Chapter 4

Diagonalization and Relativization

4.1 Hierarchy Theorems

We say that a TM M *recognizes* a language L if $L = L(M)$ (i.e., M halts and accepts all $x \in L$, and M may reject or not halt on $x \notin L$). We say that a language L is *recursively enumerable* if it is recognized by some M. Let RE be the class of recursively enumerable languages.

A TM M is called a *decider* if it halts on every input. We say that a TM M *decides* a language L if $L = L(M)$ & M is a decider (i.e., M recognizes L, and, furthermore, M halts on all inputs—in the accepting state if $x \in L$, and in the rejecting state if $x \notin L$). A language L is called *decidable* (or *recursive*) if there exists a decider M such that $L = L(M)$. Let Rec be the class of recursive (i.e., decidable) languages[1].

It is well known that the language

$$\mathsf{A}_{\mathrm{TM}} = \{\langle M, x\rangle |\ M \text{ accepts } x\ \}$$

is recursively enumerable, but not recursive. A_{TM} is clearly in RE; we use

[1]Note that all complexity classes are assumed to be contained in Rec, i.e., if $\mathcal{C}$ is a complexity class, then $\mathcal{C} \subseteq \mathsf{Rec}$. For time-bounded computations this follows by definition; for space-bounded computations, we can always modify a space-bounded TM M to be a decider M'. This is so because we can always add a counter which checks that the number of moves does not exceed the number of possible configurations, and halt and reject when it does. Such a counter need not be longer than the available space.

the *diagonal argument* to show that it is not recursive[2].

Theorem 4.1.1 A_{TM} *is not recursive.*

PROOF: Suppose $M_{\mathsf{A}_{\mathrm{TM}}}$ decides A_{TM}. Define D, the "diagonal" machine, as follows: on input $\langle M \rangle$, D first simulates $M_{\mathsf{A}_{\mathrm{TM}}}$ on $\langle M, \langle M \rangle \rangle$, until $M_{\mathsf{A}_{\mathrm{TM}}}$ is about to halt ($M_{\mathsf{A}_{\mathrm{TM}}}$ is a decider, so by assumption it always halts). If $M_{\mathsf{A}_{\mathrm{TM}}}$ is about to accept, D rejects. If $M_{\mathsf{A}_{\mathrm{TM}}}$ is about to reject, then D accepts.

Consider $D(\langle D \rangle)$. If $D(\langle D \rangle)$ rejects, then $M_{\mathsf{A}_{\mathrm{TM}}}$ must accept $\langle D, \langle D \rangle \rangle$, by definition of D, so $D(\langle D \rangle)$ must accept by definition of $M_{\mathsf{A}_{\mathrm{TM}}}$. If $D(\langle D \rangle)$ accepts, then $M_{\mathsf{A}_{\mathrm{TM}}}$ must reject, by definition of D, so by definition of $M_{\mathsf{A}_{\mathrm{TM}}}$ $D(\langle D \rangle)$ rejects. Thus, $D(\langle D \rangle)$ accepts iff it does not. Contradiction. □

We shall now use a slightly modified version of this diagonal argument to show some strong separations between complexity classes; these results go under the name of Hierarchy Theorems. Recall the definition of "little-oh" (see page v), and of space constructible functions (see page 27).

Theorem 4.1.2 (Space Hierarchy) *Given any space constructible function f, a language A exists that is decidable in space $O(f(n))$ but not in $o(f(n))$.*

PROOF: Let D be a deterministic TM which on input $\langle M \rangle \#^m$, $m \geq 0$, and M is a single-tape TM, simulates $M(\langle M \rangle)$ within space bound $f(n)$, $n = |\langle M \rangle \#^m| = |\langle M \rangle| + m$. The suffix of "#" works as padding; its purpose will be clear later. D works as follows.

- If M halts within the alloted space $f(n)$, then D accepts iff M rejects (i.e., D "does the opposite").
- If M does not halt (but stays throughout in the alloted space), D rejects. In order to accomplish this, D keeps a counter of the number of moves of M, and it rejects when the counter reaches $2^{cf(n)}$, for

[2]Note that it follows directly from the diagonal argument that there are undecidable languages (in fact that there are not recursively enumerable languages): the set of all TMs has the same cardinality as $\mathbb{N}$, and the set of all languages has the same cardinality as $\mathcal{P}(\mathbb{N})$, the power set of $\mathbb{N}$, and by Cantor's diagonal arguement $\mathbb{N} < \mathcal{P}(\mathbb{N})$, so there are languages which do not have a TM. However, theorem 4.1.1 says more than that; it gives a concrete example of an undecidable language, i.e., A_{TM}, which is recursively enumerable, and in fact complete for the class of recursively enumerable languages.

some constant c depending on M. The number $2^{cf(n)}$ is the number of possible configurations of M on inputs of lenght n; this number requires $cf(n)$ bits to encode.

- If M requires more than $f(n)$ squares of space, then D rejects directly. Note that D keeps track of the space by placing a special symbol ♣ on the square number $f(n)$ of each work-tape, and if the simulation ever wants to move right of this symbol, the computation ends and D rejects. Since f is space constructible, computing which is the $f(n)$-th square can be done in space $c'f(n)$, for some constant c'.

Clearly, D runs in space $\max\{1, c, c'\} \cdot f(n)$, i.e., in space $O(f(n))$.

We now argue that the language $L(D)$ cannot be decided in space $o(f(n))$. Suppose that N runs in space $g(n) \in o(f(n))$ (if N is a k-tape TM we can convert it into a single-tape TM which runs in space $kg(n)$, which is still in $o(f(n))$), so $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$, and so for any $\varepsilon > 0$ there exists an n_0 such that for all $n \geq n_0$, $g(n) < \varepsilon f(n)$. Thus, given $\langle N\rangle \#^{n_0}$ as input, D will simulate $N(\langle N\rangle)$ within space

$$g(|\langle N\rangle| + n_0) < \varepsilon f(|\langle N\rangle| + n_0),$$

and so, by selecting $\varepsilon < 1$, we ensure that D and N differ on $\langle N\rangle \#^{n_0}$, and thus $L(N) \neq L$. Note that $|\langle N\rangle|$ could be small, so the padding $\#^{n_0}$ ensures that the input is of sufficient length.

