Chapter 7

Deterministic Polynomial Time

7.1. Motivating Completeness for $\mathcal{P}$

Having dealt with the first containment in $\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP}$, we move on to $\mathcal{P}$ and a study of $\leq_m^{\mathcal{L}}$–completeness for $\mathcal{P}$. One motivation, of course, is that any problem $\leq_m^{\mathcal{L}}$–complete for $\mathcal{P}$ will be in $\mathcal{L}$, or $\mathcal{NL}$, or $\mathcal{SC}$, if and only if all problems in $\mathcal{P}$ are. The following complexity class encompasses all three of these classes:

Definition 7.1: $\mathcal{POLYLOG} = \bigcup_{c>0} \text{DSPACE}(\log^c n)$.

Note that $\mathcal{POLYLOG} = \bigcup_{c>0} \text{NSPACE}(\log^c n)$ due to Corollary 4.15, and that $\mathcal{POLYLOG} = \bigcup_{c>0} \text{ATIME}(\log^c n)$ due to Theorems 4.11 and 4.14. It is also clear that $\mathcal{SC} \subseteq \mathcal{POLYLOG}$, since $\mathcal{SC}$ is a time-bounded version of $\mathcal{POLYLOG}$.

As an example of a problem in $\mathcal{POLYLOG}$, group isomorphism (*i.e.*, given the multiplication tables of two finite groups G and H, is $G \cong H$?) is in $\text{DSPACE}(\log^2 n)$ [29], but not known to be in $\mathcal{NL}$ or in $\mathcal{SC}$, or even in $\mathcal{P}$.

Open Problem 7.2: Is $\mathcal{P} \subseteq \mathcal{POLYLOG}$? The conjecture is "no", and some evidence is given by Cook and Sethi [6].

Since $\mathcal{POLYLOG}$ is closed under $\leq_m^{\mathcal{L}}$ (left as an exercise to the reader), any problem that is $\leq_m^{\mathcal{L}}$–complete for $\mathcal{P}$ is in $\mathcal{POLYLOG}$ if and only if $\mathcal{P} \subseteq \mathcal{POLYLOG}$, by Proposition 5.6.

Another motivation is that problems that are $\leq_m^{\mathcal{L}}$–complete for $\mathcal{P}$ are "inherently sequential", assuming $\mathcal{P} \not\subseteq \mathcal{POLYLOG}$, meaning that they cannot be very efficiently parallelized. We will return to this topic in Section 7.5.

7.2. Boolean Circuits

The immediate reason to introduce Boolean circuits is that they are the subject of the first problem that will be shown $\leq_m^{\mathcal{L}}$–complete for $\mathcal{P}$. In addition, we will have occasion to study circuits as computational devices in Section 7.5.

Definition 7.3: A *(Boolean* or *combinational) circuit* is an acyclic, oriented, directed graph with two distinguished subsets of vertices, called *inputs* and *outputs*. ("Oriented" means that the edges directed into any particular vertex are ordered.) The vertices have indegrees and labels as follows:

- *Inputs* have indegree 0 and are labeled consecutively from the set of indeterminates $\{x_1, x_2, \ldots, x_n\}$.
- All noninput vertices are called *gates*. A gate with indegree d is labeled by any function $f : \{0,1\}^d \to \{0,1\}$.

In what follows, unless specified otherwise it is assumed that the maximum indegree d is a constant with respect to the number n of input labels. Such circuits are said to have *bounded fanin*.

Definition 7.4: The *value* $\text{val}(u) \in \{0,1\}$ of vertex u in a circuit on input $(b_1, b_2, \ldots, b_n) \in \{0,1\}^n$ is defined as follows:

- If u is an input with label x_i then $\text{val}(u) = b_i$.
- If u is a gate with label $f : \{0,1\}^d \to \{0,1\}$, and the d vertices with edges directed to u are $u_1, u_2, \ldots, u_d$, then $\text{val}(u) = f(\text{val}(u_1), \text{val}(u_2), \ldots, \text{val}(u_d))$.

