CHAPTER 17

# THE POLYNOMIAL HIERARCHY

*Although the complexity classes we shall study now are in one sense byproducts of our definition of NP, they have a remarkable life of their own.*

## 17.1 OPTIMIZATION PROBLEMS

Optimization problems have not been classified in a satisfactory way within the theory of **P** and **NP**; it is these problems that motivate the immediate extensions of this theory beyond **NP**.

Let us take the traveling salesman problem as our working example. In the problem TSP we are given the distance matrix of a set of cities; we want to find the shortest tour of the cities. We have studied the complexity of the TSP within the framework of **P** and **NP** only indirectly: We defined the decision version TSP (D), and proved it **NP**-complete (corollary to Theorem 9.7). For the purpose of understanding better the complexity of the traveling salesman problem, we now introduce two more variants.

EXACT TSP: Given a distance matrix and an integer $B$, is the length of the shortest tour *equal* to $B$? Also,

TSP COST: Given a distance matrix, compute the length of the shortest tour.

The four variants can be ordered in "increasing complexity" as follows:

TSP (D); EXACT TSP; TSP COST; TSP.

Each problem in this progression can be reduced to the next. For the last three problems this is trivial; for the first two one has to notice that the reduction in

the corollary to Theorem 9.7 proving that TSP (D) is **NP**-complete can be used to reduce HAMILTON PATH to EXACT TSP (the graph has a Hamilton path if and only if the optimum tour has length *exactly* $n+1$). And since HAMILTON PATH is **NP**-complete and TSP (D) is in **NP**, we must conclude that there is a reduction from TSP (D) to EXACT TSP.

Actually, *we know that these four problems are polynomially equivalent* (since the first and the last one are, recall Example 10.4). That is, there is a polynomial-time algorithm for one if and only if there is for all four. Admittedly, from the point of view of the practical motivation for complexity theory (namely, to identify problems that are likely to require exponential time) this coarse characterization should be good enough. However, reductions and completeness provide far more refined and interesting categorizations of problems. In this sense, of these four variants of the TSP we know the precise complexity only of the **NP**-complete problem TSP (D). *In this section we shall show that the other three versions of the* TSP *are complete for some very natural extensions of* **NP**.

## The Class DP

Is the EXACT TSP in **NP**? Given a distance matrix and the alleged optimum cost $B$, how can we certify succinctly that the optimum cost is indeed $B$? The reader is invited to ponder about this question; no obvious solution comes to mind. It would be equally impressive if we could certify that the optimum cost *is not* $B$; in other words, EXACT TSP does not even appear to be in **coNP**. In fact, the results in this section will suggest that if EXACT TSP is in $\mathbf{NP} \cup \mathbf{coNP}$, this would have truly remarkable consequences; the world of complexity would have to be vastly different than it is currently believed.

However, EXACT TSP is closely related to **NP** and **coNP** in at least one important way: Considered as a language, *it is the intersection of a language in* **NP** (the TSP language) *and one in* **coNP** (the language TSP COMPLEMENT, asking whether the optimum cost is *at least* $B$). In other words, an input is a "yes" instance of EXACT TSP if and only if it is a "yes" instance of TSP, and a "yes" instance of TSP COMPLEMENT. This calls for a definition:

**Definition 17.1:** A language $L$ is in the class **DP** if and only if there are two languages $L_1 \in \mathbf{NP}$ and $L_2 \in \mathbf{coNP}$ such that $L = L_1 \cap L_2$. □

We should warn the reader immediately against a quite common misconception: **DP** *is not* $\mathbf{NP} \cap \mathbf{coNP}$[†]. There is a world of difference between these two classes. For one thing, **DP** is not likely to be contained even in $\mathbf{NP} \cup \mathbf{coNP}$, let alone the much more restrictive $\mathbf{NP} \cap \mathbf{coNP}$. The intersection in the definition of $\mathbf{NP} \cap \mathbf{coNP}$ is in the domain of *classes of languages*, not languages as

[†] We mean, these two classes are not *known* or *believed* to be equal. In the absence of a proof that $\mathbf{P} \neq \mathbf{NP}$ one should not be too emphatic about such distinctions.



with **DP**.

For another important difference between $\mathbf{NP} \cap \mathbf{coNP}$ and **DP**, the latter is a perfectly syntactic class, and therefore has *complete problems*. Consider for example the following problem:

SAT-UNSAT: Given two Boolean expressions $\phi, \phi'$, both in conjunctive normal form with three literals per clause. Is it true that $\phi$ is satisfiable and $\phi'$ is not?

**Theorem 17.1:** SAT-UNSAT is **DP**-complete.

***Proof:*** To show that it is in **DP** we have to exhibit two languages $L_1 \in \mathbf{NP}$ and $L_2 \in \mathbf{coNP}$ such that the set of all "yes" instances of SAT-UNSAT is $L_1 \cap L_2$. This is easy: $L_1 = \{(\phi, \phi') : \phi \text{ is satisfiable}\}$ and $L_2 = \{(\phi, \phi') : \phi' \text{ is unsatisfiable}\}$.

To show completeness, let $L$ be any language in **DP**. We have to show that $L$ reduces to SAT-UNSAT. All we know about $L$ is that there are two languages $L_1 \in \mathbf{NP}$ and $L_2 \in \mathbf{coNP}$ such that $L = L_1 \cap L_2$. Since SAT is **NP**-complete, we know that there is a reduction $R_1$ from $L_1$ to SAT, and a reduction $R_2$ from the complement of $L_2$ to SAT. The reduction from $L$ to SAT-UNSAT is this, for any input $x$:

$$R(x) = (R_1(x), R_2(x)).$$

We have that $R(x)$ is a "yes" instance of SAT-UNSAT if and only if $R_1(x)$ is satisfiable and $R_2(x)$ is not, which is true if and only if $x \in L_1$ and $x \in L_2$, or equivalently $x \in L$. □

As usual, starting from our basic "satisfiability-oriented" complete problem we can show many more **DP**-completeness results:

**Theorem 17.2:** EXACT TSP is **DP**-complete.

***Proof:*** We already argued that it is in **DP**. To prove completeness, we shall reduce SAT-UNSAT to it. So, let $(\phi, \phi')$ be an instance of SAT-UNSAT. We shall use the reduction from 3SAT to HAMILTON PATH (recall the proof of Theorem 9.7) to produce from $(\phi, \phi')$ two graphs $(G, G')$, each of which has a Hamilton path if and only if the corresponding expression is satisfiable. But our construction will be novel in this way: Whether or not the expressions are satisfiable, the graphs $G$ and $G'$ will always contain a *broken Hamilton path*, that is, two node-disjoint paths that cover all nodes.

To this end, we modify slightly each expression so that it has an *almost satisfying* truth assignment, that is, a truth assignment that satisfies all clauses except for one. This is easy to do: We add a new literal, call it $z$, to all clauses, and add the clause $(\neg z)$. This way, by setting all variables to **true** we satisfy all clauses except for the new one. We then turn the expression into one with three literals per clause by replacing the clause $(x_1 \vee x_2 \vee x_3 \vee z)$, say, by the two clauses $(x_1 \vee x_2 \vee w)$ and $(\neg w \vee x_3 \vee z)$.

If we now perform the reduction in Theorem 9.7 starting from a set of clauses that has such an almost satisfying truth assignment, call it $T$, it is easy

to see that the resulting graph always has a broken Hamilton path: It starts at node 1, it traverses all variables according to $T$, and continues to the clauses *except for the one that may be unsatisfied*, where the path is broken once (you may want to examine the "constraint gadget" in Figure 9.6 to verify that it causes at most one such break). The path then continues normally up to node 2.

We shall use this fact to show that SAT-UNSAT can be reduced to EXACT TSP. Given an instance $(\phi, \phi')$ of SAT-UNSAT, we apply to both $\phi$ and $\phi'$ the reduction to HAMILTON PATH, to obtain two graphs, $G$ and $G'$, respectively, both guaranteed to have broken Hamilton paths. We next combine the two graphs *in a cycle* by identifying node 2 of $G$ with node 1 of $G'$, and vice-versa (Figure 17.1). Let $n$ be the number of nodes in the new graph.

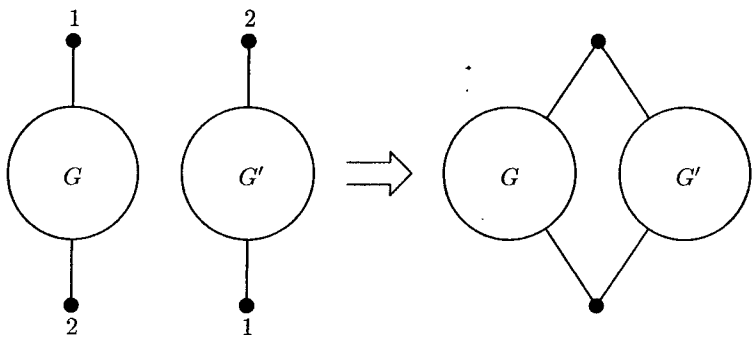



**Figure 17-1.** Combining $G$ and $G'$.

We next define distances between the nodes of the combined graph to obtain an instance of the TSP. The distance between nodes $i$ and $j$ are defined as follows: If $[i, j]$ is an edge of either graph $G$ or graph $G'$, then the distance is one. If $[i, j]$ is not an edge, but both $i$ and $j$ are nodes of graph $G$ then its distance is two; all other non-edges have distance 4.

What is the length of the shortest tour of this instance of the traveling salesman problem? Obviously, this depends on whether $\phi$ and $\phi'$ are satisfiable or not. If they are both satisfiable, then the optimum cost is $n$, the number of nodes in the combined graph (there is a Hamilton cycle in the combined graph). If they are both unsatisfiable, then the optimum cost is $n + 3$ (the optimum tour combines the two broken Hamilton paths, and thus both a non-edge of $G$ and a non-edge of $G'$ will have to be used). If $\phi$ is satisfiable and $\phi'$ is not, then the optimum cost is $n+2$ (a non-edge of $G'$ will have to be used, but not of $G$). And if $\phi$ is unsatisfiable and $\phi'$ is satisfiable, then the optimum cost is $n + 1$.



*It follows that $(\phi, \phi')$ is a "yes" instance of* SAT-UNSAT *if and only if the optimum cost is $n + 2$.* Taking $B$ to be equal to this number completes our reduction from SAT-UNSAT to EXACT TSP. □

The "exact cost" versions of all **NP**-complete optimization problems that we have seen (INDEPENDENT SET, KNAPSACK, MAX-CUT, MAX SAT, to name a few) can be shown **DP**-complete, each by a different trick that combines two instances, and forces the optimum cost to precisely reflect the status of the two expressions. So, **DP** appears to be the natural niche of the "exact cost" aspect of optimization problems.

But **DP** is much richer than this. For example, besides SAT-UNSAT, there are two more satisfiability-related problems in **DP**:

CRITICAL SAT: Given a Boolean expression $\phi$, is it true that $\phi$ is unsatisfiable, but deleting any clause makes it satisfiable?

UNIQUE SAT: Given a Boolean formula $\phi$, is it true that it has a unique satisfying truth assignment?

CRITICAL SAT exemplifies an important and novel genre of problems, those asking whether the input is *critical* with respect to a given property, that is, it has the property but its slightest unfavorable perturbation does not. Other examples:

CRITICAL HAMILTON PATH: Given a graph, is it true that it has no Hamilton path, but addition of any edge creates a Hamilton path?

CRITICAL 3-COLORABILITY: Given a graph, is it true that it is not 3-colorable, but deletion of any node makes it 3-colorable?

All three "critical" problems are known to be **DP**-complete. On the other hand UNIQUE SAT, and many other problems asking whether a given instance has a unique solution, are simply not known to be in any weaker class. They are not known (or believed) to be **DP**-complete (see the references). Incidentally, UNIQUE SAT should not be confused with the class **UP** of unambiguous nondeterministic computations (recall Section 12.2). The two address very different aspects of unique solutions in decision problems: UNIQUE SAT is about *determining* whether the solution exists and is unique; **UP** concerns the computational power of instances that are guaranteed either to have a unique solution or no solution. The satisfiability problem for **UP**, call it UNAMBIGUOUS SAT, would be the following: Given a Boolean expression that is known to have at most one satisfying truth assignment, does it have one? This is a completely different problem than UNIQUE SAT.

### The Classes $\mathbf{P^{NP}}$ and $\mathbf{FP^{NP}}$

One can look at **DP** as the class of all languages that can be decided by an oracle machine (recall Section 14.3) of a very special nature: The machine makes two

queries to a SAT oracle, and then accepts if and only if the first answer was "yes" and the second was "no." Obviously, one can generalize this to situations where the acceptance pattern is *any fixed Boolean expression* (in the case of **DP**, for example, the expression is $x_1 \wedge \neg x_2$, see the references).

But the more interesting generalization is to allow *any polynomial number of queries*, and in fact queries computed *adaptively*, based on the answers of previous queries. This way we arrive at the class $\mathbf{P}^{\mathrm{SAT}}$, the class of all languages decided by polynomial-time oracle machines with a SAT oracle. Since SAT is **NP**-complete, instead of it we could use as an oracle any language in **NP**—this is why we can equivalently write $\mathbf{P}^{\mathrm{SAT}}$ as $\mathbf{P}^{\mathbf{NP}}$. Yet another name for this class is $\Delta_2\mathrm{P}$; this name identifies $\mathbf{P}^{\mathbf{NP}}$ as one of the first levels of an important progression of classes, discussed in the next section.

Having defined $\mathbf{P}^{\mathbf{NP}}$, we can now define its corresponding class of *functions* $\mathbf{FP}^{\mathbf{NP}}$ (recall **FP** and **FNP** in Chapter 10). That is, $\mathbf{FP}^{\mathbf{NP}}$ is the class of all functions from strings to strings that can be computed by a polynomial-time Turing machine with a SAT oracle. In fact, we shall be much more interested in $\mathbf{FP}^{\mathbf{NP}}$ than in $\mathbf{P}^{\mathbf{NP}}$, because the former class happens to have many natural complete problems, *including many important optimization problems*. For example, $\mathbf{FP}^{\mathbf{NP}}$ finally provides the sought precise characterization of the complexity of the TSP.

There are several natural $\mathbf{FP}^{\mathbf{NP}}$-complete problems. The version of satisfiability appropriate for this level is the following:

MAX-WEIGHT SAT: Given a set of clauses, each with an integer weight, find the truth assignment that satisfies a set of clauses with the most total weight.

But our reductions this time will start with a problem that is even closer to computation than satisfiability:

MAX OUTPUT: We are given a nondeterministic Turing machine $N$ and its input $1^n$. $N$ is such that, on input $1^n$, and for any sequence of nondeterministic choices, it halts after $\mathcal{O}(n)$ steps with a binary string of length $n$ on its output string. We are asked to determine *the largest output*, considered as a binary integer, of any computation of $N$ on $1^n$.