Finally, note that there is a certain (small) overhead associated with the simulation. We can compensate for this by selecting ε suitably small. □

Corollary 4.1.3 *For $\varepsilon_1 < \varepsilon_2$ in $\mathbb{Q} \cap [0, \infty)$, $\mathsf{SPACE}(n^{\varepsilon_1}) \subset \mathsf{SPACE}(n^{\varepsilon_2})$.*

Corollary 4.1.4 $\mathsf{NL} \subset \mathsf{PSPACE}$.

Proof: From Savitch's theorem (theorem 3.1.4, pg. 29) we know that NL is contained in $\mathsf{SPACE}(\log^2(n))$, and from the Space Hierarchy theorem we know that $\mathsf{SPACE}(\log^2(n)) \subset \mathsf{SPACE}(n)$. □

Corollary 4.1.5 QSat $\notin \mathsf{NL}$.

Proof: By corollary 4.1.4, $\mathsf{NL} \subset \mathsf{PSPACE}$, and QSat is complete for PSPACE by theorem 3.1.7. □

Corollary 4.1.6 $\mathsf{PSPACE} \subset \mathsf{EXPSPACE}$.

PROOF: For all constant k, $\mathsf{SPACE}(n^k)$ is contained in $\mathsf{SPACE}(n^{\log(n)})$, while $\mathsf{SPACE}(n^{\log(n)}) \subset \mathsf{SPACE}(2^n)$. □

We can now give an example of a *provably intractable* language. Recall that regular expressions are built up from $\emptyset, \varepsilon, a \in \Sigma$, by using the regular operations $\cup, \cdot, *$ (union, concatenation, and Kleene's star, respectively). Formally, $\emptyset, \varepsilon, a \in \Sigma$ are regular expressions, and if R, S are regular expressions, then so are $R \cup S, R \cdot S, R^*, (R)$. The language of a regular expression R, denoted $L(R)$, is defined inductively as follows:

$$\begin{aligned} &L(\emptyset) = \emptyset, L(\varepsilon) = \{\varepsilon\}, L(a) = \{a\} \\ &L(R \cup S) = L(R) \cup L(S) \\ &L(R \cdot S) = \{xy | x \in L(R) \ \ \& \ \ y \in L(S)\} \\ &L(R^*) = \{x_1 x_2 \ldots x_n | n \geq 0, x_i \in L(R)\} \end{aligned}$$

The language

$$\mathsf{EQ}_{\mathrm{rex}} = \{\langle R, S\rangle |\ R, S \text{ are regular expressions and } L(R) = L(S)\}$$

is in PSPACE.

If we also introduce the exponentiation operator $R \uparrow k = R \cdot R \cdot \cdots \cdot R$ (k-times), and let $\mathsf{EQ}_{\mathrm{rex}\uparrow}$ be the corresponding language, then we have the following theorem.

Theorem 4.1.7 $\mathsf{EQ}_{\mathrm{rex}\uparrow}$ *is* $\mathsf{EXPSPACE}$*-complete.*

Corollary 4.1.8 $\mathsf{EQ}_{\mathrm{rex}\uparrow}$ *is intractable.*

PROOF: If it were in P, then it would certainly be in PSPACE, but by completeness it would follow that $\mathsf{EXPSPACE} \subseteq \mathsf{PSPACE}$, contradicting the separation we obtained from the Space Hierarchy Theorem. □

A function $t : \mathbb{N} \longrightarrow \mathbb{N}$, where $t(n)$ is at least $O(n \log(n))$ is *time constructible* if the function that maps any $x \in \{0,1\}^n$ to the binary representation of $t(n)$ is computable in time $O(t(n))$.

Theorem 4.1.9 (Time Hierarchy) *Given a time constructible function t, there exists a language A that is decidable in time $O(t(n))$ but not in time $o(t(n)/\log(t(n)))$.*

Exercise 4.1.10 *Prove the Time Hierarchy Theorem.*

Corollary 4.1.11 $\mathsf{P} \subset \mathsf{EXPTIME}$.

We can also prove Hierarchy Theorems for nondeterministic classes. This is a little bit more tricky since it is not clear how to "flip" the result of the computation efficiently in the nondeterministic case. We demonstrate the main idea with concrete nondeterministic time classes, and then leave the general result as an exercise.

Theorem 4.1.12 $\mathsf{NTIME}(n) \subset \mathsf{NTIME}(n^{1.5})$.

PROOF: Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be defined as follows:

$$f(1) = 2$$
$$f(i+1) = 2^{f(i)^{1.2}}.$$

Now D is going to flip the answer of M_i on *some* input in the set of strings $\{1^n | f(i) < n \leq f(i+1)\}$.

First, assume once again that we have an enumeration $M_1, M_2, M_3, \ldots$ of all TMs such that each TM M appears infinitely often. Thus, given any M, and given any i_0, we can always find an $i \geq i_0$ such that $M = M_i$. Assume also that the degree of non-determinism is bounded *for the entire list*, so it is, say, 2.

Define the diagonal machine D as follows: on input $x = 1^n$ (if $x \notin \{1\}^*$, reject outright), compute an i such that $f(i) < n \leq f(i+1)$ (note that this must, and can, be done in time $O(n^{1.5})$). The machine D now considers two cases.

Case 1. $f(i) < n < f(i+1)$ (i.e., n is strictly in between). Then, D simulates M_i on input 1^{n+1}, using nondeterminism, in time $n^{1.1}$ (if M_i does not halt within this time, D simply accepts) and outputs the same answer.

Case 2. $n = f(i+1)$, then D accepts 1^n iff M_i rejects $1^{f(i)+1}$ in time $(f(i)+1)^{1.1}$. For D to know that M_i rejects, it must go through $2^{(f(i)+1)^{1.1}}$-many branches of M_i on $1^{f(i)+1}$. But this is doable, since the input size is $n = f(i+1) = 2^{f(i)^{1.2}}$.

Thus D is a nondeterministic machine that runs in time $O(n^{1.5})$. We want to show now that $L(D) \notin \mathsf{NTIME}(n)$. Suppose that it is, so we can find an i large enough so that M_i decides $L(D)$ in time $O(n)$ and on inputs of length $n \geq f(i)$, M_i can be simulated in less than $n^{1.1}$ many steps.