A circuit with 1 output vertex u is said to *output* b on input $(b_1, b_2, \ldots, b_n)$ if the value of u on input $(b_1, b_2, \ldots, b_n)$ is equal to b.

We usually restrict the labels f to be from {AND, OR, NOT}.

Definition 7.5: If the labels f must be in {AND, OR}, then the circuit is called *monotone*.

Note that arbitrary functions cannot be computed by monotone circuits. For example, the function NOT cannot be so computed. Only "monotone functions" can be computed by monotone circuits, where a monotone function has the property that, if any input is changed from 0 to 1, the output cannot change from 1 to 0.

The next definitions introduce natural complexity measures for circuits.

Definition 7.6: The *size* of a circuit C is the number of gates in C.

Definition 7.7: If u is a vertex of a circuit C, then $\text{depth}(u)$ is the length of any longest path from any input to u. The *depth of* C is the maximum, over all vertices u in C, of $\text{depth}(u)$.

If a circuit is viewed as a sequential computing device, then size is a measure of sequential time, because each gate represents one operation. On the other hand, if a circuit is viewed as a parallel computing device, then depth is the appropriate measure of time, since all gates at a given depth can be evaluated in parallel. In the latter case, size is a measure of the number of parallel processors required.

7.3. One-Read Alternating Turing Machines

Definition 7.8: A *one-read alternating Turing machine* is an alternating Turing machine that, on each path of its computation tree, writes only one string of the form ia on its index tape, where $i \in \{0,1\}^*$ and $a \in \Sigma$. It then enters a special "read" state q_{read} and halts in the next step, accepting if and only if $x_i = a$, where x_i is the ith input symbol. Its computation on that path, and in particular whether it accepts, depends on no other input symbol.

Lemma 7.9: An alternating Turing machine M that runs in $T(n) \geq \log_2 n$ time and $S(n) \geq \log_2 n$ space can be simulated by a one-read alternating Turing machine N that runs in $O(T(n))$ time and $O(S(n))$ space.

Proof:

CONSTRUCTION: On input x, N simulates M, but records M's index tape on a separate worktape. If M has i written on its index tape, N does the following to simulate one more step of M:

existentially choose $a \in \Sigma$;
universally choose $b \in \{0,1\}$;
case b **of**
  0: **if** $x_i = a$ **then accept else reject;**
  1: continue simulating M as though M received the response $x_i = a$
end .

N does the test "$x_i = a$?" by writing ia on its index tape and entering q_{read}.

Example 7.10: Suppose $\Sigma = \{a, b\}$, and on some computation path in M we have two steps that read input characters x_i and x_j, respectively. Then the corresponding portion of the computation tree for N is shown in Figure 7.1.

CORRECTNESS: The only new universal configurations that can possibly be accepting are those for which the correct value x_i was guessed, and those are accepting if and only if M's continuation is accepting.

ANALYSIS: Along any path in the original computation tree for M, each step is replaced by 3 steps of N; in addition, at the end of the path there is an added $O(\log n)$ time to copy i to the index tape. Thus, the total time is $3T(n) + \log_2 n = O(T(n))$, since $T(n) \geq \log_2 n$. The only additional space that N uses is the $\log_2 n$ space needed to record the contents of M's index tape on a separate worktape. Since $S(n) \geq \log_2 n$, this is $O(S(n))$. Note, though, that this method may greatly increase the number of alternations between existential and universal configurations. □

7.4. The Circuit Value Problem

Let

$$MCV = \{(C, b_1, b_2, \ldots, b_k) \mid C \text{ is a monotone circuit with } k \text{ inputs that outputs 1 on input } (b_1, b_2, \ldots, b_k)\ \} .$$

