denote the set of all words in $L^\#$ that contain exactly one occurrence of $\#$. For example, if L consists of the single word ab , then $L^{1-\#}$ is the language

$$\{\#ab, a\#b, ab\#\}.$$

Show that if L is a regular language, then so is $L^{1-\#}$. You can use either of the proof strategies suggested in (a), but it is easier to use the RESULT of (a) and apply closure properties of regular languages.

(c) The following is a “proof” that the language

$$T = \{a^n\#a^n : n \geq 0\}$$

is regular. If we apply k to all the words in T , erasing the single occurrence of $\#$, we get the regular language L of words over $\{a\}$ having even length. Thus $T = L^{1-\#}$, and hence is regular, by the result of (b).

Of course, T is not regular, so there is an error in this argument. What is wrong?

3 A Grammar for Functional Expressions

The terminal alphabet for this context-free grammar G consists of function symbols f, g, h , variable symbols x, y, z , the comma and parentheses. The grammar below generates nested functional expressions like

$$f(x, y, g(x)).$$

$$S \longrightarrow f(T)|g(T)|h(T)$$

$$T \longrightarrow T, T|S|x|y|z$$

(a) Show a derivation tree for $f(x, y, g(x))$.

- (b) Show that this G is ambiguous.
- (c) Find an equivalent unambiguous grammar.
- (d) Show that $L(G)$ is not regular.

4 Effective Computability of Binary Addition

Let ADD denote the language

$$\{u\#v\#w : u, v, w \in \{0, 1\}^*, u_2 + v_2 = w_2\}.$$

Here, u_2 denotes the *integer* whose binary representation is u . Thus this language consists of all ‘correct’ binary addition equations, like

$$1011\#001111\#11010,$$

(eleven, fifteen, twenty-six.)

- (a) Give an *implementation description* of a one-tape Turing machine that decides this language. (To remind you of the style and level of detail present in such descriptions, I have attached one from the textbook.)
- (b) What is the maximum number of steps that the Turing machine you describe in (a) makes on an input of length n ? I’m not looking for an exact answer, but rather an asymptotic answer, like $O(n^3)$ or $2^{O(n)}$.
- (c) Can the asymptotic running time be speeded up by using a two-tape Turing machine? Explain briefly.

5 Decidability and Recognizability

Consider the problem, given two Turing machines M_1 and M_2 , if there is any word w such that both M_1 and M_2 accept w . In language terms, this is the language

$$K = \{ \langle M_1, M_2 \rangle : L(M_1) \cap L(M_2) \neq \emptyset \}.$$

- (a) Show that K is undecidable by reducing a problem we already know to be undecidable to K . (There is an easy reduction from A_{TM} , but there’s an even easier one from another problem.)
- (b) Show that K is recognizable. (It can help to use the fact that for every TM M , the language $L(M)$ is effectively enumerable.)
- (c) Suppose we consider instead the language

$$K' = \{ \langle M_1, M_2 \rangle : L(M_1) \cap L(M_2) = \emptyset \},$$

Is K' recognizable. (Assume the results stated in (a) and (b), even if you didn’t do those parts.)