We know that for all $f(i) < n < f(i+1)$, $D(1^n) = M_i(1^{n+1})$, and we also know that $D(1^{f(i+1)}) \neq M_i(1^{f(i)+1})$. Together with the fact that $D(x) = M_i(x)$ for all x, this is an untenable situation.

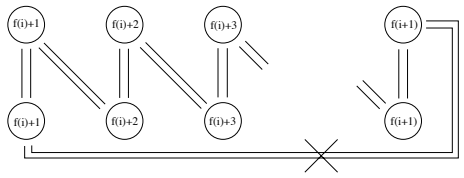

Figure 4.1: All these equalities cannot co-exist with the inequality.

The first row of figure 4.1 are the values of D on strings of as many 1s as the number in the circle. The second row represents M_i. Two solid lines represent equality. By transitivity all circles are equal, but then we have a pair that are not; this is not possible. □

Exercise 4.1.13 *Generalize theorem 4.1.12 to show that if f, g are time constructible functions satisfying the condition $f(n+1) = o(g(n))$, then* $\mathsf{NTIME}(f(n)) \subset \mathsf{NTIME}(g(n))$.

4.2 Oracles and Relativization

An *oracle* is a language O, and an *oracle TM* M^O is an ordinary TM with an extra tape called the *oracle tape*. M writes a string x on the oracle tape, queries the oracle by entering a special "oracle query state," and gets an answer in one step, i.e., the oracle O writes a 0 or a 1 on the first square of the tape denoting that $x \notin O$ or $x \in O$, respectively. Note that this definition can be extended in the natural way to oracles that output strings. Let P^O and NP^O be the set of languages decidable with polytime deterministic (respectively, nondeterministic) TMs with an oracle for O.

For example, suppose that the oracle O is SAT. Then P^{SAT} is the class of languages decidable by polytime TMs which can query SAT. Note that $\mathsf{NP} \subseteq \mathsf{P}^{\text{SAT}}$, and also $\mathsf{co\text{-}NP} \subseteq \mathsf{P}^{\text{SAT}}$. This is because an oracle for SAT can be used to answer queries about TAUT; to check that $\phi \in$ TAUT just check that $\neg\phi \notin$ SAT. In general, if $L_1 = L(M^{L_2})$, then we say that L_1 is *Turing-reducible* to L_2 (where the power of the reduction depends on M; so for example, if M were a polytime TM, L_1 would be polytime Turing-reducible

to L_2, denoted $\leq_{\mathsf{P}}^T$). Note that TAUT $\leq_{\mathsf{P}}^T$ SAT, but it is a big open question whether TAUT $\leq_{\mathsf{P}}^m$ SAT.

Consider the language

$$\text{MINFORMULA} = \{\langle\phi\rangle : \forall\phi[|\phi'| < |\phi| \Rightarrow \phi' \not\equiv \phi]\}$$

i.e.,, MINFORMULA consists of those Boolean formulas for which there is no smaller Boolean formula that computes the same Boolean function. Note that MINFORMULA $\in$ co-NP$^{\text{SAT}}$. To see this, note that a co-NP machines can examine all ϕ' such that $|\phi'| < |\phi|$, and for each ϕ' check (using an oracle for SAT) that $\neg(\phi' \equiv \phi)$ is satisfiable.

It may appear strange that NP$^{\text{SAT}}$ is believed to be a larger class than NP, but note that in NP$^{\text{SAT}}$, we are allowed to query to oracle repeatedly, and take into account negative answers—this *is more* than is allowed by many-one reductions.

The diagonal argument presented in this chapter appears very powerful; recall the Hierarchy Theorems and their consequences. Is it also possible to show that P $\neq$ NP with a diagonal argument? This question is somewhat vague, since we have not defined precisely the nature of a "diagonal argument." Still, we can make the observation that diagonal arguments *relativize*. By this we mean that if we can prove the separation of two complexity classes $\mathcal{C}_1$ and $\mathcal{C}_2$, defined in terms of TMs with bounds on resources, then the same argument should go through for showing the separation of $\mathcal{C}_1^O$ and $\mathcal{C}_2^O$, for any oracle O.

Therefore, if we can prove that there exist two oracles A, B such that $\mathcal{C}_1^A = \mathcal{C}_2^A$ and $\mathcal{C}_1^B \neq \mathcal{C}_2^B$, then we can take that as evidence (with a "grain of salt") that a diagonal argument will not work to show the separation of $\mathcal{C}_1$ and $\mathcal{C}_2$.

In the next theorem we show the existence of an oracle A such that $\mathsf{P}^A \neq \mathsf{NP}^A$; this is a very nice construction, but it implies that the P versus NP question will likely not be settled by a diagonal argument: *"A bliss in proof, and proved, a very woe"* (Shakespeare, Sonnet CXXIX).

Theorem 4.2.1 *There exist oracles A, B such that: (i) $\mathsf{P}^A \neq \mathsf{NP}^A$ and (ii) $\mathsf{P}^B = \mathsf{NP}^B$.*

PROOF: For (ii) let $B =$ QSAT. Trivially, $\mathsf{P}^{\text{QSAT}} \subseteq \mathsf{NP}^{\text{QSAT}}$, and

$$\mathsf{NP}^{\text{QSAT}} \overset{(1)}{\subseteq} \mathsf{NPSPACE} \overset{(2)}{\subseteq} \mathsf{PSPACE} \overset{(3)}{\subseteq} \mathsf{P}^{\text{QSAT}}.$$

For (1) note that QSAT is in PSPACE, so whenever the NP machine wants to query the QSAT oracle, we simply do the computation directly in PSPACE. For (2) we use corollary 3.1.6, and for (3) we use the fact that QSAT is PSPACE-complete: theorem 3.1.7.

For (i), given any oracle A, define L_A as follows:

$$\{w|\exists x, x \in A \text{ and } |x| = |w|\}.$$

Clearly, for any A we have $L_A \in \mathsf{NP}^A$: on input x, guess a w such that $|w| = |x|$, and check that $w \in A$. We now construct an A which ensures that $L_A \notin \mathsf{P}^A$.