**Theorem 17.3:** MAX OUTPUT is $\mathbf{FP}^{\mathbf{NP}}$-complete.

***Proof:*** Let us first point out that MAX OUTPUT, along with any optimization problem whose decision version is in **NP**, is in $\mathbf{FP}^{\mathbf{NP}}$. The algorithm is essentially the one used for the TSP (Example 10.4): Given $N$ and $1^n$, we repeatedly ask whether there is a sequence of nondeterministic choices leading to an output larger than an integer $x$. We repeat this for various integers $x$, converging to the value of the optimum by binary search. Each such question can be answered in **NP**, and hence the resulting algorithm establishes that MAX OUTPUT is in $\mathbf{FP}^{\mathbf{NP}}$. (Incidentally, notice that the binary search algorithm is *adaptive*, in that it makes nontrivial use of the answers to previous queries in order to



construct the next query; in some sense, the result being proved suggests that *binary seach is the most general way of doing this.*)

Suppose then that $F$ is a function from strings to strings in $\mathbf{FP}^{\mathbf{NP}}$. That is, there is a polynomial-time oracle machine $M^?$ such that for all inputs $x$ $M^{\mathrm{SAT}}(x) = F(x)$. We shall describe a reduction from $F$ to MAX OUTPUT. Since this is a reduction between function problems, what is required is two functions $R$ and $S$ such that (a) $R$ and $S$ are computable in logarithmic space; (b) for any string $x$ $R(x)$ is an instance of MAX OUTPUT; and (c) $S$ applied to the maximum output of $R(x)$ returns $F(x)$, the value of the function on the original input $x$.

Given $x$, we shall first describe the $R$ part of the reduction, that is, how to construct machine $N$ and its input $1^n$. To start, define $n = p^2(|x|)$, where $p(\cdot)$ is the polynomial bound of $M^{\mathrm{SAT}}$—this will give $N$ plenty of time to simulate $M^{\mathrm{SAT}}$. We describe $N$ informally, like any other nondeterministic Turing machine; it will be clear that its transition relation can be constructed in logarithmic space, starting from $x$. $N$ on input $1^n$ first generates $x$ on a string (this is the only place in the construction where $x$ is needed), and then it simulates $M$ on input $x$. The simulation is very easy and deterministic, *except for the query steps of* $M^{\mathrm{SAT}}$.

Suppose that $M^{\mathrm{SAT}}$ arrives at its first query step, asking whether some Boolean expression $\phi_1$ is satisfiable. $N$ simulates this by nondeterministically guessing the answer $z_1$ to this query—$z_1$ is 1 if $\phi$ is satisfiable, 0 otherwise. If $z_1 = 0$, then $N$ simply continues its simulation of $M^{\mathrm{SAT}}$, naturally from state $q_{\mathrm{NO}}$. But if $z_1 = 1$, then $N$ goes on to guess a satisfying truth assignment $T_1$ for $\phi_1$, and check that indeed $T_1$ satisfies $\phi_1$. If the test succeeds, then $N$ goes on to simulate $M^{\mathrm{SAT}}$ from state $q_{\mathrm{YES}}$. But if the test fails, then $N$ writes the smallest possible output, $0^n$, and halts; we call this an *unsuccessful computation.*

$N$ continues this way to simulate $M^{\mathrm{SAT}}$ on input $x$, using its nondeterminism to guess the answers $z_i, i = 1, \ldots$ of all queries. When $M^{\mathrm{SAT}}$ would halt, $N$ outputs *the bit string* $z_1 z_2 \ldots$ *of the alleged answers to the queries*, followed by enough zeros to bring the total length of the output up to $n$, followed by the output of $M^{\mathrm{SAT}}$ (needed for the $S$ part). This is a *successful computation.*

Many of the successful computations of $N$ will be *erroneous simulations of* $M^{\mathrm{SAT}}$, in the sense that maybe a query $\phi_j$ was satisfiable, and still $z_j = 0$—every successful computation will be correct about $z_j = 1$. But we claim that *the successful computation that outputs the largest integer does correspond to a correct simulation.* The reason is simple: Suppose that in the successful computation which leads to the largest output, we have $z_j = 0$ for some $j$, while $\phi_j$ was satisfiable —say by truth assignment $T_j$. Take the smallest such $j$ (that is, the earliest such mistake). But then there is another successful computation of $N$, which is identical to the present one up to the $j$th query step, at which point it guesses $z_j = 1$, and then goes on to correctly guess

the truth assignment $T_j$, check it, and continue successfully to the end. But the output of this other computation agrees with the present one in the first $j-1$ bits, and has a 1 in its $j$th position. Hence it represents a larger number, contradicting the maximality of the present computation. It follows that the computation of $N$ with the largest output does indeed correspond to a correct simulation of $M$.

To summarize the structure of $N$, it has $|x|$ states for writing $x$ on its string, and uses its $p^2(|x|)$-long input as an alarm clock. The rest of its transition relation reflects the transition function of $M^?$, with the exception of the query state, which is simulated by a simple nondeterministic routine. It should be clear that $N$ can be constructed in logarithmic space. As for the $S$ part of the reduction, $F(x)$ can be simply read off the end of the largest output of $N$. □

**Theorem 17.4:** MAX-WEIGHT SAT is $\mathbf{FP}^{\mathbf{NP}}$-complete.

***Proof:*** The problem is in $\mathbf{FP}^{\mathbf{NP}}$: By binary search, using a SAT oracle, we can find the largest possible total weight of satisfied clauses, and then, by setting the variables one-by-one, the truth assignment that achieves it.

We must now reduce MAX OUTPUT to MAX-WEIGHT SAT. As in the reduction in Cook's theorem (Theorem 8.2), starting from the nondeterministic machine $N$ and its input $1^n$ we can construct a Boolean expression $\phi(N,n)$ such that any satisfying truth assignment of $\phi(N,n)$ corresponds to a legal computation of $N$ on input $1^n$. All clauses in $\phi(N,n)$ are given a huge weight, say $2^n$, so that any truth assignment that aspires to be optimum must satisfy all clauses of $\phi(N,n)$.

We now add some more clauses to $\phi(N,n)$. Recall that in $\phi(N,n)$ there are variables corresponding to the symbol contained in every position of every string of $N$, at every step. So, there are $n$ variables, call them $y_1,\ldots,y_n$, corresponding to the bits of the output string at halting. *We add to our instance of* MAX-WEIGHT SAT *the one-literal clauses* $(y_i) : i = 1,\ldots,n$, *where clause* $(y_i)$ *has weight* $2^{n-i}$. It is easy to see that, because of these new clauses, and their weights that are the right powers of two, the optimum truth assignment must now not represent just any legal computation of $N$ on input $1^n$, but it must represent the computation that produces the output with the largest possible binary integer value. Finally, for the $S$ part of the reduction, from the optimum truth assignment of the resulting expression (in fact, even from the optimum weight alone!) we can easily recover the optimum output of $N$. □

We can now proceed to the main result of this section:

**Theorem 17.5:** TSP is $\mathbf{FP}^{\mathbf{NP}}$-complete.

***Proof:*** We know that TSP is in $\mathbf{FP}^{\mathbf{NP}}$ (Example 10.4) To prove completeness, we shall reduce MAX-WEIGHT SAT to it. Given any set of clauses $C_1,\ldots,C_m$ on $n$ variables $x_1,\ldots,x_n$, with weights $w_1,\ldots,w_m$, we shall construct an instance



of the TSP such that the optimum truth assignment of the set of clauses can be easily recovered from the optimum tour.

The TSP instance will be given as usual in terms of a graph. All distances that do not correspond to edges in the graph are prohibitively large, say $W = \sum_{i=1}^{m} w_i$. The graph is a variant of that used in the **NP**-completeness proof of the Hamilton path problem (see Figure 17.2, and compare with the proof of Theorem 9.7). There are "choice" gadgets for the variables connected *in tandem* as before, but the "constraint" gadgets for the clauses are now different: Each constraint gadget consists of *four parallel edges*, three corresponding to the literals of the clause (so that the tour will traverse one of the **true** literals in the clause), plus an extra parallel edge functioning as an "emergency exit:" If the clause is unsatisfied and has no **true** literals, then the three parallel edges will not be available, and the emergency exit must be taken. *All edges of the graph have length 0 except for the emergency exits for the clauses, whose length is the weight of the corresponding clause.* This way, each time an emergency edge is taken, the lost weight of the corresponding clause is accurately represented in the cost of the tour.

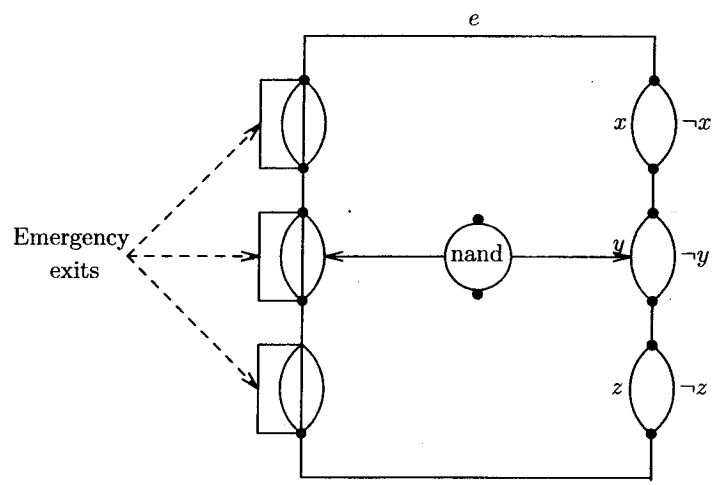



**Figure 17-2.** The overall construction.

What remains is perhaps the more subtle part of the construction, the "consistency" gadget. Because of the new constraint gadget based on three parallel edges (which is in some sense the "dual" of the triangle we used in Figure 9.6), we must connect literal occurrences with the *opposite literal* in the

choice gadget, not with the same literal as before. More importantly, we must allow for an edge corresponding to an occurrence of a **true** literal *not* to be traversed (in the case that there are two or three **true** literals in the clause). As a result, the "exclusive or" gadget in Figure 9.5 is not appropriate. We must design a "nand" gadget, allowing for the possibility that neither edge is traversed. Such a gadget would connect each literal edge of each clause with the opposite literal in the corresponding choice gadget, ensuring that, once a choice is made, the opposite literals cannot be traversed by the Hamilton cycle.

Our nand gadget is rather complicated (it has 36 nodes!), but the idea in designing it is quite simple: After all, a nand gadget is nothing else but an exclusive-or gadget, which also has the additional option of being "turned off," left untraversed. We can achieve this effect by using the "diamond gadget" shown in Figure 17.3. This graph has the following interesting property, easily checked with a little experimentation: Suppose that it is part of a graph so that, as usual, only the black nodes have edges going to the rest of the graph. Then, it can be traversed by a Hamilton cycle only in one of the two ways shown in the figure: Either "from North to South," or "from East to West." In other words, if a Hamilton cycle enters the graph from any one of the four black nodes, it will have to traverse the whole graph and exit from the opposite node.

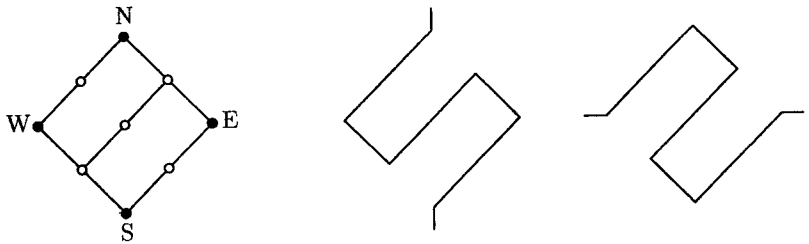



**Figure 17-3.** The diamond.

Our nand gadget is nothing else but our exclusive-or gadget of Figure 9.5, *only with its four vertical paths of length two replaced each by the diamond gadget* as shown in Figure 17.4(a). It is easy to see that, with this replacement, the overall graph functions exactly as before, as an exclusive-or between its upper and lower edge. The point is that now *we can use the East-West endpoints to turn off the device at will*, by taking the horizontal path shown in the figure. We shall represent the nand gadget as an exclusive-or gadget with an extra path which, if traversed, can leave the rest of the device untraversed, and thus "turned off" (Figure 17.4(b)).



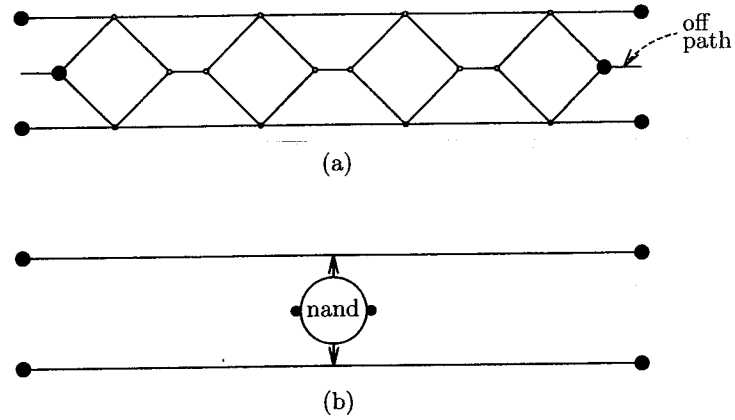



**Figure 17-4.** The nand gadget.

We now have to modify slightly the "constraint" gadgets in a way that allows just one of the possibly two or three parallel edges corresponding to **true** literals in a clause to be traversed. Recall that each nand gadget corresponds to an occurrence of a literal in a clause. Order arbitrarily the three literals of the clause as *first*, *second*, and *third*. The parallel path corresponding to the first literal now starts by another choice (see Figure 17.5), where the choice is between (1) turning off the nand gate of the second literal (if that literal also happens to be **true**), and (2) not turning it off. Then the path continues with another choice, that of turning off the third literal, in case it is also **true**, or not turning it off. The path corresponding to the second literal has only one choice, between turning off the third literal or not. The third literal has no such choices. In other words, we have given the three literals *priorities:* If the first literal is **true**, then it is traversed and must turn off any of the other literals that may also be **true**. Failing this, if the second literal is **true**, then it must be traversed and possibly turn off the third literal, if **true**. Finally, if only the third literal is **true**, then it must be traversed. And if no literal is $T$, then the emergency exit must be taken.

The construction is now complete. To review it (see Figure 17.2), we start with a choice for each variable, then four parallel paths for each clause, with extra choices for each of the first two paths to turn off the exclusive-or of the subsequent paths, and finally the cycle is closed. Each literal occurrence edge is connected with the opposite literal in the choice gadget of the corresponding variable by a nand gadget. The emergency edge corresponding to clause $C_i$ has length $w_i$, all other edges have length zero, while the length of all non-edges is

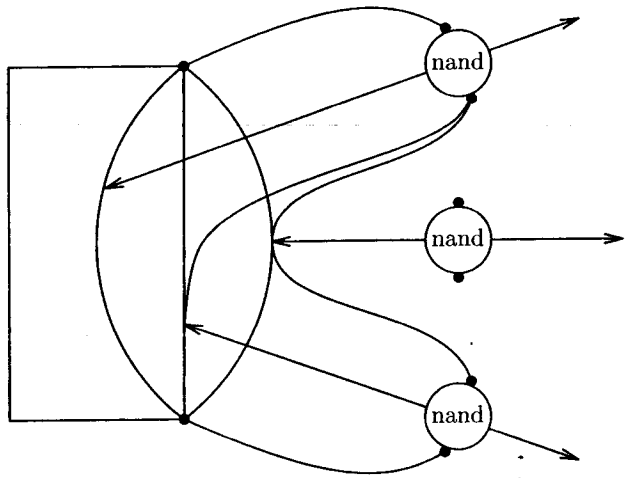



**Figure 17-5.** The clause gadget.

prohibitively large, say $W$, the sum of all weights.