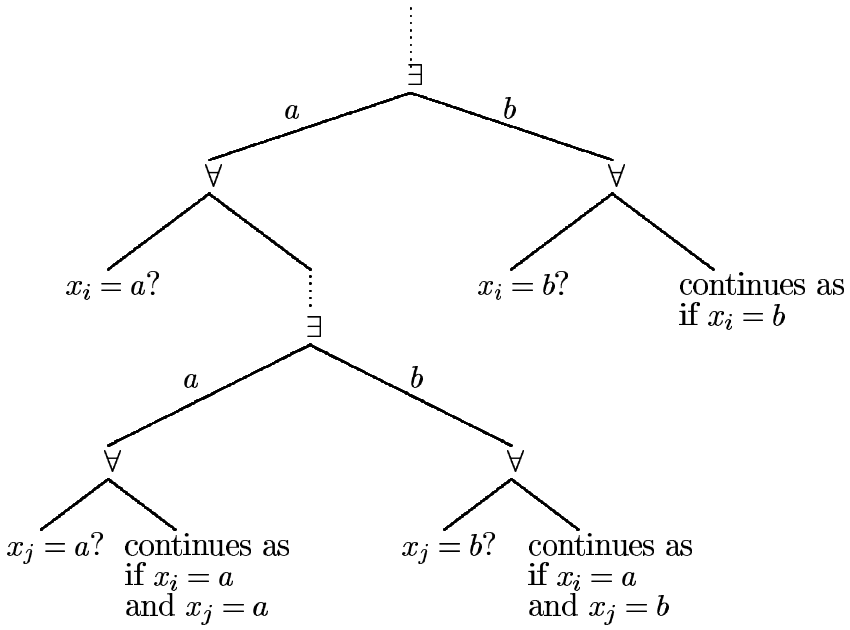

Figure 7.1: Computation Tree for One-Read Alternating Turing Machines

Theorem 7.11 (Ladner [27], Goldschlager [11]): MCV is $\leq_m^{\mathcal{L}}$–complete for $\mathcal{P}$.

Proof:

1. $MCV \in \mathcal{P}$: By Theorem 4.7 it suffices to show that MCV is accepted by an alternating Turing machine A that runs in space $O(\log n)$.

CONSTRUCTION: Starting at C's output gate, A does the following. Let u be the vertex currently being evaluated by A.

- Case 1: If u is an input of C labeled x_i, then A accepts if and only if $b_i = 1$.
- Case 2: If u is an OR (AND) gate with input vertices v_0 and v_1, then A existentially (respectively, universally) chooses $b \in \{0,1\}$, replaces u with v_b, and continues by evaluating v_b.

CORRECTNESS: By induction on depth(u), A is in an accepting configuration when evaluating u if and only if val(u) = 1.

Basis (depth(u) = 0): Then u is an input. By Case 1 above, A accepts if and only if val(u) = $b_i = 1$.

Induction (depth(u) > 0): Assume the induction hypothesis holds for all vertices v such that depth(v) $\leq k$. Let u be a vertex in C such that depth(u) = $k+1$, and $u = v_0$ OR v_1 ($u = v_0$ AND v_1). By the induction hypothesis, A is in an accepting configuration when evaluating v_0 if and only if val(v_0) = 1, and A is in an accepting configuration when evaluating v_1 if and only if val(v_1) = 1. By the construction, A is in an accepting configuration when evaluating u if and only if A is in an accepting configuration when evaluating v_0 or (respectively, and) when evaluating v_1,

which occurs if and only if $\text{val}(v_0) = 1$ or (respectively, and) $\text{val}(v_1) = 1$, which in turn occurs if and only if $\text{val}(u) = 1$.

ANALYSIS: Since $(C, b_1, b_2, \ldots, b_k)$ is already in A's input, A need only store u, v_b, and i, all three of which use $O(\log n)$ space.

2. *MCV* is $\leq_m^{\mathcal{L}}$-hard for $\mathcal{P}$: Let L be an arbitrary language in $\mathcal{P}$. By Theorem 4.9, we can assume that L is accepted by an alternating Turing machine A in space $\log_2 n$. It suffices to produce a deterministic Turing machine D running in $O(\log n)$ space that reduces L to *MCV*. Without loss of generality, we will make three simplifying assumptions about A:

1. By Lemma 7.9, we can assume that A is a one-read alternating Turing machine.
2. Assume that every configuration of A has either zero or two immediate successors.
3. Assume that A uses a separate worktape to count how many steps it has taken. The purpose of this is to prevent A from repeating any configuration along any path of its computation tree. If this counter exceeds the maximum running time of A (which is easily computable since its space bound is $\log_2 n$), then A halts and rejects.