Let $M_1^{\sqcup}, M_2^{\sqcup}, M_3^{\sqcup}, \ldots$, be a list of polytime oracle TMs. The symbol "$\sqcup$" denotes that these are TMs with a slot for an oracle (think of it as a PCI slot), but without an actual oracle attached (or the empty oracle, i.e., $O = \emptyset$, so the answer to each query is "no"). We make sure that $M_i^{\sqcup}$ runs in time n^i by attaching a counter. We build A and $\hat{A}$ ($\hat{A} \subseteq \overline{A}$) simultaneously in stages to ensure at stage i that $L(M_i^A) \neq L_A$. In the first stage, **stage** 0, we let $A, \hat{A} := \emptyset$.

At **stage** i A and $\hat{A}$ are finite, so we pick an n such that $n > |w|$ for all $w \in A \cup \hat{A}$, and $2^n > n^i$ (where recall that n^i is the running time of $M_i^{\sqcup}$). We now extend A in such a way so that $L(M_i^A) \neq L_A$.

Run $M_i^{\sqcup}(1^n)$: each time M_i queries the oracle with a string y, if $y \stackrel{?}{\in} A$ has been determined (i.e., it has been established at an earlier stage whether y is in A or $\hat{A}$), we do nothing to A or $\hat{A}$, and we answer the query consistently with what has been decided earlier. Look at this as intercepting the query to the empty slot, and answering according to the scheme presented here. (Note that because of the way we have chosen n, all the strings whose status has been determined are shorter than n.)

If y's status is undetermined, we answer "no" (i.e., declare $y \notin A$, and thereby $y \in \hat{A}$). If at the end $M_i^{\sqcup}(1^n) =$ "yes", declare all remaining y, $|y| = n$, not to be in A (and thereby in $\hat{A}$). If, on the other hand, $M_i^{\sqcup}(1^n) =$ "no", find a y_0 such that $|y_0| = n$, and y_0 has not been queried (it must exist, because in n^i time $M_i^{\sqcup}$ cannot query all $2^n > n^i$ y's of length n), and put y_0 in A.

Therefore, $M_i^A(1^n) =$ "yes" iff $1^n \notin L_A$. We still need to determine the status of all the y's of length $\leq n$ that have not been considered; say we declare all of them not in A, i.e, in $\hat{A}$. □

Theorem 4.2.2 *There exist oracles A, B such that $\mathsf{NP}^A \neq \text{co-}\mathsf{NP}^A$ and $\mathsf{NP}^B = \text{co-}\mathsf{NP}^B$.*

Exercise 4.2.3 *Prove theorem 4.2.2.*

4.2.1 A random oracle separating P and NP

In this section we are going to show that for a random oracle A, we have that $\mathsf{P}^A \neq \mathsf{NP}^A$ with probability 1.

Suppose that we generate a random oracle A by tossing a coin: for every string $x \in \{0,1\}^*$ we toss a coin to determine if $x \in A$. In other words, $\forall x$, $\Pr[x \in A] = \frac{1}{2}$. Then, for such an A, $\Pr[\mathsf{P}^A \neq \mathsf{NP}^A] = 1$, which means that there must exist an oracle separating P and NP (something that we know already from theorem 4.2.1), and in fact "*most*" oracles separate P and NP.

Just as in the proof of theorem 4.2.1, let $M_1^{\sqcup}, M_2^{\sqcup}, M_3^{\sqcup}, \ldots$ be a list of polytime oracle TMs. For any given oracle A, $\mathsf{P}^A = \{L(M_i^A) | i \geq 1\}$.

Given a random oracle A, define the language $L(A)$ as follows: arrange all strings in $\{0,1\}^*$ by length and then in lexicographic order. Given an x, $|x| = n$, to determine if $x \in L(A)$, look at the segment of $n2^n$ strings that follow x (in this ordering). Imagine this segment consisting of 2^n blocks, of n strings each. Then, $x \in L(A)$ iff there is at least one such block of n strings all of which are in A.

$$0, 1, 00, 01, \ldots, x = x_i, \underbrace{\underbrace{\ldots\ldots}_{n}, \underbrace{\ldots\ldots}_{n}, \cdots, \underbrace{\ldots\ldots}_{n}}_{2^n}, x_{i+(n2^n+1)}, \cdots$$

Note that for any oracle A, $L(A) \in \mathsf{NP}^A$: the certificate for $x \in L(A)$ is a number $i \leq 2^n$ (giving the index of the block containing only 1s; note that i can be encoded with polynomially many bits in binary), and we verify those n numbers with the oracle A.

Clearly, $\Pr[\mathsf{P}^A = \mathsf{NP}^A] \leq \Pr[L(A) \in \mathsf{P}^A]$, and:

$$\begin{aligned}
&\Pr[L(A) \in \mathsf{P}^A] && (4.1)\\
=&\Pr[\exists i, L(M_i^A) = L(A)] && (4.2)\\
\leq&\sum_i \Pr[\forall x, x \in L(M_i^A) \square L(A)] && (4.3)\\
\leq&\sum_i \Pr[\forall j, x_j \in L(M_i^A) \square L(A)] && (4.4)\\
=&\sum_i \underbrace{\prod_j \Pr[x_j \in L(M_i^A) \square L(A) \quad | \quad \underbrace{\forall k < j, x_k \in L(M_i^A) \square L(A)}_{(**)}]}_{(*)} && (4.5)
\end{aligned}$$

where the "$\square$" introduced in equation (4.3) denotes the following operation on sets: $X \square Y = \{w | w \in X \iff w \in Y\}$ (i.e., $X \square Y = \overline{X \triangle Y}$, the complement of the symmetric difference). The x_j's introduced in equation (4.4) are any subsequence of our ordering of $\{0,1\}^*$, and in equation (4.5) we introduce the conditional probability, i.e., $\Pr[A|B] = \Pr[A \text{ and } B]/\Pr[B]$, which when it "telescopes" with the $\prod_j$ is equal to the previous line.

Now we are concerned with finding an appropriate subsequence $\{x_j\}$ so that we can bound $(*)$ in equation (4.5) by 0.9. This will have the desirable effect that $\sum_{i=1}^{\infty} \prod_{j=1}^{\infty} 0.9 = \sum_{i=1}^{\infty} 0 = 0$.