Consider now the optimum traveling salesman tour of this instance. Obviously, no non-edges are traversed, and thus the tour is in fact a Hamilton cycle of the graph (and so our gadgets come into play). The tour must traverse the choices for the variables, thus defining a truth assignment, call it $T$. It then traverses the choices for the nand gates, turning some of them on and some of them off. It finally traverses the clause part. For each clause it must be the case that the tour traverses exactly one of the four parallel edges. This edge can be either a literal that is **true** in $T$, or the emergency edge. All nand gadgets corresponding to traversed literal occurrences must have been turned "on," while all corresponding to **false** or untraversed **true** literal occurrences must be "off." Finally, the tour is closed, at a total cost equal to the *sum of the weights of the clauses not satisfied by* $T$, that is, $W$ minus the total weight of $T$. It follows that *the minimum-length tour corresponds to the maximum-weight truth assignment*, and the proof is complete. □

**Corollary:** TSP COST is $\mathbf{FP}^{\mathbf{NP}}$-complete.

***Proof:*** Consider the variant of the MAX-WEIGHT SAT problem in which we only return the optimum weight, rather than the optimum truth assignment. It is easy to see that this problem $\mathbf{FP}^{\mathbf{NP}}$-complete; the reduction is essentially the same as that in the proof of Theorem 17.4. Finally, the proof of Theorem 17.5 establishes that this variant of MAX-WEIGHT SAT can be reduced to TSP COST. □



### The Class $\mathbf{P}^{\mathbf{NP}[\log n]}$

Many other optimization problems are known to be $\mathbf{FP}^{\mathbf{NP}}$-complete: The (full optimization version of) KNAPSACK, the *weighted versions* of MAX-CUT and BISECTION WIDTH, and so on. Conspicuously absent from this list are problems whose cost is polynomially large and hence has *logarithmically many bits*, such as CLIQUE, UNARY TSP (the TSP with distances written in unary) and the unweighted versions of MAX SAT, MAX-CUT, and BISECTION WIDTH.

And there is a reason for this. Consider for example the problem

CLIQUE SIZE: Given a graph, determine the size of its largest clique.

The binary search algorithm that proves that CLIQUE SIZE is in $\mathbf{FP}^{\mathbf{NP}}$ asks only logarithmically many adaptive **NP** queries —the exact value that must be determined is between one and $n$, the number of nodes of the given graph, and so binary search takes $\log n$ queries to converge to the true value. Alternatively, we can think of an oracle algorithm for CLIQUE SIZE that asks polynomially many queries (i.e., whether the maximum clique is larger than $k$, for all values of $k$ from 1 to $n$); but the queries here *are not adaptive*, they do not depend at all on the answers of previous queries. In either case, the oracle algorithm for CLIQUE SIZE does not make full use of the polynomially many adaptive queries at its disposal (we later show that, quite remarkably, these two kinds of restrictions lead to the same class). Hence, CLIQUE SIZE, and the other optimization problems with polynomially large cost, must belong in a weaker complexity class.

And they do. Let us define $\mathbf{P}^{\mathbf{NP}[\log n]}$ to be the the class of all languages decided by a polynomial-time oracle machine which on input $x$ *asks a total of* $\mathcal{O}(\log |x|)$ SAT *queries*. $\mathbf{FP}^{\mathbf{NP}[\log n]}$ is the corresponding class of functions.

**Theorem 17.6:** CLIQUE SIZE is $\mathbf{FP}^{\mathbf{NP}[\log n]}$-complete.

***Proof:*** The proof mimics our argument that led to the $\mathbf{FP}^{\mathbf{NP}}$-completeness of TSP. We first show that the problem MAX OUTPUT$[\log n]$, the version of MAX OUTPUT in which the output has $\log n$, not $n$, bits, is $\mathbf{FP}^{\mathbf{NP}[\log n]}$-complete; the proof is completely analogous to that of Theorem 17.3. We then reduce MAX OUTPUT$[\log n]$ to MAX SAT SIZE (the version of MAX SAT in which the maximum number of satisfied clauses is sought). The idea here is that, since the output of the machine has logarithmically many bits, the weights needed in the proof of Theorem 17.4 are polynomial in $n$, and hence they can be simulated by *multiple copies of the same clause*. Finally, MAX SAT SIZE is reduced to CLIQUE SIZE by the usual reduction (via INDEPENDENT SET, recall Theorem 9.4 and its corollaries). □

Similarly, the other optimization problems with polynomial-size cost mentioned above can be shown $\mathbf{FP}^{\mathbf{NP}[\log n]}$-complete.

But what about the other restriction on $\mathbf{FP}^{\mathbf{NP}}$, in which the oracle machine

must decide which queries to ask *non-adaptively*, before it knows the answer to any query? Define $\mathbf{P}_{\parallel}^{\mathbf{NP}}$ (for an oracle machine that asks its queries *in parallel*, that is) to be the class of all languages that can be decided by an oracle machine operating as follows: On input $x$, the machine computes in polynomial time a polynomial number of instances of SAT (or any other problem in **NP**), and receives the correct answers. Based on these answers, the machine decides whether $x \in L$ in polynomial time.

**Theorem 17.7:** $\mathbf{P}_{\parallel}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{NP}[\log n]}$.

***Proof:*** To show that $\mathbf{P}^{\mathbf{NP}[\log n]} \subseteq \mathbf{P}_{\parallel}^{\mathbf{NP}}$, consider a machine that uses at most $\mathcal{O}(\log n)$ adaptive **NP** queries. When the first query is asked, there are two possibilities, one for each possible answer. For each of these two possibilities there is a next query to be asked, and two possible answers for each. It is easy to see that overall there are $2^{k \log n} = \mathcal{O}(n^k)$ queries that can be possibly asked during the computation. To simulate this machine by a non-adaptive oracle machine, we first compute all $\mathcal{O}(n^k)$ possible queries, find the answers to all of them, and from that we easily determine the correct path to be followed and answer given.

For the other direction, suppose that we have a language decidable by polynomially many non-adaptive SAT queries. We can decide this language with logarithmically many adaptive **NP** queries, as follows: First, in $\mathcal{O}(\log n)$ **NP** queries we determine (by binary search) *the precise number of "yes" answers to the non-adaptive queries.* Notice that each question in this binary search, asking whether the given set of Boolean expressions has satisfying truth assignments for at least $k$ of them, is itself an **NP** query—the $k$ satisfying truth assignments, together with an indication of which expression is satisfied by each, comprise an adequate certificate. Once the exact number $k$ of "yes" answers is known, we ask the last query: "Do there exist $k$ satisfying truth assignments for $k$ of the expressions such that, if all other expressions were unsatisfiable (which we *know* they must be...) the oracle machine would end up accepting?" □

## 17.2 THE POLYNOMIAL HIERARCHY

Now that we have defined $\mathbf{P}^{\mathbf{NP}}$ we find ourselves in a familiar position: We have defined an important deterministic complexity class (it *is* deterministic, since the oracle machines in terms of which it is defined are deterministic), and we are tempted to consider *the corresponding nondeterministic class,* $\mathbf{NP}^{\mathbf{NP}}$ Naturally, this class most likely will not be closed under complement, and hence we should also consider oracle machines that use *that class.* And so on:

**Definition 17.2:** The *polynomial hierarchy* is the following sequence of classes: First, $\Delta_0\mathbf{P} = \Sigma_0\mathbf{P} = \Pi_0\mathbf{P} = \mathbf{P}$; and for all $i \geq 0$



$\Delta_{i+1}\mathbf{P} = \mathbf{P}^{\Sigma_i\mathbf{P}}$
$\Sigma_{i+1}\mathbf{P} = \mathbf{NP}^{\Sigma_i\mathbf{P}}$
$\Pi_{i+1}\mathbf{P} = \mathbf{coNP}^{\Sigma_i\mathbf{P}}$.

We also define the *cumulative polynomial hierarchy* to be the class $\mathbf{PH} = \bigcup_{i \geq 0} \Sigma_i\mathbf{P}$. □

Since $\Sigma_0\mathbf{P} = \mathbf{P}$ does not help polynomial-time oracle machines, the first level of this hierarchy makes up our familiar important complexity classes: $\Delta_1\mathbf{P} = \mathbf{P}$, $\Sigma_1\mathbf{P} = \mathbf{NP}$, $\Pi_1\mathbf{P} = \mathbf{coNP}$. The second level starts with the class $\Delta_2\mathbf{P} = \mathbf{P}^{\mathbf{NP}}$ studied in the previous section, and continues with $\Sigma_2\mathbf{P} = \mathbf{NP}^{\mathbf{NP}}$, and its complement $\Pi_2\mathbf{P} = \mathbf{coNP}^{\mathbf{NP}}$. As with the first level, there is every reason to believe that all three classes are distinct. The same holds for the third level, and so on. Naturally, the three classes at each level are related by the same inclusions that we know about **P**, **NP**, and **coNP**. Also, each class at each level includes all classes at previous levels.

In order to show that a problem is in **NP** we are more likely to argue in terms of "certificates" or "witnesses," rather than in terms of nondeterministic Turing machines. We have found it simple and convenient to use the characterization of **NP** in terms of polynomially balanced relations (Proposition 9.1). In the polynomial hierarchy with its complex recursive definition such conceptual simplification is even more welcome, almost essential. We prove below a direct generalization of Proposition 9.1 for the polynomial hierarchy.

**Theorem 17.8:** Let $L$ be a language, and $i \geq 1$. $L \in \Sigma_i\mathbf{P}$ if and only if there is a polynomially balanced relation $R$ such that the language $\{x; y : (x, y) \in R\}$ is in $\Pi_{i-1}\mathbf{P}$ and

$$L = \{x : \text{ there is a } y \text{ such that } (x, y) \in R\}.$$

***Proof:*** By induction on $i$. For $i = 1$, the statement is exactly Proposition 9.1. So suppose that $i > 1$, and such a relation $R$ exists. We must show that $L \in \Sigma_i\mathbf{P}$. That is, we must describe a nondeterministic polynomial-time oracle machine, with a language in $\Sigma_{i-1}\mathbf{P}$ as an oracle, that decides $L$. This is easy: The nondeterministic machine on input $x$ simply guesses an appropriate $y$, and asks a $\Sigma_{i-1}\mathbf{P}$ oracle whether $(x, y) \in R$ (more correctly, since $R$ is a $\Pi_{i-1}\mathbf{P}$ relation, whether $(x, y) \notin R$).

Conversely, suppose that $L \in \Sigma_i\mathbf{P}$. We must show that an appropriate relation $R$ exists. What we know is that $L$ can be decided by a polynomial-time nondeterministic Turing machine $M^?$ using as an oracle a language $K \in \Sigma_{i-1}\mathbf{P}$. Since $K \in \Sigma_{i-1}\mathbf{P}$, by induction there is a relation $S$ recognizable in $\Pi_{i-2}\mathbf{P}$ such that $z \in K$ if and only if there is a $w$ with $(z, w) \in S$.

We must describe a polynomially balanced, polynomially decidable relation $R$ for $L$; that is, a succinct certificate for each $x \in L$. We know that $x \in L$ if and only if there is a correct, accepting computation of $M^K$ on $x$. The certificate

of $x$ will be a string $y$ recording such a computation of $M^K$ (compare with the proof of Proposition 9.1). But recall that $M^K$ is now an oracle machine with an oracle $K \in \Sigma_{i-1}\mathbf{P}$, and thus several of its steps will be queries to $K$. Some of these steps will have "yes" answers, and some "no" answers. *For each "yes" query $z_i$, our certificate $y$ also includes $z_i$'s own certificate $w_i$* such that $(z_i, w_i) \in S$. This is the definition of $R$: $(x, y) \in R$ if and only if $y$ records an accepting computation of $M^?$ on $x$, together with a certificate $w_i$ for each "yes" query $z_i$ in the computation.

We claim that checking whether $(x, y) \in R$ can be done in $\Pi_{i-1}\mathbf{P}$. First, we must check whether all steps of $M^?$ are legal; but this can be done in deterministic polynomial time. Then we must check for polynomially many pairs $(z_i, w_i)$ whether $(z_i, w_i) \in S$; but this can be done in $\Pi_{i-2}\mathbf{P}$, and thus certainly in $\Pi_{i-1}\mathbf{P}$. Finally, for all "no" queries $z_i'$ we must check that indeed $z_i' \notin K$. *But since $K \in \Sigma_{i-1}\mathbf{P}$, this is another $\Pi_{i-1}\mathbf{P}$ question.* Thus $(x, y) \in R$ if and only if several $\Pi_{i-1}\mathbf{P}$ queries all have answers "yes;" and it is easy to see that this can be done in a single $\Pi_{i-1}\mathbf{P}$ computation. □

The "dual" result for $\Pi_i\mathbf{P}$ is this:

**Corollary 1:** Let $L$ be a language, and $i \geq 1$. $L \in \Pi_i\mathbf{P}$ if and only if there is a polynomially balanced binary $R$ such that the language $\{x; y : (x, y) \in R\}$ is in $\Sigma_{i-1}\mathbf{P}$ and

$$L = \{x : \text{ for all } y \text{ with } |y| \leq |x|^k, (x, y) \in R\}.$$

***Proof:*** Just recall that $\Pi_i\mathbf{P}$ is precisely $\mathbf{co}\Sigma_i\mathbf{P}$. □

Notice that in the description of $L$ in Corollary 1 we must explicitly state for the universally quantified string $y$ the bound $|y| \leq |x|^k$. Since $R$ is known to be polynomially balanced, this constraint is, in this context, superfluous, and will be omitted. Also, we shall use quantifiers such as $\forall x$ and $\exists y$ in the descriptions of languages such as the one displayed in Corollary 2 below. This will help bring out the elegant mathematical structure of these descriptions, as well as their affinity with logic.

In order to get rid of the recursion in Theorem 17.8, let us call a relation $R \subseteq (\Sigma^*)^{i+1}$ polynomially balanced if, whenever $(x, y_1, \ldots, y_i) \in R$, we have that $|y_1|, \ldots, |y_i| \leq |x|^k$ for some $k$.

**Corollary 2:** Let $L$ be a language, and $i \geq 1$. $L \in \Sigma_i\mathbf{P}$ if and only if there is a polynomially balanced, polynomial-time decidable $(i+1)$-ary relation $R$ such that

$$L = \{x : \exists y_1 \forall y_2 \exists y_3 \ldots Q y_i \text{ such that } (x, y_1, \ldots, y_i) \in R\}$$

where the $i$th quantifier $Q$ is "for all" if $i$ is even, and "there is" if $i$ is odd.



***Proof:*** Repeatedly replace languages in $\Pi_j\mathbf{P}$ or $\Sigma_j\mathbf{P}$ by their certificate forms as in Theorem 17.8 and its Corollary 1. □

Using these characterizations we can prove the basic fact concerning the polynomial hierarchy: As it is built by patiently adding layer after layer, always using the previous layer as an oracle for defining the next, the resulting structure is extremely fragile and delicate. Any jitter, at any level, has disastrous consequences further up:

**Theorem 17.9:** If for some $i \geq 1$ $\Sigma_i\mathbf{P} = \Pi_i\mathbf{P}$, then for all $j > i$ $\Sigma_j\mathbf{P} = \Pi_j\mathbf{P} = \Delta_j\mathbf{P} = \Sigma_i\mathbf{P}$.