CONSTRUCTION: On input x, D outputs $(C, b_1, b_2, \ldots, b_k)$, where C contains a vertex u_P for every configuration P of A, and has labels and edges as follows:

- If P is an existential (universal) configuration with $P \vdash_{A,x} Q$ and $P \vdash_{A,x} R$, then u_P is an OR (respectively, AND) gate with inputs u_Q and u_R.
- If P is a configuration in state q_{read} with ia on its index tape, then u_P is an input vertex with value
$$\begin{cases} 1 & \text{if } x_i = a \\ 0 & \text{if } x_i \neq a \end{cases}.$$
D can determine whether $x_i = a$, since i is written on its worktape as part of P, and x is on its input tape.
- If P is a final (nonfinal) halting configuration, then u_P is an input vertex with value 1 (respectively, 0).
- If P is the initial configuration, then u_P is the output gate.

Note that C is acyclic, since A is constructed so that it never repeats any configuration along a single computation path. Note also why we need A to be one-read.

CORRECTNESS: By induction on $\text{depth}(u_P)$, P is an accepting configuration if and only if $\text{val}(u_P) = 1$.

Basis ($\text{depth}(u_P) = 0$): Then u_P is an input vertex. By construction, $\text{val}(u_P) = 1$ if and only if P is either a final configuration, or a configuration containing q_{read} that is about to accept.

Induction ($\text{depth}(u_P) > 0$): Suppose that P is an existential (universal) configuration with $P \vdash_{A,x} Q$ and $P \vdash_{A,x} R$. Then P is accepting if and only if either Q or R is accepting (respectively,

both Q and R are accepting) which, by the induction hypothesis, is true if and only if $\text{val}(u_Q) = 1$ or (respectively, and) $\text{val}(u_R) = 1$, which is true if and only if $\text{val}(u_P) = 1$.

ANALYSIS: D uses $O(\log n)$ space to enumerate each of A's configurations P, to find the configurations Q such that $P \vdash_{A,x} Q$ in order to output the edge (u_Q, u_P), and to count to i in order to find x_i on the input tape. This is all possible since A's configurations only have length $O(\log n)$, and $i \leq n$. □

Let $CV = \{(C, b_1, b_2, \ldots, b_k) \mid C$ is a circuit with k inputs and with gates labeled from {AND, OR, NOT} that outputs 1 on input $(b_1, b_2, \ldots, b_k)\}$.

Corollary 7.12 (Ladner [27]): CV is $\leq_m^{\mathcal{L}}$–complete for $\mathcal{P}$.

Notice that the fact that monotone circuits are provably weaker than general circuits does not contradict the fact that $CV \leq_m^{\mathcal{L}} MCV$: the latter says that it is as hard to predict the output of a monotone circuit as it is to predict the output of a general circuit.

7.5. Relating Circuits to $\mathcal{P}$

In this section we present a slightly different view of the relationship between circuits and $\mathcal{P}$. Rather than viewing circuits as the subject of a problem to be solved by a polynomial time deterministic Turing machine, circuits will be viewed as computing devices themselves, and we will be interested in how their power relates to the power of Turing machines. We begin by discussing circuits as computing devices with associated complexity measures.

Definition 7.13: Let C be a circuit with n inputs and 1 output. The *language accepted by* C is $L(C) = \{x \in \{0,1\}^n \mid C$ outputs 1 on input $x\}$.

Note that $L(C) \subseteq \{0,1\}^n$ is always finite. Therefore, in defining circuit complexity classes below, we must consider an infinite family of circuits, one for each input size n.

Definition 7.14: $\text{SIZE}(S(n)) = \{L \mid$ for all n, $L \cap \{0,1\}^n$ is accepted by some circuit C_n with n inputs, bounded fanin, and size $S(n)\}$.