We want to choose the x_j's very far apart from each other, say $|x_{j+1}| > 2^{|x_j|}$, so that the events "$x_j \in L(A)$" are independent (because when the x_j's are this far apart, their respective blocks, used to establish if they are in $L(A)$ or not, do not intersect). It is not possible to make the events "$x_j \in L(M_i^A)$" completely independent, since nothing stops M_i^A from making small queries; but we can make them sufficiently independent by noting that M_i^A cannot make *huge* queries and cannot make *too many* queries.

Consider the following table of probabilities, where events E_1, E_2, E_3, E_4, are taking place under the condition of $(**)$ from equation (4.5).

	$x_j \in L(M_i^A)$	$x_j \notin L(M_i^A)$
$x_j \in L(A)$	E_1	E_2
$x_j \notin L(A)$	E_3	E_4

We want to show that $\Pr[E_2 \text{ or } E_3]$ is at least 0.1, and so that $(*) \leq 0.9$ (in

equation (4.5)) as desired.

$$\begin{aligned} \Pr[E_1 \text{ or } E_2] &= \Pr[x_j \in L(A) | (**)] = \Pr[x_j \in L(A)] > 0.6 \\ \Pr[E_3 \text{ or } E_4] &= \Pr[x_j \notin L(A) | (**)] = \Pr[x_j \notin L(A)] > 0.3. \end{aligned} \tag{4.6}$$

First note that we can get rid of the condition $(**)$ since the events "$x_j \in L(A)$" are independent. Then, the probability that a block contains only strings in A is $\frac{1}{2^n}$, so the probability that some block contains only strings in A is $1-(1-\frac{1}{2^n})^{2^n}$. Note that $\lim_{m\to\infty}(1-\frac{1}{m})^m = \frac{1}{e} = 0.367\ldots > 0.3$.

The expression

$$\frac{\Pr[E_2]}{\Pr[E_2]+\Pr[E_4]} \tag{4.7}$$

represents the probability that $x_j \in L(A)$ while $x_j \notin L(M_i^A)$, all under the assumption $(**)$, i.e., $\Pr[x_j \in L(A) | x_j \notin L(M_i^A), (**)]$, and since "$x_j \in L(A)$" is independent of $(**)$, we conclude $(4.7) = \Pr[x_j \in L(A) | x_j \notin L(M_i^A)]$.

Now, during the computation of M_i^A on x_j, M_i^A can query at most polynomially many strings of the oracle, and therefore, it can query at most polynomially many strings from the $2^{|x_j|}$ $|x_j|$-string blocks that follow x_j. So consider those blocks where there is a string that is queried by M_i^A to be unavailable for the random experiment that computes the probability of $x_j \in L(A)$. But, as $|x_j|$ grows, we can assume that the polynomial bounding the number of oracle queries of M_i^A is less than $2^{|x_j|}/2$. Thus,

$$(4.7) = \Pr[x_j \in L(A) | x_j \notin L(M_i^A)] \geq 1 - (1-1/2^{|x_j|})^{2^{|x_j|}/2} = 1 - \frac{1}{\sqrt{e}} \geq \frac{1}{3},$$

since the expression $(1-1/2^{|x_j|})^{2^{|x_j|}/2}$ converges to $\sqrt{1/e}$ as $|x_j|$ grows.

Consider now two cases. If $\Pr[E_3] \geq 0.1$ then we are done with our quest to show that $\Pr[E_2]+\Pr[E_3] \geq 0.1$. If, on the other hand $\Pr[E_3] < 0.1$, then, since $\Pr[E_3]+\Pr[E_4] > 0.3$ (by (4.6)), we have that $\Pr[E_4] > 0.2$. Since, $(4.7) \geq 1/3$, it follows that $\Pr[E_2] > 0.1$.

In either case, $\Pr[E_2]+\Pr[E_3] \geq 0.1$.

Exercise 4.2.4 *Show that for a random oracle A, $\Pr[\mathsf{NP}^A \neq \text{co-}\mathsf{NP}^A] = 1$.*

This results of this section are related to the *Random Oracle Hypothesis* (ROH). The ROH states that if two complexity classes are equal with respect to a random oracle with probability 1, then they are equal; ROH has been shown to be false—see [CCG+94].

4.3 Polynomial time hierarchy

Let $\Sigma_0\mathsf{P} = \Pi_0\mathsf{P} = \mathsf{P}$, and let $\Sigma_{i+1}\mathsf{P} = \mathsf{NP}^{\Sigma_i\mathsf{P}}$, and $\Pi_{i+1}\mathsf{P} = \mathsf{co\text{-}NP}^{\Sigma_i\mathsf{P}}$. We define the *polytime hierarchy* as $\mathsf{PH} = \cup_i\Sigma_i\mathsf{P} = \cup_i\Pi_i\mathsf{P}$. Note that $\mathsf{NP} = \Sigma_1\mathsf{P}$ and $\mathsf{co\text{-}NP} = \Pi_1\mathsf{P}$, and the language MinFormula (defined on page 45) is in $\Pi_2\mathsf{P}$.

An *Alternating Turing Machine* (ATM) is a nondeterministic TM with an additional feature: all its states, except q_{accept} and q_{reject}, are divided into two groups: universal and existential. We give a (recursive) definition of what it means for a node in the configuration graph to be accepting: a leaf node is accepting (rejecting) if it corresponds to an accepting (rejecting) configuration. A universal (existential) node, is accepting if all (some) of its children are accepting. We determine acceptance by looking at the root node; we accept iff the root node is accepting.

Let Σ_i^{Alt} be the class of languages decidable by polytime ATMs where the initial state is existential and there are at most i "runs" of states on any computational branch (i.e., a sequences of existential states, followed by a sequence of universal states, then another sequence of existential states, etc., and there are at most i such sequences). Π_i^{Alt} is defined analogously, except the machines start in a universal state.

Recall that Σ_i^p and Π_i^p were defined in section 2.3 on page 21.

Theorem 4.3.1 *For all i, $\Sigma_i\mathsf{P} = \Sigma_i^{\text{Alt}} = \Sigma_i^p$, and the same holds for Π.*

Proof: We prove it by induction on i. The basis case is $i = 0$ and all three are equal to P (by definition). To show the inductive step assume the claim for i, and show it for $(i + 1)$.