***Proof:*** It suffices to show that $\Sigma_i\mathbf{P} = \Pi_i\mathbf{P}$ implies $\Sigma_{i+1}\mathbf{P} = \Sigma_i\mathbf{P}$. So, consider a language $L \in \Sigma_{i+1}\mathbf{P}$. By Theorem 17.8 there is a relation $R$ in $\Pi_i\mathbf{P}$ with $L = \{x : \text{ there is a } y \text{ such that } (x, y) \in R\}$ . But since $\Pi_i\mathbf{P} = \Sigma_i\mathbf{P}$, $R$ is in $\Sigma_i\mathbf{P}$. That is, $(x, y) \in R$ if and only if there is a $z$ such that $(x, y, z) \in S$ for some relation $S \in \Pi_{i-1}\mathbf{P}$. Thus $x \in L$ if and only if there is a string $y; z$ such that $(x, y, z) \in S$, where $S \in \Pi_{i-1}\mathbf{P}$. But this means that $L \in \Sigma_i\mathbf{P}$. □

The statements of many results in complexity theory end like that of Theorem 17.9: "then for all $j > i$ $\Sigma_j\mathbf{P} = \Pi_j\mathbf{P} = \Delta_j\mathbf{P} = \Sigma_i\mathbf{P}$." This conclusion is usually abbreviated "then the polynomial hierarchy collapses to the $i$th level." For example:

**Corollary:** If $\mathbf{P} = \mathbf{NP}$, or even if $\mathbf{NP} = \mathbf{coNP}$, the polynomial hierarchy collapses to the first level. □

The last corollary makes one thing abundantly clear: In the absence of a proof that $\mathbf{P} \neq \mathbf{NP}$, there is no hope of proving that the polynomial "hierarchy" is indeed a hierarchy of classes each properly containing the next (although, once again, we strongly believe that it is). Still, the polynomial hierarchy is interesting for several reasons. First it is the polynomial analog of an important (provable) hierarchy of "more and more undecidable problems," the *arithmetic* or *Kleene hierarchy* (recall Problem 3.4.9). Second, its various levels do contain some, even though not very many, interesting and natural problems; some of them are complete. For example, consider the following decision problem:

MINIMUM CIRCUIT: Given a Boolean circuit $C$, is it true that there is no circuit with fewer gates that computes the same Boolean function?

MINIMUM CIRCUIT is in $\Pi_2\mathbf{P}$, and not known to be in any class below that. To see that it is in $\Pi_2\mathbf{P}$, notice that $C$ is a "yes" instance if and only if *for all circuits $C'$ with fewer gates there is an input $x$ for which $C(x) \neq C'(x)$*. Then use Corollary 2, noting that the last inequality can be checked in polynomial time.

It is open whether MINIMUM CIRCUIT is $\Pi_2\mathbf{P}$-complete. Fortunately, and as usual, for every $i \geq 1$ there is a version of satisfiability very appropriate for the corresponding level of the hierarchy:

$\text{QSAT}_i$ (for *quantified satisfiability with $i$ alternations of quantifiers*): Given a Boolean expression $\phi$, with Boolean variables partitioned into $i$ sets $X_1, \ldots, X_i$, is it true that for all partial truth assignments for the variables in $X_1$ there is a partial truth assignment for the variables in $X_2$ such that for all partial truth assignments for the variables in $X_3$, and so on up to $X_i$, $\phi$ is satisfied by the overall truth assignment? We represent an instance of $\text{QSAT}_i$ as follows (by slightly abusing our first-order quantifiers):

$$\exists X_1 \forall X_2 \exists X_3 \ldots Q X_i \quad \phi$$

where, as usual, the quantifier $Q$ is $\exists$ if $i$ is odd and $\forall$ if $i$ is even.

**Theorem 17.10:** For all $i \geq 1$ $\text{QSAT}_i$ is $\Sigma_i\mathbf{P}$-complete.

***Proof:*** Both directions rest heavily on Theorem 17.8 and its Corollary 2. To show that $\text{QSAT}_i \in \Sigma_i\mathbf{P}$ we just note that it is defined in the form required by Corollary 2.

To reduce any language $L \in \Sigma_i\mathbf{P}$ to $\text{QSAT}_i$, we first bring $L$ in the form of Corollary 2 to Theorem 17.8. Since the relation $R$ can be decided in polynomial time, there is a polynomial-time deterministic Turing machine $M$ that accepts precisely those input strings $x; y_1; \ldots; y_i$ such that $(x, y_1, \ldots, y_i) \in R$. Suppose that $i$ is odd (the even $i$ case is symmetric). Using Cook's theorem (and thus not even taking advantage of the fact that $M$ is deterministic) we can write a Boolean formula $\phi$ that captures the computation of this machine. The variables in $\phi$ can be divided into $i + 2$ classes. Variable set $X$ contains the variables standing for the symbols in the input string before the first ";" symbol—recall that the input of $M$ is of the form $x; y_1; \ldots; y_i$. Similarly, variable set $Y_1$ stands for the next input symbols, and so on up to $Y_i$. These $i + 1$ sets are called *the input variables*. Finally, there is a (probably much larger) set of Boolean variables $Z$ that incorporates all other aspects of the computation of $M$.

Now, given any fixed values for the variables in $X, Y_1, \ldots, Y_i$, the resulting expression is satisfiable if and only if the values of the input variables spell a string in the language decided by $M$, that is, if they are related by $R$.

Consider now any string $x$, and substitute in $\phi$ the corresponding Boolean values $\hat{X}$ for $X$. We know that $x \in L$ if and only if there is a $y_1$ such that for all $y_2$ etc., there is a $y_i$ (remember, $i$ is odd) such that $R(x, y_1, \ldots, y_i)$. But this, in terms of the expression $\phi$, means that for these particular values $\hat{X}$ there are values for $Y_1$ such that for all values of $Y_2$ etc., there is a value for $Y_i$ *and there is a value of* $Z$, such that $\phi$ evaluates to **true**. Thus $x \in L$ if and only if $\exists Y_1 \forall Y_2 \ldots \exists Y_i; Z\phi(\hat{X})$, which is an instance of $\text{QSAT}_i$. □

How about the cumulative hierarchy **PH**, does it have complete sets? It turns out that it probably does not. This is not because **PH** is a "semantic class"—it is not. The reason is a little more subtle (compare with Problem 8.4.2).



**Theorem 17.11:** If there is a **PH**-complete problem, then the polynomial hierarchy collapses to some finite level.

***Proof:*** Suppose that $L$ is **PH**-complete. Since $L \in \mathbf{PH}$, there is an $i \geq 0$ such that $L \in \Sigma_i\mathbf{P}$. But any language $L' \in \Sigma_{i+1}\mathbf{P}$ reduces to $L$. Since all levels of the polynomial hierarchy are closed under reductions, this means that $L' \in \Sigma_i\mathbf{P}$, and hence $\Sigma_i\mathbf{P} = \Sigma_{i+1}\mathbf{P}$. □

There is a rather obvious upper bound on the power of the polynomial hierarchy: Polynomial space. Indeed, starting fron the characterization in Corollary 2 of Theorem 17.8, it is easy to see that the search for the strings $y_1, y_2, \ldots, y_i$ can comfortably fit within polynomial space. In fact, in the next chapter we shall see that **PSPACE** is in some sense a generalization and extension of the polynomial hierarchy.

**Proposition 17.1: PH $\subseteq$ PSPACE.** □

But is **PH** = **PSPACE**? This is an open and intriguing question. However notice this curious fact: If **PH** = **PSPACE** then by Theorem 17.11 **PH** has complete problems (**PSPACE** has), and thus the polynomial hierarchy collapses. Although the **PH** = **PSPACE** eventuality would seem to be "stretching" the polynomial hierarchy upwards, and therefore to strengthen it, in fact it does the opposite. Finally, **PH** has a very natural logical characterization (arguably more natural than Fagin's theorem, see Problem 17.3.10).

### BPP and Polynomial Circuits

When studying **BPP** in Section 11.2 we noted that it is not known to be contained in **NP** (or **coNP**; since **BPP** is closed under complementation, it is a subset of both or neither). We can now show by a probabilistic technique that it is in the second level of the polynomial hierarchy:

**Theorem 17.12: BPP $\subseteq \Sigma_2\mathbf{P}$.**

***Proof:*** Let $L \in \mathbf{BPP}$. All we know about $L$ is that there is a precise Turing machine $M$, with computations of length $p(n)$ on inputs of length $n$, that decides $L$ by clear majority. For each input $x$ of length $n$, let $A(x) \subseteq \{0,1\}^{p(n)}$ denote the set of accepting computations (the choices that lead to "yes.") We can assume that if $x \in L$ then $|A(x)| \geq 2^{p(n)}(1 - \frac{1}{2^n})$, and if $x \notin L$ then $|A(x)| \leq 2^{p(n)}\frac{1}{2^n}$. That is, the probability of a false answer (false positive or false negative) is at most $\frac{1}{2^n}$, instead of the usual $\frac{1}{4}$. This can be assured by repeating the **BPP** algorithm enough times and taking the majority outcome (recall the discussion in Section 11.3).

Let $U$ be the set of all bit strings of length $p(n)$. For $a, b \in U$ define $a \oplus b$ to be the bit string which is the componentwise exclusive or of the two bit strings. For example, $1001001 \oplus 0100101 = 1101100$. This operation has some very useful properties. First, $a \oplus b = c$ if and only if $c \oplus b = a$. That

is, the function "$\oplus b$" applied to $a$ twice gets us back to $a$. As a result, the function "$\oplus b$" is one-to-one (because its argument can be recovered). Second, if $a$ is a fixed string and $r$ a *random* string, drawn by flipping an unbiased coin independently $p(n)$ times, then $r \oplus a$ is also a random bit string. This is because "$\oplus a$" is a permutation of $U$, and thus does not affect the uniform distribution.

Let $t$ be a bit string of length $p(n)$, and consider the set $A(x) \oplus t = \{a \oplus t : a \in A(x)\}$. We call this set *the translation of* $A(x)$ *by* $t$. Since the function $\oplus t$ is one-to-one, the translation of $A(x)$ has the same cardinality as $A(x)$. We shall prove the following intuitive fact: If $x \in L$, since $A(x)$ is so large in this case, we can find a relatively small set of translations that covers all of $U$. However, if $x \notin L$, then $A(x)$ is so small that no such set of translations can exist.

More formally, suppose that $x \in L$, and consider a *random* sequence of $p(n)$ translations, $t_1 \ldots, t_{p(n)} \in U$; they are obtained by drawing $p(n)^2$ bits independently with probability $\frac{1}{2}$. Fix a string $b \in U$. We say that these translations *cover* $b$ if $b \in A(x) \oplus t_j$ for some $j \leq p(n)$. What is the probability that a point $b$ is covered? $b \in A(x) \oplus t_j$ if and only if $b \oplus t_j \in A(x)$. And since $b \oplus t_j$ is as random as $t_j$, and we have assumed that $x \in L$, we conclude that $\mathbf{prob}[b \notin A(x) \oplus t_j] = \frac{1}{2^n}$. Therefore, the probability that $b$ is *not* covered by any $t_j$ is precisely that number raised to $p(n)$, $2^{-np(n)}$.

So, every point in $U$ fails to be covered with probability $2^{-np(n)}$; it follows that the probability that there is a point that is not covered is at most $2^{-np(n)}$ times the cardinality of $U$, or $2^{-(n-1)p(n)} < 1$. Thus, a sequence of randomly drawn translations $T = (t_1, \ldots, t_{p(n)})$ has a positive (in fact, overwhelming) probability that it covers all of $U$. We must conclude that *there is at least one* $T$ *that covers all of* $U$.

Conversely, suppose that $x \notin L$. Then the cardinality of $A(x)$ is an exponentially small fraction of that of $U$, and obviously (for large enough $n$) there is no sequence $T$ of $p(n)$ translations that cover all of $U$. We conclude that *there is a sequence* $T$ *of* $p(n)$ *translations that cover* $U$ *if and only if* $x \in L$.

The proof that $L \in \Sigma_2\mathbf{P}$ now follows easily from Corollary 2 of Theorem 17.8: We have shown that $L$ can be written as

$$L = \{x : \text{ there is a } T \in \{0,1\}^{p(n)^2} \text{ such that for all } b \in U \\ \text{there is a } j \leq p(n) \text{ such that } b \oplus t_j \in A(x)\}.$$

But this is precisely the form of languages in $\Sigma_2\mathbf{P}$ according to the corollary. The last existential quantifier "there is a $j$ such that..." does *not* affect the position of $L$ in the polynomial hierarchy: It quantifies over polynomially many possibilities, and is therefore an "or" in disguise. To put it otherwise, the whole line "there is a $j \leq p(n)$ such that $b \oplus t_j \in A(x)$" can be tested in polynomial time by trying all $t_j$'s. □



Since **BPP** is closed under complement, we have in fact proved:

**Corollary:** $\mathbf{BPP} \subseteq \Sigma_2\mathbf{P} \cap \Pi_2\mathbf{P}$. □

We conclude our discussion of the polynomial hierarchy by an interesting result related to circuit complexity. In Section 14.4 we articulated an important "Conjecture B" that strengthens $\mathbf{P} \neq \mathbf{NP}$, namely that SAT (or any other **NP**-complete problem) has no polynomial circuits (uniform or not). The following result adds much credibility to the conjecture:

**Theorem 17.13:** If SAT has polynomial circuits, then the polynomial hierarchy collapses to the second level.

***Proof:*** The proof is a nice application of the self-reducibility of SAT (recall the proof of Theorems 13.2 and 14.3). That SAT is self-reducible means that there is a polynomial-time algorithm for SAT that invokes SAT on smaller instances. That is, there is a polynomial-time oracle machine $M^{\text{SAT}}$ deciding SAT *with* SAT *as an oracle*, only with the additional restriction that, on input of length $n$, its oracle string can contain at most $n-1$ symbols.

The proof rests on an important consequence of self-reducibility: *Self-testing*. Suppose that there is a family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$ deciding SAT. In the proof we shall allow for the self-reducibility machine $M^{SAT}$ to use as its oracle, instead of SAT, *an initial segment* $\mathcal{C}_n = (C_0, C_1, \ldots, C_n)$ *of this family*. That is, once a query appears in its query string, the machine $M^{\mathcal{C}_n}$ invokes not SAT, but the appropriate circuit in the segment, assuming that the length of the query is at most $n$ (and we know that the queries of $M$ have small length). We say that the initial segment $\mathcal{C}_n$ *self-tests* if for all Boolean expressions $w$ of size up to $n$

$$M^{\mathcal{C}_n}(w) = \mathcal{C}_n(w).$$

That is, all Boolean expressions $w$ fed into the appropriate circuit give the same answer as when they are the input of the self-reducibility machine for SAT, with the circuit segment as oracle. If the self-testing equality holds for all $w$, this means (by induction of the size of $w$) that $\mathcal{C}_n$ is indeed a correct initial segment of a circuit family for SAT.

On the assumption that SAT has polynomial circuits we must show that $\Sigma_j\mathbf{P} = \Sigma_2\mathbf{P}$ for all $j$. By Theorem 17.9 we need only show that $\Sigma_3\mathbf{P} = \Sigma_2\mathbf{P}$. So, we are given an $L \in \Sigma_3\mathbf{P}$ and we have to show it is in $\Sigma_2\mathbf{P}$. We can assume that $L$ is of this form:

$$L = \{x : \exists y \forall z (x, y, z) \in R\},$$

where $R$ is a polynomially balanced relation decidable in **NP**—this is a simple variant of Corollary 2 to Theorem 17.8, with the recursion stopped one step earlier. Since $R$ is decidable in **NP** and SAT is **NP**-complete, there is a reduction

$F$ such that $(x, y, z) \in R$ if and only if the Boolean expression $F(x, y, z)$ is satisfiable. Suppose that, on input $x$, the largest expression $F(x, y, z)$ that can be constructed is of length at most $p(|x|)$. Since $R$ is polynomially balanced and $F$ polynomial-time, $p(n)$ is a polynomial.