Definition 7.15: $\text{DEPTH}(D(n)) = \{L \mid$ for all n, $L \cap \{0,1\}^n$ is accepted by some circuit C_n with n inputs, bounded fanin, and depth $D(n)\}$.

Definition 7.16: $\mathcal{PSIZE} = \bigcup_{c>0} \text{SIZE}(n^c)$.

If $n = 1$, size=1 suffices.

Theorem 7.17: $\mathcal{P} \subseteq \mathcal{PSIZE}$.

Proof: It suffices to simulate an alternating Turing machine that runs in $O(\log n)$ space by a family of polynomial size circuits C_n, by Theorem 4.9. This is what was done in proving MCV is $\leq_m^{\mathcal{L}}$–hard for $\mathcal{P}$, in Theorem 7.11, except:

- if P is a configuration in state q_{read} with ia on its index tape, then u_P is
  - an input vertex labeled x_i, if $a = 1$,
  - the negation $\neg x_i$ of an input vertex, if $a = 0$, and
  - the constant 0, if $a \notin \{0, 1\}$.

□

The state of the art in refining Theorem 7.17 is given in the following theorem:

Theorem 7.18 (Pippenger and Fischer [35]): $\text{DTIME}(T(n)) \subseteq \text{SIZE}(O(T(n) \log T(n)))$.

Open Problem 7.19: Improve Theorem 7.18.

Ruzzo [39] showed a tighter relationship between alternating Turing machines and circuits than Theorem 7.17. He first observed that the same simulation demonstrates that an alternating Turing machine running in time $T(n)$ and space $S(n)$ (simultaneously) can be simulated by a family of circuits C_n of depth $O(T(n))$ and size $2^{O(S(n))}$ simultaneously. Because of the nonuniformity in the definition of SIZE and DEPTH (i.e., C_n has, in general, no resemblance to C_{n+1}), it is impossible for the converse to hold, that is, for a single alternating Turing machine to simulate an infinite family of circuits efficiently. The following exercise makes this point as conclusively as one would like:

Exercise 7.20: Prove that SIZE(1) contains nonrecursive sets. (Hint: define a nonrecursive language that, for all nonnegative integers n, either contains every string or no string in $\{0, 1\}^n$.)

To overcome this obstacle, Ruzzo defined "uniform" families of circuits, and showed that alternating time $O(T(n))$ and space $O(S(n))$ is *equivalent* to uniform circuit depth $O(T(n))$ and size $2^{O(S(n))}$ simultaneously. In this sense, alternating Turing machines are a good model of parallel algorithms, with time measuring parallel time and space measuring the logarithm of the number of processors.

Definition 7.21: For any $k \geq 1$, $\mathcal{NC}^k = \{L \mid L \text{ is accepted by some alternating Turing machine in time } O(\log^k n) \text{ and space } O(\log n) \text{ simultaneously}\}$.

Definition 7.22: $\mathcal{NC} = \bigcup_{k \geq 1} \mathcal{NC}^k$.

By Ruzzo's Theorem, $\mathcal{NC}$ is the class of languages that can be accepted by extremely fast parallel algorithms using a polynomial amount of hardware.

Here is how $\mathcal{NC}$ fits in with previously studied complexity classes:

$$\mathcal{NC}^1 \subseteq \mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{NC}^2 \subseteq \mathcal{NC} \subseteq \mathcal{P} \cap \mathcal{POLYLOG}.$$

The first containment follows from Theorem 4.11, the third from the proof of Theorem 4.14 (where only $O(\log n)$ space is needed to retain three configurations), $\mathcal{NC} \subseteq \mathcal{P}$ from Theorem 4.7, and $\mathcal{NC} \subseteq \mathcal{POLYLOG}$ again from Theorem 4.11.

If any language that is $\leq_m^{\mathcal{L}}$–hard for $\mathcal{P}$ is in $\mathcal{NC}$, then $\mathcal{P} = \mathcal{NC}$. Thus, assuming $\mathcal{P} \neq \mathcal{NC}$, these hard problems are "inherently sequential". This provides another strong motivation for the study of completeness for $\mathcal{P}$.

