University of Nevada, Las Vegas Computer Science 456/656 Spring 2022

Answers to Assignment 6: Due Tuesday May 3 2022 23:59:59

Throughout this assignment, you may assume that a language is recursively enumerable if and only if it is accepted by some machine. Recall that "L is recursively enumerable (RE)" means that there is a machine that enumerates L.

1. In the course of getting your degree in computer science, you will learn many things; but some of those things are significantly more important than others. Because of the introduction of massively parallel machines, knolwedge about the relationship between $\mathcal{P}$–TIME and $\mathcal{NC}$ is among the most important. Read the document `dpNC02` on dynamic programming and $\mathcal{NC}$ with that in mind.

   (a) Explain how addition of binary numerals for integers is a Boolean dynamic program with reachback two.[1]

   In reality, the problem has reachback 1.

   Let x, y be binary strings of length n. Our goal is to compute a binary string $x + y$. Let s_i be the i^{th} bit is $x + y$, and let c_i be the carry bit generated while computing s_i.

   Let $T_i \in \{0, 1, 2\}$ be the integer sum of x_i and y_i. T_i is not a variable of the dynamic program, since it is an input.

   The $2n$ variables of the DP are $s_1, s_2, c_2, \ldots c_n$, in that order. Each of those can be computed in terms of its immediate predecessor:

$$s_i = \begin{cases} !c_{i-1} \text{ if } T_i = 1 \\ c_{i-1} \text{ otherwise} \end{cases}$$

$$c_i = \begin{cases} 0 \text{ if } T_i = 0 \\ !s_i \text{ if } T_i = 1 \\ 1 \text{ if } T_i = 2 \end{cases}$$

   (b) Describe an $\mathcal{NC}$ algorithm to solve any Boolean dynamic program with reachback 3.

   This is a special case of Theorem 1 in the document, which is quite important. Despite the fact that the result is "obvious," it is not at all obvious why it is obvious. (Smiley face.) I will explain it in formal terms mixed with informal remarks to help you along.

   Here is the formal definition of the dynamic program. There are n Boolean valued subproblems, $S_1, \ldots S_n$ whose values are $x_1 \ldots x_n$. Because reachback is bounded by 3, we can write $x_1 = F_1$ (with no parameters), $x_2 = F_2(x_1)$, $x_3 = F_3(x_2, x_1)$, and $x_i = F_i(x_{i-1}, x_{i-2}, x_{i-3})$ for $i \geq 4$. It turns out that, for any $i > j \geq 3$, $x_i = F_i^{j,j-1,j-2}(x_j, x_{j-1}, x_{j-2})$, where $F_i^{j,j-1,j-2}$ is a Boolean function with three Boolean parameters. (Note that $F_i = F_i^{i-1,i-2,i-3}$.) We will compute all those Boolean functions in $O(\log n)$ time, using $O(n)$ processors. Since we can't spend linear time, we need to compute $F_i^{j,j-1,j-2}$ without knowing the values of the parameters! How can we do this? We first note that there are only three Boolean parameters for each function, and there are only $2^3 = 8$ choices of those parameters. Thus each $F_i^{j,j-`,j-2}$, which we also denote F_i^j for short, can be stored in $O(1)$ space. For each $i > j$, we will compute and store a Boolean valued array $F_i^j[2][2][2]$: in other words, $F_i^j(p, q, r)$ for $p, q, r \in \{0, 1\}$.

[1] Your computer contains an $\mathcal{NC}$ microprogram which finds the sum of any two binary numerals of length n.

Our method is the typical "bootstrap" method of dynamic programming. We start with the fact that $F_i^{i-1,i-2,i-3} = F_i$, hence the formula for F_i^{i-1}, for $i \geq 4$, is an input of the problem, which we store as a vector of 8 Booleans, one for each choice of (p, q, r).

Define the "gap" of F_i^j to be $i - j$. We already have the formulae for the functions of gap = 1. We proceed to work out the formulae for functions with increasing gaps.

It turns out that for any given g, we can compute all F_i^j for $i - j = g$ simultaneously with n processors, in $O(1)$ time, provided we have answers for all smaller gaps. But this will not yield an $\mathcal{NC}$ computation. There are $\Theta(n)$ possible values of g, and if we work them sequentially, the time will be $O(n)$, which is not polylogarithmic.