To show that $L$ is in $\Sigma_2\mathbf{P}$, we shall argue that $x \in L$ if and only if the following holds:

*There exists an initial segment $C_{p(|x|)}$ and there exists a string $y$ such that for all strings $z$ and expressions $w$—all of length at most $p(|x|)$—we have: (a) $C_{p(|x|)}$ self-tests successfully on $w$, that is, $M^{C_n}(w) = C_n(w)$, and (b) $C_{p(|x|)}$ outputs* **true** *on expression $F(x, y, z)$.*

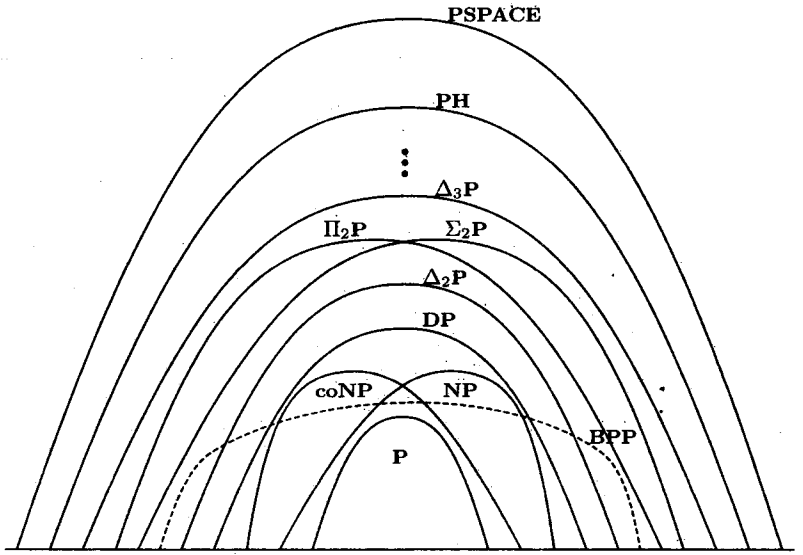
Notice that, since the above condition involves two alternations of quantifiers, and the innermost property can be tested in polynomial time, this would settle that $L \in \Sigma_2\mathbf{P}$.

If the above condition holds, then by (a) we know that $C_{p(|x|)}$ is a correct initial segment of a circuit family for SAT, and thus it can be used to correctly establish in (b) that $R(x, y, z)$, and thus the condition implies $x \in L$. Conversely, if $x \in L$ then there is a $y$ such that for all $z$ $R(x, y, z)$. Furthermore, by our hypothesis that SAT has polynomial circuits, we know that a correct segment exists that will self-test. The same segment will then certify that $(x, y, z) \in R$ for the appropriate $y$ and $z$. The proof is complete. □



## 17.3 NOTES, REFERENCES, AND PROBLEMS

### 17.3.1 Class review:




The class **DP** was introduced in

- C. H. Papadimitriou and M. Yannakakis "The complexity of facets (and some facets of complexity)," *Proc. 24th ACM Symp. on the Theory of Computing*, pp. 229–234, 1982; also, *J.CSS 28*, pp. 244–259, 1984.

Many **DP**-completeness results can be found in this paper, and also in

- C. H. Papadimitriou and D. Wolfe "The complexity of facets resolved," *Proc. 16th IEEE Symp. on the Foundations of Computer Science*, pp. 74–78, 1985; also, *J.CSS 37*, pp. 2–13, 1987.

As for UNIQUE SAT, there is an oracle under which it is *not* **DP**-complete, and so it appears to be a less worthy representative of **DP** than the other problems we have seen:

- A. Blass and Y. Gurevich "On the unique satisfiability problem," *Information and Control, 55*, pp. 80–88, 1982.

But see Problem 18.3.5 in this regard.

The "D" in **DP** stands for "difference": A language in **DP** is just the set-theoretic difference of two languages in **NP**. The corresponding class of differences of two *recursively enumerable languages* was defined in

- H. Rogers *Theory of Recursive Functions and Effective Computability*, MIT Press, Cambridge, Massachusetts, 1987 (second edition).

Incidentally, the class that we call **DP** is denoted in the literature as $D^p$. We have adopted this new notation, as well as that for the polynomial hierarchy, whose classes are also usually denoted $\Sigma_2^p$ etc., in order to arrive at a uniform nomenclature for all classes "between" **P** and **PSPACE**: All names end with **P**, and the prefix is indicative of the mode of computation involved.

**17.3.2 Problem:** (a) Show that the problems CRITICAL SAT, CRITICAL HAMILTON PATH, and CRITICAL 3-COLORABILITY are in **DP**.

(b) Show that UNIQUE SAT is in **DP**.

(c) Show that if UNIQUE SAT is in **NP** then **NP** = **coNP**.

**17.3.3 Problem:** Show that **DP** $\subseteq$ **PP**.

**17.3.4 True or false?** (Or equivalent to **P** = **NP**?)
(a) If $L$ is **NP**-complete and $L'$ is **coNP**-complete, then $L \cap L'$ is **DP**-complete.
(b) If $L$ is **NP**-complete $L \cap \overline{L}$ is **DP**-complete.

**17.3.5** **DP** can be extended to classes in which an arbitrary *bounded* number of SAT queries are allowed. The resulting *Boolean hierarchy*, somewhat sparse in natural complete problems, was studied in

- J.-Y. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung "The Boolean hierarchy I: Structural properties" *SIAM Journal on Computing 17*, pp. 1232–1252, 1988. Part II: Applications in *vol. 18*, pp. 95–111, 1989.

**17.3.6** Show that the following language is $\Delta_2$**P**-complete: Given an instance of the TSP, is the optimum tour length *odd?* Is the optimum tour *unique?*

**17.3.7** The relationship between $\mathbf{FP}^{\mathbf{NP}}$ and optimization problems (Theorems 17.5 and 17.6), hinted at in

- C. H. Papadimitriou "The complexity of unique solutions," *Proc. 23rd IEEE Symp. on the Foundations of Computer Science*, pp. , pp. 14–20, 1983; also *J.ACM 31*, pp. 492–500, 1984,

was established in

- M. W. Krentel "The complexity of optimization problems," *Proc. 18th ACM Symp. on the Theory of Computing*, pp. 79–86, 1986; also *J.CSS 36*, pp. 490–509, 1988.

Theorem 17.7 is from

- S. R. Buss and L. Hay "On truth-table reducibility to SAT and the difference hierarchy over NP," *Proc. 3rd Symp. on Structure in Complexity Theory*, pp. 224–233, 1988.

**17.3.8 Problem:** Show that, if **NP** $\subseteq$ **TIME**$(n^{\log n})$, then **PH** $\subseteq$ **TIME**$(n^{\log^k n})$.

**17.3.9** The polynomial hierarchy was introduced and studied in

- L. J. Stockmeyer "The polynomial hierarchy," *Theor. Comp. Science, 3*, pp. 1–22, 1976.



Theorem 17.10 on the completeness of $\mathrm{QSAT}_i$ is from

- C. Wrathall "Complete sets for the polynomial hierarchy," *Theor. Comp. Science, 3*, pp. 23–34, 1976.

**17.3.10** Show that **PH** is the class of all graph-theoretic properties that can be expressed in *second-order logic.* (Compare with Theorem 8.3.)

**17.3.11** Suppose that the cities in a Euclidean instance of the TSP are the vertices of a convex polygon. Then not only is the optimum tour easy to find (it is the perimeter of the polygon), but the instance has the *master tour property:* There is a tour such that the optimum tour of any subset of cities is obtained by simply omitting from the master tour the cities not in the subset.

**Problem:** Show that deciding whether a given instance of the TSP has the master tour property is in $\Sigma_2$**P**.

**17.3.12** We know that converting Boolean expressions in disjunctive normal form to conjunctive normal form can be exponential in the worst case, simply because the output may be exponentially long in the input. But suppose the output is small. In particular, consider the following problem: We are given a Boolean expression in disjunctive normal form, and an integer $B$. We are asked whether the conjunctive normal form has $B$ or fewer clauses.

**Problem:** Show that the problem is in $\Sigma_2$**P**.

Incidentally, the previous two problems are two good candidates for natural $\Sigma_2$**P**-complete problems.

**17.3.13 Default logic.** A *default* is an object of the form $\delta = \frac{\phi:\psi\&\chi}{\chi}$, where $\phi$, $\chi$, and $\psi$ are Boolean expressions in conjunctive normal form called the *prerequisite*, the *justification*, and the *consequence* of $\delta$, respectively. Intuitively, the above default means that *if $\phi$ has been established, and neither $\neg\psi$ nor $\neg\chi$ have been established, then we can "assume $\chi$ by default."* For example, here is the intended use of this device in artificial intelligence:

$$\frac{\text{bird(Tweety)} : \neg\ \text{penguin(Tweety)}\&\text{flies(Tweety)}}{\text{flies(Tweety)}}.$$

A *default theory* is a pair $D = (\alpha_0, \Delta)$, where $\alpha_0$ is a Boolean expression (intuitively, comprising our initial knowledge of the world), and $\Delta$ is a set of defaults.

The semantics of a default theory is defined in terms of a peculiar kind of model called an *extension.* Given a default theory $(\alpha_0, \Delta)$, an *extension* of $(\alpha_0, \Delta)$ is an expression $\alpha$ such that the following sequence of expressions in conjunctive normal form, starting from $\alpha_0$, converges to $\alpha$:

$$\alpha_{i+1} = \Theta(\alpha_i \cup \{\chi : \text{for some default } \tfrac{\phi:\psi\&\chi}{\chi} \in \Delta, \quad \alpha_i \Rightarrow \phi \text{ and } \alpha \not\Rightarrow \neg(\psi \wedge \chi)\}).$$

Here $\Theta(\phi)$ denotes the deductive closure, that is, all clauses deducible from $\phi$. That is, at each stage we add to $\alpha_i$ all default consequences whose prerequisites have been established already, and whose justifications and consequences do not contradict the

extension sought; we then take all possible logical consequences of the resulting expression. Notice that the sought extension $\alpha$ appears in the iteration. Obviously this process must converge after $|\Delta|$ or fewer steps, but not necessarily to $\alpha$; if not, $\alpha$ fails to be an extension. Default theories may have one, many, or no extensions. Let DEFAULT SAT be the following problem: "Given a default theory, does it have an extension?"

(a) Show that DEFAULT SAT is $\Sigma_2$**P**-complete.

(b) Consider the special case of DEFAULT SAT in which all defaults are of the form $\frac{:x\&y}{x}$, where $x$ and $y$ are literals. Show that DEFAULT SAT in this special case is **NP**-complete.

Default logic was proposed and studied by Ray Reiter

- R. Reiter "A logic for default reasoning," *Artificial Intelligence 13*, 1980.

It is one of the many formalisms representing attempts in artificial intelligence to capture the elusive notion of *common-sense reasoning*, see for example

- M. Genesareth and N. Nilsson *Logical Foundations of Artificial Intelligence*, Morgan-Kaufman, San Mateo, California, 1988.

The complexity results in parts (a) and (b) above are from

- C. H. Papadimitriou and M. Sideri "On finding extensions of default theories," *Proc. International Conference in Database Theory*, pp. 276–281, Lecture Notes in Computer Science, Springer-Verlag, 1992.

A very comprehensive complexity-theoretic treatment of this and other formalizations of common-sense reasoning, resulting in several natural problems complete for various levels of the polynomial hierarchy, is contained in

- G. Gottlob "Complexity results in non–monotonic logics," CD–TR 91/24, T. U. Wien, August 1991. Also, *J. of Logic and Computation*, June 1992.

**17.3.14** There are now oracles known with respect to which **PH** $\neq$ **PSPACE** and the polynomial hierarchy is infinite, separated from **PSPACE**, or collapses to any desired level, see

- A. C.-C. Yao "Separating the polynomial hierarchy by oracles," *Proc. 26th IEEE Symp. on the Foundations of Computer Science*, pp. 1–10, 1985, also
- J. Håstad *Computational Limitations for Small-depth Circuits*, MIT Press, Cambridge, 1987, and
- K.-I. Ko "Relativized polynomial-time hierarchies with exactly $k$ levels" *SIAM J. Computing, 18*, pp. 392–408, 1989.

Both questions had been open for some time. In fact, separation from **PSPACE** is known to hold for a random oracle

- J.-Y. Cai "With probability one, a random oracle separates PSPACE from the polynomial hierarchy," *Proc. 18th ACM Symp. on the Theory of Computing*, pp. 21–29, 1986; also, *J.CSS, 38*, pp. 68–85, 1988.

**17.3.15** A weaker form of Theorem 17.12 was announced in



- M. Sipser "A complexity theoretic approach to randomness," *Proc. 15th ACM Symp. on the Theory of Computing*, pp. 330–335, 1983.

Our proof is from

- C. Lautemann "BPP and the polynomial time hierarchy," *IPL 17*, pp. 215–218, 1983.

Theorem 13.13 is form

- R. M. Karp and R. J. Lipton "Some connections between nonuniform and uniform complexity classes," *Proc. 12th ACM Symp. on the Theory of Computing*, pp. 302–309, 1980; retitled "Turing machines that take advice," *Enseign. Math., 28*, pp. 191–201, 1982,

where its current strong form is attributed to Mike Sipser.

CHAPTER

# 20 A GLIMPSE BEYOND

*In this chapter we shall see at last some truly, provably intractable problems...*

## 20.1 EXPONENTIAL TIME

Recall our definition of exponential time

$$\mathbf{EXP} = \mathbf{TIME}(2^{n^k}),$$

and the corresponding nondeterministic class

$$\mathbf{NEXP} = \mathbf{NTIME}(2^{n^k}).$$

The counterpart of $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ at this level is the $\mathbf{EXP} \stackrel{?}{=} \mathbf{NEXP}$ question — unfortunately, we are not any closer to resolving it. However, there is something simple that can be said about the relation between these two problems:

**Theorem 20.1:** If $\mathbf{P} = \mathbf{NP}$ then $\mathbf{EXP} = \mathbf{NEXP}$.

***Proof:*** Let $L \in \mathbf{NEXP}$; under the assumption that $\mathbf{P} = \mathbf{NP}$, we shall show that it is in **EXP**. By definition, $L$ is decided by a nondeterministic Turing machine $N$ in time $2^{n^k}$, for some $k$. Consider now the "exponentially padded version" of $L$:

$$L' = \{x\sqcap^{2^{|x|^k}-|x|} : x \in L\}.$$

That is, $L'$ consists of all strings $x$ in $L$ padded by enough "quasiblanks" to bring the total length of the string to $2^{|x|^k}$.

*We claim that* $L' \in \mathbf{NP}$. This is easy to show: The polynomial-time nondeterministic machine that decides $L'$ is precisely $N$, slightly modified so

that it first checks whether its string ends in exponentially many quasiblanks; if not, it rejects, otherwise it simulates $N$, treating quasiblanks as blanks. The machine works in polynomial time simply because its input is exponentially long.

Since $L' \in \mathbf{NP}$, and we are assuming that $\mathbf{P} = \mathbf{NP}$, we know that $L' \in \mathbf{P}$. Thus there is a deterministic Turing machine $M'$ that decides $L'$ in time $n^\ell$, say. We can in fact assume that $M'$ is a machine with input and output, so that it never writes on its input string. *We shall now invert the previous construction* to get a deterministic machine $M$ that decides $L$ in time $2^{n^\ell}$, for some integer $\ell$, thus completing the proof. But this is easy to do: $M$ on input $x$ simply simulates $M'$ on input $x\sqcap^{2^{|x|^k}-|x|}$. The only difficulty, tracking down $M'$'s input cursor when it wanders off in the $\sqcap$'s, can be solved by maintaining the position of the input cursor as an integer in binary. □

Contrapositively, if $\mathbf{EXP} \neq \mathbf{NEXP}$ then $\mathbf{P} \neq \mathbf{NP}$. That is, *class equality propagates upwards,* while *class inequality propagates downwards.* In other words, showing that $\mathbf{EXP} \neq \mathbf{NEXP}$, as we fully believe is the case, might turn out to be even harder than showing $\mathbf{P} \neq \mathbf{NP}$; conceivably $\mathbf{P} \neq \mathbf{NP}$ and still $\mathbf{EXP} = \mathbf{NEXP}$ (see the references).