7.6. Other Problems Complete for $\mathcal{P}$

To indicate the variety of natural problems that are known to be $\leq_m^{\mathcal{L}}$–complete for $\mathcal{P}$, seven of them are described below. The source for these and other problems is a compendium by Greenlaw, Hoover, and Ruzzo [13], which describes approximately 100 such problems. The proof of the first of these seven will be given in Section 7.7, but the remaining proofs are omitted.

1. Linear Programming [7, 24]

   INPUTS: $n \times d$ integer matrix A, vector b of n integers.

   PROBLEM: Is there a vector x of d rational numbers such that $Ax \leq b$?

   The ordinary linear programming problem, which asks to maximize the inner product $c \cdot x$ subject to $Ax \leq b$, is transformed most naturally into a language recognition problem by adding one more constraint $c \cdot x \geq k$, where c and k are also part of the input. This extra constraint can then be incorporated as a new row of A and b.

2. Maximum Flow [12]

   INPUTS: Directed graph G with a nonnegative integer capacity c_e for each edge e, and two designated vertices s and t.

   PROBLEM: Is the maximum "flow" from s to t odd?

   The *flow* f_e on any edge e is a nonnegative number that satisfies the following conditions:

   - $f_e \leq c_e$ for each edge e, and
   - the total flow into vertex v equals the total flow out of vertex v, for all $v \notin \{s, t\}$.

   The *flow from s to t* is the total net flow into t. The question of whether the maximum flow from s to t is at least a designated integer k is also $\leq_m^{\mathcal{L}}$–complete for $\mathcal{P}$ [28].

3. First Fit Decreasing Bin-Packing [1]

   INPUTS: Rational numbers $v_1, v_2, \ldots, v_n \in [0, 1]$, integers i and b.

   PROBLEM: Is v_i put into bin b by the First Fit Decreasing heuristic?

   First Fit Decreasing is the following heuristic: consider each v_i in decreasing order, inserting it into the least numbered bin in which it still fits.

4. Depth First Search [37]

   INPUTS: Adjacency lists for a graph G, and three distinguished vertices s, u, and v.

   PROBLEM: Is u visited before v by the Depth First Search algorithm, assuming it starts at vertex s and handles adjacencies in the order dictated by the adjacency lists?

5. Unit Resolution [20]

   INPUTS: A Boolean formula F in conjunctive normal form.

   PROBLEM: Can the empty clause $\Box$ be deduced from F by unit resolution?

   As an example of unit resolution, if two of the clauses in F are $A = \neg y$ and $B = x \vee y \vee \neg z$, then unit resolution adds the clause $x \vee \neg z$ to F.

Deducing the empty clause is a way of proving unsatisfiability of F in resolution. If instead one wanted to prove that F implies some clause E, then $\neg E$ could be added to F and proved unsatisfiable.

6. Unification [8]

   INPUTS: Two terms s and t, each composed of variables and function symbols.

   PROBLEM: Are there substitutions of terms for the variables in s and t that cause them to become equal?

   As an example, the terms $s = f(x, g(x, y))$ and $t = f(g(w, w), z)$ can be unified by setting $x = g(w, w)$ and $z = g(g(w, w), y)$.

7. Context-Free Grammar Emptiness

   Citation purposely omitted: to be used as homework

   INPUTS: Context-free grammar G.

   PROBLEM: Is $L(G) = \emptyset$?

7.7. Linear Programming

Theorem 7.23 (Dobkin, Lipton, and Reiss [7], Khachian [24]): Linear Programming is $\leq_m^{\mathcal{L}}$–complete for $\mathcal{P}$.

Proof:

1. Linear Programming $\in \mathcal{P}$: This is difficult and was open for many years. It was finally proved by Khachian [24].

2. $MCV \leq_m^{\mathcal{L}}$ Linear Programming: This proof is due to Cook [13].