We show the following chain of containments:

$$\Sigma_{i+1}\mathsf{P} \overset{(1)}{\subseteq} \Sigma_{i+1}^{\text{Alt}} \overset{(2)}{\subseteq} \Sigma_{i+1}^p \overset{(3)}{\subseteq} \Sigma_{i+1}\mathsf{P}.$$

(1) Suppose $L \in \Sigma_{i+1}\mathsf{P}$. Then L is decided by an NP TM with a $\Sigma_i\mathsf{P}$ oracle, i.e., M^O. By IH this oracle O is in Σ_i^{Alt}. Consider the ATM N which simulates M as follows: First, N guesses at most polynomially many queries s_i, *together with* the answers to those queries, and for the "yes" queries special "witnessing" strings w_i. That is, N initially guesses:

$$\{(\langle s_1, w_1\rangle, \text{"yes"}), (s_2, \text{"no"}), \ldots, (\langle s_m, w_m\rangle, \text{"yes"})\},$$

and writes it all down. Second, N simulates M^O, but when M^O is about to query the oracle O for the i-th time, N intercepts this query and checks that the query is indeed s_i. If this test passes, it responds with the answer it has guessed. The role of the witnesses w_i will become apparent in the checking stage.

Now N must verify that it guessed the correct answers to the queries. So N branches universally on all the m queries (polynomially many of them). If s_i is a "no" query, then N checks if $s_i \in \overline{O}$ (this is a Π_i^{Alt} language since oracle O is a Σ_i^{Alt} language).

If s_i is a "yes" query, N checks if $\langle s_i, w_i \rangle \in O'$, where O' is just like O, except it also takes as input the initial existential nondeterministic choices the machine deciding O (i.e., a Σ_i^{Alt} machine) makes, and thus O' is *de facto* a Π_{i-1}^{Alt} machine. Thus, the w_i are *witnesses* of what the machine for O must do on the initial existential run of the "yes" input s_i, in order to eventually accept.

Thus, N performs a $\Sigma_{i+1}^{\mathrm{Alt}}$ computation, and so $\Sigma_{i+1}\mathsf{P} \subseteq \Sigma_{i+1}^{\mathrm{Alt}}$.

(2) Suppose that L is decided by a $\Sigma_{i+1}^{\mathrm{Alt}}$ machine M. A computational path of M, on input x, can be described with a sequence of $(i+1)$ numbers in base d, where d is the degree of nondeterminism of M. For example, y_1, y_2, y_3 describe a sequence of $|y_1|$-many existential choices, followed by $|y_2|$-many universal choices, followed by $|y_3|$-many existential choices. Let $R(x, y_1, y_2, \ldots, y_{i+1})$ be a predicate which holds whenever the sequence of choices $y_1, y_2, \ldots, y_{i+1}$ on input x leads to an accepting leaf. Since M is polytime, R is polytime. Clearly,

$$M \text{ accepts } x \text{ iff } (\exists y_1)(\forall y_2)\cdots(Qy_{i+1})R(x, y_1, y_2, \ldots, y_{i+1})$$

(bounds on quantified variables omitted). This shows $\Sigma_{i+1}^{\mathrm{Alt}} \subseteq \Sigma_{i+1}^p$.

(3) Finally, show $\Sigma_{i+1}^p \subseteq \Sigma_{i+1}\mathsf{P}$. Suppose $L \in \Sigma_{i+1}^p$, so $x \in L \iff (\exists y_1)\alpha(y_1, x)$, where $\alpha(y_1, x)$ is a Π_i^p formula, so $\neg\alpha(y_1, x)$ is a Σ_i^p formula, and so by induction $\neg\alpha(y_1, x)$ is a $\Sigma_i\mathsf{P}$ oracle. Now decide L with a $\Sigma_{i+1}\mathsf{P}$ machine as follows: nondeterministically guess a b_1 (of polynomial length), query the $\Sigma_i\mathsf{P}$ oracle for $\neg\alpha(b_1, x)$, and accept iff the answer comes back "no".

For Π it follows by complementing all the classes in the equality given in the statement of the lemma. □

Theorem 4.3.2 *If* $\mathsf{P} = \mathsf{NP}$*, then* PH collapses *to the ground level, i.e.,* $\mathsf{PH} = \mathsf{P}$.

PROOF: It suffices to show that for any i, if $\Sigma_i\mathsf{P} = \Pi_i\mathsf{P}$, then $\Sigma_{i+1}\mathsf{P} = \Sigma_i\mathsf{P}$. By theorem 4.3.1 we can use any of the three formulations of PH. We show that if $\Sigma_i^p = \Pi_i^p$ then $\Sigma_{i+1}^p = \Sigma_i^p$. This is easy, since if $\Sigma_i^p = \Pi_i^p$, then we can "swap" two (opposite) quantifiers (Q_i and Q_{i-1}), and then we can combine the resulting duplicate quantifiers into one, bringing the number of alternations down by one. An inductive procedure now presents itself to finish the proof. □

It is not known whether PH is a sequence of properly contained classes, but it is believed to be so. Let

$$\text{QSAT}_i = \{\psi | \psi = \exists\vec{y}_1 \forall\vec{y}_2 \ldots Q\vec{y}_i \phi(\vec{x}, \vec{y}_1, \vec{y}_2, \ldots, \vec{y}_i) \text{ is satisfiable } \}$$

where ϕ is quantifier free, $Q\vec{z}$ ($Q \in \{\exists, \forall\}$) denotes a vector of variables, i.e., $Q\vec{z}$ is $Qz_1Qz_2 \ldots Qz_n$.

Theorem 4.3.3 QSAT$_i$ *is* $\Sigma_i\mathsf{P}$*-complete.*

Clearly $\mathsf{PH} \subseteq \mathsf{PSPACE}$, and the containment is believed to be proper. Since PH is not believed to collapse at any level, it is not believed to have complete problems, since a complete problem for PH would have to be at some level of PH, and then everything above that level would collapse to it.

The *arithmetical hierarchy* (AH) is defined analogously to PH. Recall that Rec and RE are the classes of recursive and recursively enumerable languages, respectively. (See page 39.) Then, $\Sigma_0\mathsf{Rec} = \Pi_0\mathsf{Rec} = \mathsf{Rec}$, and $\Sigma_{i+1}\mathsf{Rec} = \mathsf{RE}^{\Sigma_i\mathsf{Rec}}$. Unlike PH, AH provably does not collapse at any level.