This can be generalized, of course (for a proof, see Problem 20.2.3):

**Corollary:** If $f(n)$ and $g(n) \geq n$ are proper functions, then $\mathbf{TIME}(f(n)) = \mathbf{NTIME}(f(n))$ implies $\mathbf{TIME}(g(f(n))) = \mathbf{NTIME}(g(f(n)))$. □

Also, for the analog of Theorem 20.1 in space complexity, as well as for the interaction between time and space complexity see Problem 20.2.4.

It is interesting to compare **EXP** and **NEXP** with two other classes that capture a more benign aspect of exponential time:

$$\mathbf{E} = \mathbf{TIME}(k^n), \text{ and } \mathbf{NE} = \mathbf{NTIME}(k^n).$$

That is, the time bounds in these classes have *linear*, not polynomial, exponent. The main drawback of these classes is that they are not closed under reductions (recall Problem 7.4.4). Still, they are closely related to **EXP** and **NEXP**:

**Lemma 20.1:** For any language $L$ in **NEXP** there is a language $L'$ in **NE** such that $L$ reduces to $L'$.

***Proof:*** : Just notice that, if $L \in \mathbf{TIME}(2^{n^k})$, then $L' = \{x\sqcap^{|x|^k} : x \in L\}$ is in **NE**, and $L$ reduces to it. □

To put it otherwise, **NEXP** is the closure under reductions of **NE**. This lemma is useful in proving **NEXP**-completeness results.

### Succinct Problems

But what kinds of complete problems do these classes have? In the next subsection we shall see some interesting problems from logic that capture this level of



complexity. Another interesting kind of **EXP**- and **NEXP**-complete problems comes from the consideration that led to Theorem 20.1: **NEXP** and **EXP** are nothing else but **P** and **NP** *on exponentially more succinct input.*

Several **NP**-complete graph-theoretic problems (including MAX CUT, MAX FLOW, BISECTION WIDTH, and so on) have important applications to the automated design of VLSI chips. In chip design, however, there are ways of describing chips that are not explicit and direct, listing all components and connections of the chip, but *succinct* and *implicit*, describing a chip in terms of repeated patterns and encoded configurations. For a simple example, a highly regular circuit could be described like this:

"Repeat the (given) component $C$ horizontally and vertically in an $N \times M$ grid $(i,j), i = 1,\ldots,N; j = 1,\ldots,M$ (where $N$ and $M$ are given large integers), except at the positions $i = j$, the positions $i = 2$, and the positions $i = N-1$; at these positions place the other given component $C'$ . . ."

And so on. Since $M$ and $N$ are given in binary, such descriptions are conceivably *exponentially more succinct* than the circuits they describe. Accordingly, the graphs that abstract the structure and operation of such circuits (and for which we need to solve several computational problems, such as MAX-CUT, BISECTION WIDTH, and so on) are described in a manner exponentially more succinct than our usual explicit representation that lists all edges.

We can define a way of representing graphs that captures the effect of such "hardware description languages." A *succinct representation of a graph with $n$ nodes*, where $n = 2^b$ is a power of two, is a Boolean circuit $C$ with $2b$ input gates. The graph represented by $C$, denoted $G_C$, is defined as follows: The nodes of $G_C$ are $\{1,2,\ldots,n\}$. And $[i,j]$ is an edge of $G_C$ *if and only if $C$ accepts the binary representations of the $b$-bit integers $i, j$ as inputs.*

The problem SUCCINCT HAMILTON PATH is now this: Given the succinct representation $C$ of a graph $G_C$ with $n$ nodes, does $G_C$ have a Hamilton path? Similarly for SUCCINCT MAX CUT, SUCCINCT BISECTION WIDTH, or the succinct version of any graph-theoretic problem (for these latter problems, as well as for any graph-theoretic optimization problem, a binary integer budget/goal $K$ is provided along with $C$).

We can also define SUCCINCT 3SAT, SUCCINCT CIRCUIT SAT, and SUCCINCT CIRCUIT VALUE. To encode Boolean circuits, we first assume that all gates are predecessors to at most two other gates. That is, we think that a gate in the circuit has four *neighbors*, of which the first two are predecessors and the other two successors (if there are fewer neighbors, the missing neighbors are set to a fictitious gate 0, say). The succinct representation of a Boolean circuit is another Boolean circuit with many outputs. On input of the form $i;k$, where $i$ is a gate number in binary and $0 \leq k \leq 3$, the output of the encoding circuit is of the form $j;s$, where gate $j$ is the $k$th neighbor of gate $i$, and $s$ encodes the

sort (AND, OR, NOT) of gate $i$. To encode succinctly a Boolean expression in conjunctive normal form, we assume that all clauses have three literals and each literal appears three times (again, with missing literals and clauses represented as 0). Suppose that the encoded expression has $n$ variables and $m$ clauses. The encoding circuit on input $0;i;k$, where $i \leq n$ and $k \leq 2$, returns the index of the clause where the literal $\neg x_i$ appears for the $k$th time; on input $1;i;k$ it returns the number of the clause where the literal $x_i$ appears for the $k$th time; and on input $2;i;k$, with $i \leq m$ and $1 \leq k \leq 3$, it returns the $k$th literal of clause $i$. It should be clear that all Boolean expressions can be thus encoded. SUCCINCT 3SAT is now the problem of telling, given a circuit $C$, whether the Boolean expression $\phi_C$ represented by it† is satisfiable. Similarly for SUCCINCT CIRCUIT SAT and and SUCCINCT CIRCUIT VALUE.

**Theorem 20.2:** SUCCINCT CIRCUIT SAT is **NEXP**-complete.

***Proof:*** It is clear that the problem is in **NEXP**: A nondeterministic machine can guess a satisfying truth assignment for all gates $[t_1, \ldots, t_N]$, where $N$ is exponential in the size of the input $C$, and then verify that the output gate is **true** and all gates have legitimate values.

To prove completeness, we shall reduce any language in **NEXP** to SUCCINCT CIRCUIT SAT. So, suppose that $L$ is a language decided by a nondeterministic Turing machine $N$ in time $2^n$ (here we use Lemma 20.1). For each input $x$ we shall construct an instance $R(x)$ of SUCCINCT CIRCUIT SAT. $R(x)$ is a circuit, which encodes another circuit $C_{R(x)}$, with the following property: $C_{R(x)}$ is satisfiable if and only if $x \in L$. The circuit $C_{R(x)}$ is essentially the one constructed in the proof of Cook's theorem (Theorem 8.2), only exponentially larger. That is, there are now going to be $2^n \times 2^n$ copies of the basic circuit $C$. The gates of $C_{R(x)}$ will thus be of the form $i, j, k$, where $i, j \leq 2^n$, and $k \leq |C|$, where $C$ is the size of the basic circuit. $R(x)$ on input $i, j, k$ outputs in binary $s; i, j, k'; i, j, k''$, where $s$ is an encoding of the sort of gate $k$ of $C$, and $k', k''$ are the predecessors of $k$ in $C$. It is very easy to finish the construction of $R(x)$ so that it appropriately identifies inputs and outputs of adjacent copies of $C$, and forces the upper and lower row and leftmost and rightmost columns of the computation table to contain the correct symbols (recall Figure 8.3). □

For more discussion of the complexity of problems of this sort see Problem 20.2.9. A proof very similar to that of Theorem 20.2 above establishes the following:

**Corollary 1:** The problems SUCCINCT 3SAT and SUCCINCT HAMILTON PATH are **NEXP**-complete.

***Proof:*** It is clear that SUCCINCT CIRCUIT SAT and SUCCINCT 3SAT are in **NEXP**. For completeness, the ordinary reduction from CIRCUIT SAT to 3SAT (recall Example 8.3) can be modified in a straight-forward way to establish that SUCCINCT CIRCUIT SAT is reducible to SUCCINCT 3SAT. Given a circuit $K$ encoding some circuit, call it $C_K$, we must construct a circuit $R(K)$ which encodes an equivalent expression $\phi_{R(K)}$. Expression $\phi_{R(K)}$ must have as many variables as $C_K$ has gates, and twice as many clauses, with a structure that directly reflects that of $C_K$. This is quite easy to do; $R(K)$ is essentially $K$ with some simple pre-processing of the input and post-processing of the output to conform with the new conventions.

We shall now reduce SUCCINCT 3SAT to SUCCINCT HAMILTON PATH. Given a circuit $C$ describing a Boolean expression $\phi_C$, we can construct the circuit $R(C)$ which encodes the graph resulting from our reduction from 3SAT to HAMILTON PATH (Theorem 9.7). The graph will have three nodes for each variable corresponding to the choice gadget, recall Figure 9.7), three for each clause (the nodes of the triangle in the constraint gadget, Figure 9.7), plus twelve nodes for each occurrence of a literal to a clause (the exclusive-or gadget): Whether any two such nodes are connected by an edge in the graph can be easily determined from the indices of the two nodes, plus the occurrence relation of the Boolean expression, as described by circuit $C$. Therefore, the circuit $R(C)$ that encodes the resulting graph can be again designed so that it is $C$ with some easy pre-processing and post-processing of indices. □

**Corollary 2:** SUCCINCT CIRCUIT VALUE is **EXP**-complete.

***Proof:*** It is clear that it is in **EXP**. The deterministic version of the proof of Theorem 20.2 (recall Theorems 8.1 and 8.2) establishes completeness. □

Finally we have the exponential counterpart of Corollary 1 of Theorem 16.5 (which could also be obtained by a "padding argument" like the one used in the proof of Theorem 20.1):

**Corollary 3:** **EXP** coincides with alternating polynomial space.

***Proof:*** SUCCINCT CIRCUIT VALUE is complete for both classes (see Problem 20.2.9(e)). □

SUCCINCT 3SAT plays a central role in the study of interactive protocols, and ultimately of approximability, see 13.4.14 and thereafter.

### A Special Case of First-Order Logic

FIRST-ORDER SAT, the problem of telling whether an expression in first-order logic has a model, is of course undecidable (Corollary 1 to Theorem 6.3). However, there are several interesting "syntactic classes" of expressions for which this problem is decidable. We shall examine one of them below (see the references in 20.2.11 for many others).

† *If* such an expression exists, that is. Most circuits fail to encode a legitimate Boolean expression for any one of a long array of possible reasons.

An expression in first-order logic is said to be a *Schönfinkel-Bernays* expression if (a) its alphabet has only relational and constant symbols, no function symbols, and *no equality;* and (b) it is of the form

$$\psi = \exists x_1 \ldots \exists x_k \forall y_1 \ldots \forall y_\ell \phi, \tag{1}$$

that is, it is in prenex form with a sequence of existential quantifiers followed by a sequence of universal ones. SCHÖNFINKEL-BERNAYS SAT is this problem: Given a Schönfinkel-Bernays expression as in equation (1), does it have a model?

**Theorem 20.3:** SCHÖNFINKEL-BERNAYS SAT is **NEXP**-complete.

***Proof:*** We shall first show that it is in **NEXP**. Consider a Schönfinkel-Bernays expression $\psi$ as in (1), and suppose that there are $m$ constants appearing in $\phi$.

**Lemma 20.2:** $\psi$ is satisfiable if and only if it has a model with $k+m$ or fewer elements.

***Proof:*** Suppose that $\psi$ has a model $M$ with a universe $U$. By the definition of satisfaction, there are elements $u_1, \ldots, u_k \in U$, not necessarily distinct, such that $M_{x_1=u_1,\ldots,x_k=u_k} \models \forall y_1 \ldots \forall y_\ell \phi$. Now let $U'$ be the set $\{u_1 \ldots, u_k\}$, plus all elements of $U$ that are images under $M$ of some constant appearing in $\phi$ (notice that the set $U'$ has at most $k+m$ elements); and let $M'$ be $M$ *restricted to* $U'$. That is, $M'$ has universe $U'$, and maps all constant symbols in the vocabulary to the same elements of $U'$ as $M$ (notice that the images of constants are by definition in $U'$). Finally, a $k$-tuple of elements of $U'$ is related by relation symbol $R$ under $M'$ if and only if it is under $M$.

We claim that $M' \models \psi$. The reason is that, since $M_{x_1=u_1,\ldots,x_k=u_k} \models \forall y_1 \ldots \forall y_\ell \phi$, then certainly $M'_{x_1=u_1,\ldots,x_k=u_k} \models \forall y_1 \ldots \forall y_\ell \phi$, because $M$ and $M'$ agree on all constant and relation symbols, and deleting elements from the universe makes a universal sentence even easier to satisfy. □

That SCHÖNFINKEL-BERNAYS SAT is in **NEXP** follows immediately from this lemma: To verify that an Schönfinkel-Bernays expression is satisfiable, all we have to do is guess a model with $|U| \leq k+m$ elements, and verify that this model satisfies $\psi$. Let $n$ be the length of the representation of expression $\psi$. Since $k+m \leq n$, and each relation and function symbol appearing in $\psi$ has arity at most $n$, the length of the description of the model is $\mathcal{O}(n^{2n})$; and testing satisfiability can be done in time $\mathcal{O}(n^q)$, where $q$ is the total number of quantifiers. We conclude that the problem is in **NEXP**.

To show completeness, consider a language $L$ decided by a nondeterministic Turing machine $N$, with two nondeterministic choices per step, in time $2^n$ (using Lemma 20.1). For each input $x$ we shall construct in logarithmic space a Schönfinkel-Bernays expression $R(x)$ such that $x \in L$ if and only if $R(x)$ has a model.

The construction is essentially the same as that in the proof of Fagin's theorem (Theorem 8.3), except in several ways simpler. Now we do not need



second-order existential quantifiers, because asking whether a model exists has the same effect. The arity of the relation symbols can now depend on $n$, and this simplifies matters tremendously.

We have $2n$ variables $x_1, \ldots, x_n, y_1, \ldots, y_n$. The whole expression $R(x)$ consists of the conjunction of all expressions described below, preceded by the universal quantification of all $2n$ variables. To simulate the values 0 and 1 for the variables, we have a *unary relation symbol* called $\mathbf{1}(\cdot)$. Intuitively, $\mathbf{1}(x_1)$ means that $x_1 = 1$, and $\neg\mathbf{1}(x_1)$ means that $x_1 = 0$ (we need this trick because we have no equality in our language).

For $k = 1, \ldots, n$ we have a $2k$-ary predicate $S_k(x_1, \ldots, x_k, y_1, \ldots, y_k)$ expressing that the $y_i$'s spell in binary the successor of the number spelled in binary by the $x_i$'s. We can define $S_k$ inductively exactly as in the proof of Theorem 8.3; and for $k = 1$, we have $S_1(x_1, y_1) \Leftrightarrow (\neg\mathbf{1}(x_1) \wedge \mathbf{1}(y_1))$.

For each symbol $\sigma$ that can appear on the computation table of $N$ we have a $2n$-ary relation symbol $T_\sigma(\mathbf{x}, \mathbf{y})$ (where $\mathbf{x}$ stands for $x_1, \ldots, x_n$ and $\mathbf{y}$ stands for $y_1, \ldots, y_n$), expressing that at the $\mathbf{x}$th step (where $\mathbf{x}$ is interpreted as a $n$-bit binary integer) the $\mathbf{y}$th symbol of $N$'s string is a $\sigma$. There are two $n$-ary relations $C_0$ and $C_1$, where $C_0(\mathbf{x})$ means that at the $\mathbf{x}$th step nondeterministic choice 0 was taken, and $C_1(\mathbf{x})$ means that choice 1 was followed. Again we require that at each step exactly one of the two happens. We also require that the first row of the table is filled properly by $x$ followed by blanks, and that the leftmost and rightmost columns contain only $\triangleright$'s and $\sqcup$'s, respectively. For each quintuple $(\alpha, \beta, \gamma, c, \sigma)$ such that, if three consecutive string symbols are $\alpha\beta\gamma$ and choice $c$ is made, then $\sigma$ appears in the next step in the place of $\beta$, we have an expression that guarantees this, exactly as in the proof of Theorem 8.3. Finally, we require that there be a "yes" in the last row. This concludes our sketch of the construction of $R(x)$. It is easy to argue that the conjuncts of $R(x)$ completely axiomatize the intended meaning of the relation symbols, and thus there is a model for $R(x)$ if and only if $x \in L$. □

One last reminder about **EXP** and **NEXP**-complete problems: Unlike all other completeness results in previous chapters in this book, these are problems that *we know* are not in **P**, and therefore *provably intractable* according to our criterion.