Saving the Day! For any $i > j \geq 3$, let $k = (i + j)/2$, where we use integer (truncated) division. Then

$$\begin{aligned} x_i &= F_i^j(x_j, x_{j-1}, x_{j-2}) \\ &= F_j^k(x_k, x_{k-1}, x_{k-2})) \\ &= F_j^k(F_k^j(x_j, x_{j-1}, x_{j-2}), F_{k-1}^j(x_j, x_{j-1}, x_{j-2}), F_{k-2}^j(x_j, x_{j-1}, x_{j-2}) \end{aligned}$$

we can thus compute F_i^j which has gap $g = i - j$ by using four functions whose gaps are approximately at most $\frac{g+1}{2}$, in a constant time computation:

$$F_i^j(p, q, r) = F_j^k(F_k^j(p, q, r), F_{j-1}^k(p, q, r), F_{j-2}^k(p, q, r))$$

for each choice of p, q, r.

Batching the Gaps. We first compute x_1, x_2, x_3, and record F_i^{i-1} for all $i \geq 4$. The remaining computation is broken into $O(\log n)$ phases. During the first phase, we compute all functions of gap 2. During the second phase, we compute all functions of gaps 3 and 4. During the t^{th} phase, we compute all functions with gaps between 2^{t-1} and 2^t, namely F_i^j for $2^{t-1} < i - j \leq 2^i$. When we are done, we have $x_i = F_i^3(x_3, x_2, x_1)$. Each phase takes constant time with $O(n^2)$ processors. Thus, the computation is $\mathcal{NC}$.

Different Infinities If S is any set the set of all subsets of S is written 2^S. Georg Cantor (1845–1918) was the first to note that infinite sets may be of different sizes (cardinalities). We say that two sets A and B have the same *cardinality* if there is a 1-1 correspondents $f : A \to B$. That is, for every $b \in B$, there is exactly one $a \in A$ such that $f(a) = b$. If S is any set, the cardinality S is written as $|S|$, and the cardinality of 2^S is written $2^{|S|}$.

The finite cardinals are called zero, one, two, *etc.* But some sets are infinite, such as $\mathcal{N}$, the set of natural numbers. We use the Hebrew letter $\aleph$ to denote infinite cardinals. By definition, $\aleph_0$ is the smallest infinite cardinal, $\aleph_1$ is the next smallest, and so forth.

The cardinality of the set of natural numbers $\mathcal{N}$ is $\aleph_0$. Thus the cardinality of every enumerable (countable) set is also $\aleph_0$. The set of all integers and the set of all fractions are both countable. Every language is countable, and every subset of a countable set is countable; that subset is either finite or has cardinality $\aleph_0$.

Cantor proved, using diagonalization, that the set of real numbers $\mathbb{R}$ is uncountable. (Sets which are not countable are called uncountable.) Here is his proof, which is by contradiction.

Suppose $\mathbb{R}$ is countable. Then there is a 1-1 correspondence $f : \mathcal{N} \to \mathbb{R}$. Each real number has a decimal expansion. Let x be a real number between 0 and 1 such that the ith decimal digit of x (that is, in the 10^{-i} place) is one greater than the i^{th} decimal digit of $f(i)$ unless that digit is 9, in which case we use 8. Then $x \neq f(i)$ for all i, and thus the image of f does not contain x, contradiction.

Another example of an uncountable set is the set of all languages over a given alphabet Σ. That set is the set of all subsets of Σ^*, and is written 2^{Σ^*}. The cardinality of that set is $2^{\aleph_0}$, which is the same as the cardinality of $\mathbb{R}$.

The continuum hypothesis is that the cardinality of $\mathbb{R}$ is $\aleph_1$, that is, $2^{\aleph_0} = \aleph_1$ Cantor died without being able to prove it. In 1964, Paul Cohen proved that the continuum hypothesis cannot be proved using the standard axioms of set theory. It was already known that it could not be disproved, either. (The room was packed when I saw Cohen's presentation.)

2. Which of these sets are countable? Give a justification in each case.

   (a) The set of all recursively enumerable languages.

   (b) The set of all undecidable languages.

   (c) The set of all real numbers whose decimal expansions are computed by a machine.