CONSTRUCTION: Given a monotone circuit C and its k input values $b_1, b_2, \ldots, b_k$, we must output A and b such that $Ax \leq b$ has a solution x if and only if C outputs 1 on inputs $b_1, b_2, \ldots, b_k$.

For each vertex u of C, there is a corresponding variable x_u and some constraints as follows:

- For each input vertex u with value b, output the inequalities representing
  - $x_u = b$ (actually represented by two inequalities $x_u \leq b$ and $-x_u \leq -b$).
- For each OR gate u with inputs v and w, output the inequalities
  - $0 \leq x_u \leq 1$,
  - $x_v \leq x_u$ (i.e., if val$(v) = 1$, then val$(u) = 1$),
  - $x_w \leq x_u$ (i.e., if val$(w) = 1$, then val$(u) = 1$), and
  - $x_u - x_v - x_w \leq 0$ (i.e., if val$(v) = 0$ and val$(w) = 0$, then val$(u) = 0$).
- For each AND gate u with inputs v and w, output the inequalities
  - $0 \leq x_u \leq 1$,
  - $x_u \leq x_v$ (i.e., if val$(v) = 0$, then val$(u) = 0$),

- ▷ $x_u \leq x_w$ (i.e., if $\text{val}(w) = 0$, then $\text{val}(u) = 0$), and
- ▷ $x_v + x_w - x_u \leq 1$ (i.e., if $\text{val}(v) = 1$ and $\text{val}(w) = 1$, then $\text{val}(u) = 1$).

- If u is the output of C, output the inequalities representing
  - ▷ $x_u = 1$.

CORRECTNESS: By induction on depth(u), the only possible solution for x_u is val(u). Hence, $Ax \leq b$ has a solution if and only if C outputs 1.

ANALYSIS: Space $O(\log n)$ suffices to hold u, v, w, and the label of a single input vertex. □

7.8. Problems in $\mathcal{P}$ Not Known to be Complete

Open Problem 7.24: Each of the following problems is known to be in $\mathcal{P}$, but not known to be either complete for $\mathcal{P}$ or to be in $\mathcal{POLYLOG}$. (Of course, they need not necessarily be either.)

1. Greatest Common Divisor

   INPUTS: n-bit integers x and y.

   PROBLEM: Compute $\gcd(x, y)$.

   This problem is known to be in $\mathcal{P}$ by Euclid's algorithm. The problem is still open even if we just want to determine if x and y are relatively prime, i.e., $\gcd(x, y) = 1$.

2. Modular Exponentiation

   INPUTS: n-bit integers a, e, and m.

   PROBLEM: Compute $a^e \bmod m$.

   This may not seem entirely natural, but there are other important problems that are reducible to this. These include probabilistic primality checking, RSA encryption and decryption [38], computing inverses modulo primes, and computing square roots modulo certain primes.

3. Edge Weighted Matching

   INPUTS: Graph G with positive integer edge weights.

   PROBLEM: Find a matching of maximum weight.

4. Stable Marriage

   INPUTS: n men and n women each with a complete ordered list of marital preferences.

   PROBLEM: Find n marriages such that there do not exist a man and a woman who each prefer the other over his or her own spouse.

5. Comparator Circuit Value

   INPUTS: Circuit C of comparators (each of which takes two inputs x and y and outputs exactly one copy each of $\max(x, y)$ and $\min(x, y)$), vector $(b_1, b_2, \ldots, b_k) \in \{0, 1\}^k$, and integer i.

   PROBLEM: Is the ith output of C on input $(b_1, b_2, \ldots, b_k)$ a one?

It is interesting to note that problems 3, 4, and 5 are all equivalent (i.e., reducible to each other) and that there are no known reductions between problems 1 and 2.

7.9. Exercises

1. Show that $CV \in \mathcal{P}$.
2. Do Exercise 7.20.
3. Prove that $\mathcal{NC}$ is closed under $\leq_m^{\mathcal{L}}$.
4. Let *CFGempty* be the set of context-free grammars G such that $L(G) = \emptyset$. Prove that *CFGempty* is $\leq_m^{\mathcal{L}}$–complete for $\mathcal{P}$.