### And Beyond...

There is no reason to stop at **NEXP**: Beyond that one finds of course the *exponential hierarchy*, with its alternations of quantifiers. It is supposed that, as in the polynomial case, this hierarchy is indeed an infinite increasing sequence of classes; however, the same hierarchy starting from the class **NE** *does collapse* (see the references). Then we have exponential space **EXPSPACE** = **SPACE**$(2^{n^k})$; and even further we arrive at *doubly exponential time:* 2-**EXP** =

$\mathbf{TIME}(2^{2^{n^k}})$. Of course there is also 2-$\mathbf{NEXP} = \mathbf{NTIME}(2^{2^{n^k}})$. If 2-$\mathbf{EXP}$ and 2-$\mathbf{NEXP}$ were unequal, inequality would propagate down to $\mathbf{EXP} = \mathbf{NEXP}$ and finally $\mathbf{P} = \mathbf{NP}$. Further up is the class 3-$\mathbf{EXP} = \mathbf{TIME}(2^{2^{2^{n^k}}})$, and so on.

We have thus an *exponential hierarchy*—for a change, this time a true, provable hierarchy, since each of these classes properly includes the previous one by the time hierarchy theorem. The cumulative complexity class of this hierarchy is called *the class of elementary*† *languages*. That is, a language is elementary if it is in the class

$$\mathbf{TIME}(2^{2^{\cdot^{\cdot^{\cdot^{2^{n}}}}}}),$$

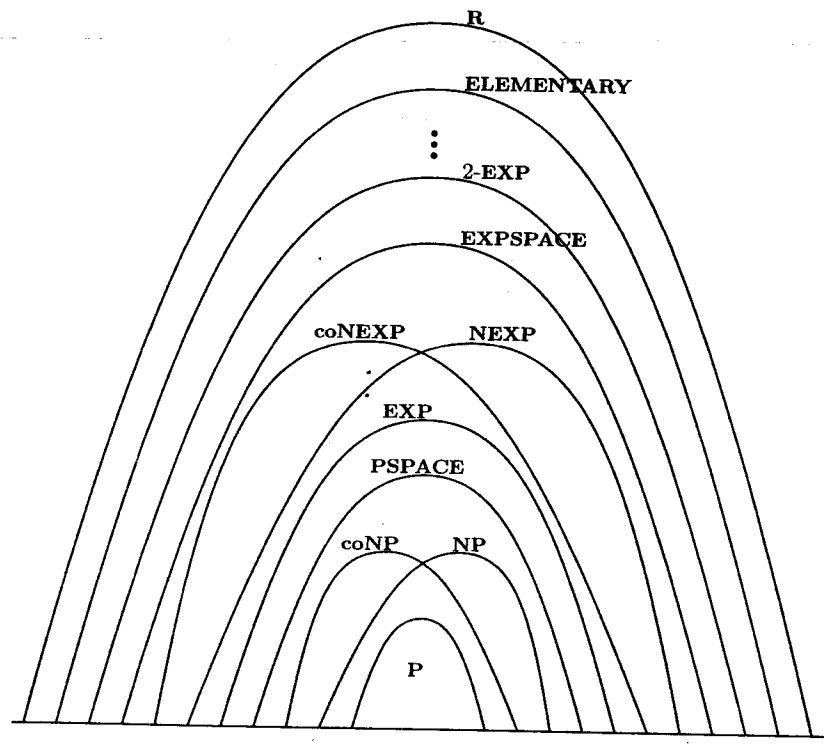
for some finite number of exponents. Notice that, in this context, nondeterminism, alternation, space bounds, or the $n^k$ in the final exponent are insignificant details... As it turns out, there are some fairly natural, decidable problems that are not even elementary (see Problem 20.2.13).

† The optimism in this term may seem a little overstated; the term was introduced in the context of undecidability.



## 20.2 NOTES, REFERENCES, AND PROBLEMS

**20.2.1 Class review:**




There is much confusion in the literature regarding the notation for exponential complexity classes. For example, **EXPTIME** has been sometimes used to denote our **E**, and similarly for nondeterministic classes.

**20.2.2** Oracles are known for which $\mathbf{P} \neq \mathbf{NP}$, and still $\mathbf{EXP} = \mathbf{NEXP}$, see

- M. I. Dekhtyar "On the relativization of deterministic and nondeterministic complexity classes," in *Mathematical Foundations of Computer Science*, pp. 255–259, Lecture Notes in Computer Science, vol. 45, Springer Verlag, Berlin, 1976.

Also, oracles are given in this paper that separate **EXP** from **PSPACE**. The relation of these classes with **E** and **NE** is also subject to all kinds of relativizations.

**20.2.3 Problem:** Show that, if $f(n)$ and $g(n) \geq n$ are proper complexity functions,
(a) $\mathbf{TIME}(f(n)) = \mathbf{NTIME}(f(n))$ implies $\mathbf{TIME}(g(f(n))) = \mathbf{NTIME}(g(f(n)))$;
(b) Similarly for space.

**20.2.4 Problem:** Show that, if $\mathbf{L} = \mathbf{P}$ then $\mathbf{PSPACE} = \mathbf{EXP}$.

**20.2.5 Problem (the nondeterministic space hierarchy):** (a) Show that $\mathbf{NSPACE}(n^3) \neq \mathbf{NSPACE}(n^4)$. (Use Problem 20.2.3(b) repeatedly, combined with Savitch's theorem and the space hierarchy theorem.)

(b) More generally, show that $\mathbf{NSPACE}(f(n)) \neq \mathbf{NSPACE}(f^{1+\epsilon}(n))$ for any proper function $f \geq \log n$ and $\epsilon > 0$.

**20.2.6** Theorem 20.1, as well as Problems 7.4.7 and 20.2.5, are from

- O. Ibarra "A note concerning nondeterministic tape complexities," *J.ACM, 19*, pp. 608–612, 1972,
- R. V. Book "Comparing complexity classes," *J.CSS, 9*, pp. 213–229, 1974, and
- R. V. Book "Translational lemmas, polynomial time, and $\log^j n$ space," *Theoretical Comp. Science 1*, pp. 215–226, 1976.

**20.2.7 Problem:** Show that $\mathbf{P}^{\mathbf{E}} = \mathbf{EXP}$.

**20.2.8 Problem:** (a) Show that $\mathbf{E} \neq \mathbf{NE}$ if and only if there are unary languages in $\mathbf{NP} - \mathbf{P}$. (Consider the unary version of any language in **E** (or **NE**); show it is in **P** (respectively, **NP**).) This result can be strengthened as follows:

(b) Show that $\mathbf{E} \neq \mathbf{NE}$ if and only if there are sparse languages in $\mathbf{NP} - \mathbf{P}$. (This is from

- J. Hartmanis, V. Sewelson, and N. Immerman "Sparse sets in NP–P: EXPTIME vs. NEXPTIME," *Information and Control, 65*, pp. 158–181, 1985.)

That the **NE** hierarchy collapses was shown in

- L. Hemachandra "The strong exponential hierarchy collapses" *J.CSS 39*, 3, pp. 299–322, 1989.

**20.2.9** Succinctness tends to increase the complexity of a problem by an exponential; Theorem 20.2 is only one example of the possibilities:

(a) Define SUCCINCT KNAPSACK and show that it is **NEXP**-complete.

(b) Show that SUCCINCT REACHABILITY is **PSPACE**-complete. Repeat for the case in which the graph is known to be a tree (recall Problem 16.4.4).

(c) Define SUCCINCT ODD MAX FLOW and show that it is **EXP**-complete.

(d) Define NON-EMPTINESS to be the following problem: Given a graph, does it have an edge? Show that SUCCINCT NON-EMPTINESS is **NP**-complete.

(e) Show that SUCCINCT CIRCUIT VALUE is complete for alternating polynomial space.

The complexity of succinct versions of graph-theoretic problems was studied in

- H. Galperin and A. Wigderson "Succinct representations of graphs," *Information and Control, 56*, pp. 183–198, 1983,



where the exponential increase in the complexity of these problems was first observed. The general reduction technique in the proof of Theorem 20.2 is from

- C. H. Papadimitriou and M. Yannakakis "A note on succinct representations of graphs," *Information and Control, 71*, pp. 181–185, 1986.

For a much more detailed treatment of the subject see

- J. L. Balcázar, A. Lozano, and J. Torán "The complexity of algorithmic problems in succinct instances," in *Computer Science*, edited by R. Baeza-Yates and U. Manber, Plenum, New York, 1992.

**20.2.10 Problem:** We are given a set of square tile types $T = \{t_0, \ldots, t_k\}$, together with two relations $H, V \subseteq T \times T$ (the *horizontal* and *vertical compatibility relations*, respectively). We are also given an integer $n$ in binary. An $n \times n$ *tiling* is a function $f : \{1, \ldots, n\} \times \{1, \ldots, n\} \mapsto T$ such that (a) $f(1,1) = t_0$, and (b) for all $i, j$ $(f(i,j), f(i+1,j)) \in H$, and $(f(i,j), f(i,j+1)) \in V$. TILING is the problem of telling, given $T$, $H$, $V$, and $n$, whether an $n \times n$ tiling exists.

(a) Show that TILING is **NEXP**-complete.

(b) Show that TILING becomes **NP**-complete if $n$ is given in unary (this is the succinctness phenomenon, backwards).

(c) Show that it is undecidable to tell, given $T$, $H$, and $V$, whether an $n \times n$ tiling exists *for all* $n > 0$.

**20.2.11 Decidable fragments of first-order logic.** Besides the Schönfinkel-Bernays fragment of first-order logic shown **NEXP**-complete in Theorem 20.3, satisfiability can also be decided for function-free expressions with the following kinds of quantifier sequences:

(a) Quantifiers of the form $\exists^*\forall\exists^*$ (that is, only one universal quantifier). This is the *Ackermann class*, and is **EXP**-complete.

(b) Quantifiers of the form $\exists^*\forall\forall\exists^*$ (that is, only two consecutive universal quantifiers). This is the *Gödel class*, and its satisfiability problem is **NEXP**-complete.

As it turns out, *the validity problem for all other quantifier sequences is undecidable.* Yet another decidable case is this:

(c) Arbitrary expressions in vocabularies with *only unary relation symbols.* This is the *monadic* case, and its satisfiability problem is **NEXP**-complete.

Research on these decidability results was a major part of Hilbert's program (see the references in Chapter 6), and generally predated the undecidability of first-order logic. For decidability, undecidability, and complexity results concerning segments of first-order logic see, respectively,

- B. S. Dreben and W. D. Goldfarb *The Decision Problem: Solvable Cases of Quantification Formulas*, Addison-Wesley, Reading, Massachusetts, 1979.
- H. R. Lewis *Unsolvable Classes of Quantification Formulas*, Addison-Wesley, Reading, Massachusetts, 1979.

- H. R. Lewis "Complexity results for classes of quantification formulas," in *J.CSS 21*, pp. 317–353, 1980.

**20.2.12 The theory of reals.** We noted in Chapter 9 that it is **NP**-complete to tell whether a set of linear inequalities over the integers is satisfiable, while the same problem for reals is in **P** (recall INTEGER and LINEAR PROGRAMMING, 9.5.34). It is amusing to notice that the complexity of problems relating to the integers and reals exhibit a similar behavior in a much more general setting: While it is undecidable whether a first-order expression over the vocabulary $0, 1, +, \times, <$ is a true property of **N** (recall Coroallary 1 to Theorem 6.3), *it is decidable whether such an expressions is true a property of $\Re$, the real numbers.* (To see the significant difference between the two theories, consider

$$\forall x \forall y \exists z [x \geq y \vee (x < z \wedge z < y)].)$$

Let THEORY OF REALS be the set of all first-order sentences $\phi$ over this vocabulary such that $\Re \models \phi$. THEORY OF REALS WITH ADDITION is the subset with no occurrence of multiplication.

To show that THEORY OF REALS WITH ADDITION is decidable, we use a general technique that works in many other cases: *Elimination of quantifiers.* For any expression in prenex form $Q_1 x_1 \ldots Q_n x_n \phi(x_1, \ldots, x_n)$, where $\phi$ is quantifier-free, we show how to convert it to an equivalent quantifier-free expression. By induction, it suffices to show how we can transform the above expression to an equivalent expression $Q_1 x_1 \ldots Q_{n-1} x_{n-1} \phi'(x_1, \ldots, x_{n-1})$, with $\phi'$ quantifier-free. In fact, assume $Q_n$ is $\forall$ (otherwise rewrite the expression in terms of $\forall x_n$).

(a) Show that $\phi$ is a Boolean combination of $k$ atomic expressions of the forms $x_n \triangleright \ell_i(x_1, \ldots, x_{n-1})$, where $\triangleright \in \{=, <, >\}$ and the $\ell_i$'s are *linear functions with rational coefficients.*

(b) Show that $\phi'$ above can be taken to be $\bigvee_{t \in T} \phi[x_1 \leftarrow t]$, where $T$ is the set of all $n^2$ terms $\frac{1}{2}(\ell_i(x_1, \ldots, x_{n-1}) + \ell_j(x_1, \ldots, x_{n-1}))$ for all $i, j$.

(c) Conclude that THEORY OF REALS WITH ADDITION is in 2-**EXP**. Can you improve this to **EXPSPACE**?

For a weak lower bound, we can show that every problem in **NEXP** reduces to THEORY OF REALS WITH ADDITION. To this end, we construct for each $n \geq 0$ expressions (1) $\mu_n(x, y, z)$, (2) $\xi_n(x, y, z)$, and (3) $\beta_n(x, y)$, with length $\mathcal{O}(n)$, with the indicated free variables. These expressions have the following properties: $\Re$ satisfies them, with real numbers $a, b, c$ replacing variables $x, y, z$, if and only if, respectively: (1) $a$ is an integer between zero and $2^{2^n}$, and $a \cdot b = c$; (2) $a$ and $c$ are integers between zero and $2^{2^n}$, and $b^a = c$; and (3) $a$ and $b$ are integers $0 \leq a \leq 2^{(2^n+1)^2} + 1)$, $0 \leq b \leq 2^n$, and the $b$th bit of $a$ is a one.

(d) Show how to construct these expressions, in logarithmic space in $n$, by induction on $n$. (To avoid multiple uses of $\mu_i$ etc. in the definition of $\mu_{i+1}$ you will have to use the trick in the proof of Theorem 19.1.)



(e) Use these expressions to encode any nondeterministic exponential time computation into a sentence, such that the sentence is in THEORY OF REALS WITH ADDITION if and only if the computation is successful (this is another exponentially dilated version of Cook's theorem).

The complexity of THEORY OF REALS WITH ADDITION can be pinpointed precisely to a complexity class, but not one that we have defined in this book (at least, as far as we know...): Alternating Turing machines with exponential time (so far we would have all of exponential space) *but with only $n$ alternations per computation.* This result is from

- L. Berman "The complexity of logical theories," *Theor. Comp. Science 11*, pp. 71–78, 1980.