Theorem 4.3.4 *For all* i, $\Sigma_{i+1}\mathsf{Rec} \neq \Sigma_i\mathsf{Rec}$.

PROOF: The classes $\Sigma_1\mathsf{Rec} = \mathsf{RE}$ and $\Sigma_0\mathsf{Rec} = \mathsf{Rec}$ were separated by diagonalization in theorem 4.1.1. The same diagonal argument relativizes to higher levels of the hierarchy. □

In this paragraph we want to convey some intuition about AH, without developing too much logic; see [BM77] for the necessary background[3]. Let $\mathcal{L}_A = [0, s, +, \cdot; =, \leq]$ be the standard first order language of arithmetic, containing the constant zero, the successor function, plus, times, equality, and less-than-or-equal. Here is an example of a formula over $\mathcal{L}_A$:

$$(s0 < x) \wedge (\forall y \leq x)(\forall z \leq x)(y \cdot z = x \supset (y = 1 \vee z = 1)) \qquad (4.8)$$

[3]Or Stephen Cook's logic notes [Coo06].

where

$$\begin{array}{lll} s0 < x & \text{abbreviates} & s0 \leq x \wedge \neg(s0 = x), \\ \forall y \leq x\alpha & \text{abbreviates} & \forall y(y \leq x \supset \alpha), \\ \exists y \leq x\alpha & \text{abbreviates} & \exists y(y \leq x \wedge \alpha). \end{array}$$

We denote the set of formulas where all quantifiers are bounded as Δ_0. So (4.8) is an example of a Δ_0 formula. Let Σ_i^{A} be the set of formulas which are of the form $\exists y_1 \forall y_2 \ldots Q y_i \alpha$, where α is a Δ_0 formula.

We say that a relation $R(x)$ over $\mathbb{N}$ (where numbers are represented in binary) is arithmetical if it can be represented as a formula $A(x)$ over $\mathcal{L}_A$. For example, the relation Prime(x), which is true iff x is prime, is arithmetical as it can be represented by (4.8). It should come as no surprise (although it is not trivial to prove) that RE relations can be represented with Σ_1^{A} formulas (but note that $\Sigma_0^{\mathrm{A}} = \Delta_0$ relations are a proper subset of Rec; however, Δ_0 does correspond to LTH, the linear time hierarchy—see theorem 4.5.2). In general, for $i \geq 1$, Σ_iRec languages can be given by Σ_i^{A} relations.

While the AH certainly contains all the complexity classes mentioned in these notes, and much more (all recursive languages, and recursively enumerable, etc.), it is not all powerful.

Theorem 4.3.5 (Tarski) *Let TA be the set of sentences over $\mathcal{L}_A$ which are true in the standard model of arithmetic (i.e., TA is the set of all the theorems of number theory). Clearly, TA can be seen as a language when every such theorem is encoded in some standard way as a string over $\{0,1\}^*$. Then, TA $\notin$ AH.*

See, for example, [BM77] for a proof of this theorem.

Note that we could have defined Σ_iRec, for $i \geq 1$, with Σ_i^{A} relations; this is a *machine independent* definition of a complexity class. We have thus surreptitiously come into contact with a beautiful area of complexity, known as *Descriptive Complexity*. In Descriptive Complexity we are concerned with the richness of a language necessary to express the computational complexity of certain concepts. One of the first results in the area is Fagin's Theorem which shows that NP is precisely the set of languages expressible by second-order existential formulas; a great source for this field is ([Imm99]).

4.4 More on Alternating TMs

Define $\mathsf{ATIME}(f(n))$ and $\mathsf{ASPACE}(f(n))$ to be time and space bounded by $O(f(n))$ on an ATM, without a bound on the number of alternations. Let $\mathsf{AP}, \mathsf{APSPACE}, \mathsf{AL}$ be defined analogously to their non-alternating versions.

For example, TAUT and MINFORMULA are both in AP.

Theorem 4.4.1 *For $f(n) \geq n$ we have*

$$\mathsf{ATIME}(f(n)) \overset{(1)}{\subseteq} \mathsf{SPACE}(f(n)) \overset{(2)}{\subseteq} \mathsf{ATIME}(f^2(n))$$

and for $f(n) \geq \log(n)$ we have

$$\mathsf{ASPACE}(f(n)) \overset{(3)}{=} \mathsf{TIME}(2^{O(f(n))})$$

PROOF: **(1)** Depth first search of the ATM's computation tree, not recording configurations, but only the nondeterministic choice (in base $b =$ degree of nondeterminism).

(2) Given $C_{\text{init}}, C_{\text{accept}}$ of the deterministic TM, the ATM branches existentially to guess a mid-point configuration C, and then branches universally to check that $C_{\text{init}} \rightsquigarrow C \rightsquigarrow C_{\text{accept}}$. The depth of the recursion is

$$\log(\text{nr. of configurations}) = \log(2^{O(f(n))}) = O(f(n))$$

and at each step we require $O(f(n))$ space to write a configuration.

(3) [$\subseteq$] Construct the entire computation tree ($2^{O(f(n))}$ many nodes), and mark the accepting node (we can assume that there is a unique C_{accept}). Then, scan the graph repeatedly, and if a universal (existential) configuration is such that all (at least one) of its children are (is) accepting, mark it accepting. Stop when a scan introduces no more accepting nodes.

[$\supseteq$] We use a similar idea to the Cook-Levin theorem (theorem 2.2.8). For time $2^{O(f(n))}$, the computational tableau is of size $2^{O(f(n))} \times 2^{O(f(n))}$, so it cannot be stored in space $f(n)$. However: *"Injurious distance should not stop my way; For then despite of space I would be brought, From limits far remote where thou dost stay"* (Shakespeare, Sonnet XLIV); four pointers into the tableau can be stored:

s_1	s_2	s_3
	s	

When the first pointer is pointing to s, existentially guess s_1, s_2, s_3, check that they yield s, and then universally branch on each s_i to check that it can be obtained from the initial configuration, i.e., from the first row. Once we are in the first row, the answer can be verified directly since we know the input. We start the recursion in the lower-left corner, again assuming that there is a unique C_{accept} where the head is on the first square and the tape is "clean." □

Corollary 4.4.2 $\mathsf{AL} = \mathsf{P}$, $\mathsf{AP} = \mathsf{PSPACE}$, $\mathsf{APSPACE} = \mathsf{EXPTIME}$.