Interestingly, THEORY OF REALS (the full language, including multiplication) is also decidable. This is done also by elimination of quantifiers, but now of a much more involved kind—for example, eliminating quantifiers from $\exists x a \cdot x \cdot x + b \cdot x + c = 0$ should produce $b \cdot b \geq 4 \cdot a \cdot c$. This is a classical result due to Alfred Tarski; interestingly, it is still open whether the problem remains decidable if we also allow exponentiation.

If we weaken *number theory* by not allowing exponentiation, we know that the problem is still undecidable (recall the references in Chapter 6). If however we also remove multiplication, we arrive at a fragment of number theory known as *Presburger arithmetic.* This theory is decidable also by elimination of quantifiers, and its complexity is also high, see for example

- M. J. Fischer and M. O. Rabin "Super-exponential complexity of Presburger arithmetic," *Complexity of Computation* (R. M. Karp, ed.), SIAM-AMS Symp. in Applied Mathematics, 1974.

**20.2.13 Regular expresion equivalence.** The class of *regular expressions* is a language over the alphabet $\{0, 1, \emptyset, \cdot, \cup, ^*\}$, defined as follows: First, the unit-length strings 1, 0, and $\emptyset$ are regular expressions. Next, if $\rho$ and $\rho'$ are regular expressions, then so are $\rho \cdot \rho'$, $\rho \cup \rho'$, and $\rho^*$. The *semantics* of regular expressions is simple: The meaning of regular expression $\rho$ is a *language* $L(\rho) \subseteq \{0, 1\}^*$, defined inductively as follows: First, $L(0) = \{0\}$, $L(1) = \{1\}$, and $L(\emptyset) = \{\}$. Then, $L(\rho \cdot \rho') = L(\rho)L(\rho') = \{xy : x \in L(\rho), y \in L(\rho')\}$, $L(\rho \cup \rho') = L(\rho) \cup L(\rho')$, and $L(\rho^*) = L(\rho)^*$.

(a) Describe $L((0 \cup 1^*)^*)$ and $L((10 \cup 1)^* \cup (11 \cup 0)^*)$. Can you design *finite-state automata*, perhaps nondeterministic (recall Problems 2.8.11 and 2.8.18) that decide these languages?

In fact, a language can be decided by a finite automaton if and only if it is the meaning of a regular expression (this is why we called such languages regular in Problem 2.8.11). One direction is easy:

(b) If $\rho$ is a regular expression, show how to design a nondeterministic finite-state automaton that decides $L(\rho)$. (Obviously, by induction on the structure of $\rho$.) Can you prove the other direction?

Two regular expressions $\rho$ and $\rho'$ are *equivalent* if $L(\rho) = L(\rho')$. Deciding whether two regular expressions are equivalent is an important computational problem, whose

variants are all over the upper complexity spectrum. These variants were explored in a seminal paper

- L. J. Stockmeyer and A. R. Meyer "Word problems requiring exponential time," *Proc. 5th ACM Symp. on the Theory of Computing*, pp. 1–9, 1973.

(c) Show that the problem of deciding whether two regular expressions are equivalent is complete for **PSPACE** *even if one of the expressions is* $\{0,1\}^*$. (To show that it is in **PSPACE** use nondeterminism. To show completeness, express as a regular expression the set of all strings that *do not* encode an accepting in-place computation of machine $M$ on input $x$.)

(d) Call a regular expression $^*$*-free* if it has no occurrences of $^*$, the Kleene star. Show that deciding whether two $^*$-free regular expressions are equivalent is **coNP**-complete. (Membership in **coNP** is not hard. To prove completeness, start with an instance of 3SAT and write a $^*$-free regular expression for the set of truth assignments that *fail* to satisfy it.)

(e) Suppose now that we allow the abbreviation $^2$ (*squaring*) in our regular expressions, where $L(\rho^2) = L(\rho)L(\rho)$. We still do not allow $^*$. Show that the equivalence problem is now **coNEXP**-complete. (With $^2$ we can express nonsatisfying truth assignments of an exponentially long expression, as long as its clauses have a certain regularity. Notice that this is another instance of succinctness increasing the complexity by an exponential.)

(f) Suppose next that both $^2$ and $^*$ are allowed. Show that the problem is now complete for *exponential space!* (The succinctness phenomenon again, compare with (c).)

(g) Finally, if we also allow the symbol $\neg$ (semantics: $L(\neg\rho) = \{0,1\}^* - L(\rho)$) then *the equivalence problem is not even elementary.* (Each occurrence of $\neg$ can cause the complexity to be raised by another exponential, intuitively because it may require the conversion of a nondeterministic finite-state automaton to a nondeterministic one, recall Problem 2.8.18 .)

**20.2.14 A panorama of complexity classes.** There is an amusing and instructive way of looking at all diverse complexity classes discussed in this book from a unified point of view. We have one model of computation: The nondeterministic, polynomial time bounded Turing machine, standardized so that it has precisely two choices at each step (arbitrarily ordered as the first and second choice) and halts after precisely $p(n)$ steps on inputs of length $n$, where $p$ is bounded by a polynomial. Such a machine $N$ operating on input $x$ produces a computation tree with $2^{p(|x|)}$ leaves, each leaf labeled with a "yes" or "no". Now, since choices are ordered, these leaves are also ordered, and therefore the computation of $N$ on input $x$ can be considered as a string in $\{0,1\}^{2^{p(|x|)}}$, disregarding for a moment the distinction between "yes"-"no" and 1-0. We denote this string as $N(x)$.

A language $L \subseteq \{0,1\}^*$ will be called a *leaf language*. Let $A$ and $R$ be two *disjoint* leaf languages (the *accepting* and *rejecting* leaf language, respectively). Now, any two such languages define a *complexity class:* Let $C[A,R]$ be the class of all languages $L$ such that there is a (standardized) nondeterministic Turing machine $N$ with the following property: $x \in L$ if and only if $N(x) \in A$, and $x \notin L$ if and only if $N(x) \in R$.



(a) Show that $\mathbf{P} = C[A,R]$ where $A = 1^*$ and $R = 0^*$. Show that $\mathbf{NP} = C[A,R]$ where $A = \{0,1\}^*1\{0,1\}^*$ and $R = 0^*$. Show that $\mathbf{RP} = C[A,R]$ where $A = \{x \in \{0,1\}^* :$ $x$ has more 1's than 0's$\}$ and $R = 0^*$.

(b) Find appropriate leaf languages $A$ and $R$ such that $C[A,R]$ is: **coNP**, **PP**, **BPP**, **ZPP**, **UP**, $\oplus\mathbf{P}$, $\mathbf{NP} \cap \mathbf{coNP}$, $\mathbf{NP} \cup \mathbf{coNP}$, $\mathbf{NP} \cup \mathbf{BPP}$.

(c) Repeat for $\Sigma_2\mathbf{P}, \Sigma_j\mathbf{P}, \mathbf{PSPACE}$.

(d) Consider the leaf language $A$ which consists of all strings $x$ with the following property: If $x$ is subdivided into disjoint substrings of length $2^k$, where $k = \lceil \log \log x \rceil$, and if these $\frac{|x|}{2^k}$ strings are considered as binary integers, then the largest such integer is odd. Show that the class $C[A,\overline{A}]$ is $\Delta_2\mathbf{P}$. (Recall Theorem 17.5 and Problem 17.3.6.)

(e) Show that all leaf languages considered in (a) through (d) above are in **NL**. Show that, if $A, R \in \mathbf{NL}$, then $C[A,R] \subseteq \mathbf{PSPACE}$.

(f) Show that if $A$ is an **NL**-complete leaf language, then $C[A,\overline{A}] = \mathbf{PSPACE}$

(g) Find leaf languages $A$ and $R$ such that $C[A,R] = \mathbf{EXP}$. Repeat for **NEXP**.

(h) Which of the pairs of leaf languages $A$ and $R$ considered in (a) through (d) above are complementary, that is, $A \cup R = \{0,1\}^*$? Which can be redefined to be made complementary? (For example, the pair for **P** in (a) is not complementary, but another complementary pair exists.)

Notice the close correlation of the classes whose definitions are via complementary pairs with the classes we have been informally calling *syntactic*, as opposed to *semantic*. In fact, a perfectly reasonable definition of "syntactic class" would be "any class of the form $C[A,\overline{A}]$."

(j) Define an appropriate class of functions from leaf languages to leaf languages such that the following is true: If $f$ is a function in this class, and $A$ and $R$ are, as usual, disjoint leaf languages, then $f(A), f(R)$ are also disjoint leaf languages, and $C[A,R] \subseteq C[f(A), f(R)]$

As it turns out, a formalism closely related to the leaf languages above can be used to systematize proofs of oracle results, see

- D. P. Bovet, P. Crescenzi, and R. Silvestri "A uniform approach to define complexity classes" *Theor. Comp. Sci, 104*, pp. 263–283, 1992.

**20.2.15 Networks of queues.** Suppose that we are given a *network of queues*, that is, a finite set $V = \{1,\ldots,n\}$ of queues and a set $T$ of *customer types*, where a type $t_i = (P_i, a_i, S_i, w_i)$ is a set $P_i \subseteq V^*$ of *paths* (sequences of queues that are acceptable ways of servicing a customer of this type), an *inter-arrival time distribution* $a_i$ (how often customers of this sort arrive in the system), for each queue $j \in V$ a *service time distribution* $S_{ij}$ (how long such a customer is going to spend in queue $j$), and a *weight*

$w_i$ (how important for the system is this class of customers). The distributions are discrete ones, and are given explicitly in terms of a set of value-probability pair. The problem is to control this system —basically, to decide how to proceed each time a customer arrives or finishes service at a queue—so as to minimize the weighted sum of the expected total waiting times of the customers.

This is a well-known, important, and fantastically hard problem —for example the case of *two* queues ($n = 2$) is already a notoriously difficult problem.

**Problem:** Formulate the problem precisely, and show that it is **EXP**-complete (use alternating polynomial space).

**20.2.16 Interactive proofs and exponential time.** Recall the interactive proof systems between Alice and Bob defined in Section 12.2, and shown in Theorem 19.8 to coincide with **PSPACE**. Suppose that we extend this idea to *multiple provers*. That is, the protocol is now between Bob, who as always has polynomial computing powers and randomization, and *several provers* –call them Alice, Amy, Ann, April, and so on—each with exponential powers, and each very interested in convincing Bob that a string $x$ is actually in language $L$. Bob can now address each of his questions to any one of the provers, and the prover must answer. In fact, for each input $x$ Bob may interact with a number of provers that depends polynomially on $|x|$. Again if $x \in L$ we want Bob to accept $x$ with probability one; if $x \notin L$, then for any possible set of provers we want Bob to accept with probability less than $2^{-|x|}$.

The key feature which makes the situation interesting is that *the provers cannot communicate with each other during the protocol.* If they could, the situation would be identical to the one with a single prover (a gang of conspiring provers behaves exactly like one prover). But the inability of provers to communicate makes it harder for them to fool Bob, and, as we shall see, possibly allows for more interesting and powerful languages to be thus decided.

If a language $L$ can be decided as above, we say that *it has a multiprover interactive proof system;* we write $L \in$ **MIP**. We say that $L$ has an *oracle proof system* if the following holds: There is a randomized oracle Turing machine $M^?$ such that, if $x \in L$ then there is an oracle $A$ such that $M^A(x) =$ "yes" with probability one; and if $x \notin L$, then for all oracles $B$, $M^B(x) =$ "yes" with probability less than $2^{-|x|}$.

(a) Show that the following are equivalent:
(1) $L \in$ **MIP**.
(2) $L$ has an interactive proof system with *two* provers.
(3) $L$ has an oracle proof system.

(That (2) implies (1) is, of course, trivial. To show that (1) implies (3), think that the provers agree beforehand (as they have to, because of the lack of communication) on all answers each of them will give to any question by Bob; express this protocol as an oracle machine. To show that (3) implies (2), Bob can simulate $M^A$ by asking one of the provers the oracle queries, and at the end asking the second prover a randomly selected query among those asked of the first—to make sure that the first prover is reciting some oracle, and is not basing her answers on the interaction. Repeat enough times. This argument is from



- L. Fortnow, J. Rompel, and M. Sipser "On the power of multiprover interactive protocols," *Proc. 3rd Conference on Structure in Complexity Theory,* pp. 156–161, 1988.)

(b) Based on (a), show that **MIP** $\subseteq$ **NEXP**. (Use (3).)

Surprisingly, *it can be shown that these two classes coincide.* Once more, the power of interactive protocols achieves its limits (compare with Shamir's theorem, Theorem 19.8). This was proved in

- L. Babai, L. Fortnow, and C. Lund "Nondeterministic exponential time has two-prover interactive protocols," *Proc. 31st IEEE Symp. on the Foundations of Computer Science,* pp. 16–25, 1990; also, *Computational Complexity 1,* pp. 3–40, 1991

by extending the "arithmetization" methodology used in the proof of Shamir's theorem to devise an oracle proof system for a version of SUCCINCT 3SAT (Corollary 1 of Theorem 20.2). The proof now is much more sophisticated. An instance of SUCCINCT 3SAT, together with an alleged satisfying truth assignment provided by the oracle, is converted into a long summation, very much as in the proof of Shamir's theorem. If the truth assignment provided by the oracle is a *multilinear* function (a polynomial of degree one in each variable), then a modification of the proof of Shamir's theorem works. Finally, the oracle has to be tested *for multilinearity* —and this turns out to be the heart of the proof.

**20.2.17 NEXP and approximability.** The next important step in the array of fascinating developments which led to the proof of Theorem 13.13 was the observation that probabilistic interactive proofs are relevant to the approximability of optimization problems; this was first pointed out in

- U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy "Approximating clique is almost NP-complete," *Proc. 32nd IEEE Symp. on the Foundations of Computer Science,* pp. 2–12, 1991.

The idea is simple: Suppose that $L \in$ **NEXP**; by the previous problem, we can assume that $L$ has an oracle proof system $M^?$. Let $V(x)$ now be the set of all possible accepting computations of $M^?$ on input $x$—they are exponentially many in $|x|$. Define the following set of edges: $[c, c'] \in E(x)$, where $c, c'$ are computations in $V(x)$, if and only if there is an oracle $A$ that can cause $M^?$ to follow both $c$ and $c'$ (in other words, if $c$ and $c'$ are "compatible"). It turns out that, since $M^?$ is an oracle proof system for $L$, the maximum clique of the graph $(V(x), E(x))$ is either very small (if $x \notin L$) or very large (in the case $x \in L$).

**Problem:** Conclude that if the approximation threshold of CLIQUE (or INDEPENDENT SET, for that matter) is strictly less than one, then **EXP** = **NEXP** (compare with Corollary 2 of Theorem 13.13).

Much of the present chapter has been about ideas and techniques from the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem "scaled up" to exponential time. Going from the above result to

- S. Arora, S. Safra "Probabilistic checking of proofs," *Proc. 33rd IEEE Symp. on the Foundations of Computer Science*, pp. 2-13, 1992,

and ultimately to

- S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy "Proof verification and hardness of approximation problems," *Proc. 33rd IEEE Symp. on the Foundations of Computer Science*, pp. 14–23, 1992

and Theorems 13.12 and 13.3 involved clever arguments for efficiently "scaling down" to the polynomial domain the techniques of arithmetization and multilinear testing; in fact, this scaling-down effort had already started in the paper by Feige et al. referenced above. For a comprehensive account of these techniques see

- M. Sudan *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*, PhD dissertation, Univ. of California Berkeley, 1992.

# Index