Define the class $\mathsf{STA}(S(n), T(n), A(n))$ to be the class of languages accepted by ATMs that are simultaneously $S(n)$-space-bounded, $T(n)$-time-bounded, and make at most $A(n)$ alternations on inputs of length n. A "$*$" in any position means "don't care." We also write Σ, Π in the third position to impose that the alternations start with $\exists$ of $\forall$. For example, $\mathsf{L} = \mathsf{STA}(\log n, *, 0)$, and $\mathsf{NP} = \mathsf{STA}(*, n^{O(1)}, \Sigma 1)$.

Exercise 4.4.3 *Show that for $S(n) \geq \log n$,*

$$\mathsf{STA}(S(n), *, A(n)) \subseteq \mathsf{SPACE}(A(n)S(n) + S(n)^2).$$

4.5 Bennett's Trick

In his 1962 Ph.D. thesis ([Ben62]), James Bennett shows how to encode a long sequence of numbers with shorter sequences of numbers by using alternation of quantifiers. The idea is that $(\exists \langle x_0, x_1, \ldots, x_n \rangle)$ can be expressed with

$$(\exists \langle y_0, y_1, \ldots, y_{\sqrt{n}} \rangle)(\forall i \leq \sqrt{n})(\exists \langle z_0, z_1, \ldots, z_{\sqrt{n}} \rangle).$$

We are going to use this trick to show a nice result about the linear time hierarchy.

Define the class linear time $\mathsf{LT} = \mathsf{TIME}(n)$, and nondeterministic linear time $\mathsf{NLT} = \mathsf{NTIME}(n)$. Let $\Sigma_0^{\mathrm{LT}} = \mathsf{LT}$ and $\Sigma_{i+1}^{\mathrm{LT}} = \mathsf{NLT}^{\Sigma_i^{\mathrm{LT}}}$, which denotes the class of languages decidable in nondeterministic linear time with a Σ_i^{LT} oracle. Define the linear time hierarchy as $\mathsf{LTH} = \bigcup_i \Sigma_i^{\mathrm{LT}}$. Let $\mathsf{NTIMESPACE}(f(n), g(n))$ be the class of languages decided simultaneously in time $O(f(n))$ and space $O(g(n))$ on a nondeterministic (multi-tape) Turing machine.

Theorem 4.5.1 (Nepomnjascij) *Let $0 < \varepsilon < 1$ be a rational number, and let a be a positive integer. Then,* $\mathsf{NTIMESPACE}(n^a, n^\varepsilon) \subseteq \mathsf{LTH}$.

PROOF: In this proof we are going to forgo the notation $\vec{x}$ representing a vector of Boolean variables, as it would clutter the presentation. When we write $\mathbf{x}$ we mean a vector of vectors of Boolean variables.

Suppose M is a nondeterministic TM running in time n^a and space n^ε. Then, M accepts an input x iff

$$\exists \mathbf{y}[\ \mathbf{y} \text{ represents an accepting computation for } x\ ] \tag{4.9}$$

So $\mathbf{y} = y_1 y_2 \dots y_{n^a}$ where each $|y_i| = n^\varepsilon$, and so $|\mathbf{y}| = n^{a+\varepsilon}$. So $\mathbf{y}$ is too long to verify in linear time (in $n = |x|$). So we use Bennett's trick: let $\mathbf{z} = z_1 z_2 \dots z_{n^{1-\varepsilon}}$ where the z_i represent every $(n^{a-1+\varepsilon})$-th string in $\mathbf{y}$. So now, (4.9) can be restated as follows:

$(\exists \mathbf{z})(\forall i)(\exists \mathbf{u})[\mathbf{u}$ shows z_{i+1} follows from z_i in $n^{a-1+\varepsilon}$ steps & z_n is accepting]

Note that $|\mathbf{z}| = n^{1-\varepsilon} n^\varepsilon = n$, so this is OK, but $|\mathbf{u}| = n^{a-1+\varepsilon} n^\varepsilon = n^{a-1+2\varepsilon}$. But note that $(a - 1 + 2\varepsilon) < (a + \varepsilon)$ because $0 < \varepsilon < 1$. So we have reduced $\mathbf{u}$ with respect to $\mathbf{y}$, by a factor of $n^{1-\varepsilon}$, so to know how many times we have to repeat the above nesting of quantifiers, we solve for i in the following equation:

$$\frac{n^{a+\varepsilon}}{(n^{1-\varepsilon})^i} = n$$

and solving, we get $i = (a + \varepsilon)/(1 - \varepsilon)$, which is a constant as a, ε are fixed. □

Recall the definition of Δ_0-formulas on page 53. Let $\Delta_0^{\mathbb{N}}$ denote the class of languages that can be given by relations representable with Δ_0 formulas.

Theorem 4.5.2 $\mathsf{LTH} = \Delta_0^{\mathbb{N}}$.

An interesting open problem is whether $\mathsf{P} \subseteq \mathsf{LTH}$, or in fact what is the relation of these two complexity classes; are they even comparable?

4.6 Answers to selected exercises

Exercise 4.1.10. The proof is analogous to the proof of the Space Hierarchy Theorem (theorem 4.1.2), and the division by $\log(t(n))$ is now necessary since the machine is "delayed" by keeping a counter.

Exercise 4.2.3. Let B be QSAT just as in the proof of theorem 4.2.1.

For any oracle A define the language $L(A) = \{x|\ \{0,1\}^{|x|} \cap A = \emptyset\ \}$. Note that an NP machine with oracle A can decide $\overline{L(A)}$ by guessing y of length $|x|$ and checking if $y \in A$. So, a co-NP machine with oracle A can decide $L(A)$. Just as in the proof of theorem 4.2.1, let $M_1^{\sqcup}, M_2^{\sqcup}, M_3^{\sqcup}, \ldots$ be a list of all oracle TMs, where machine i takes up at most n^i steps to decide any string of length n.

We construct A in stages so that $L(A)$ is not in NP^A. At **stage** 0, $A_0 := \emptyset$, and at **stage** i, we choose an n such that all strings examined (for membership in A) so far are of length $< n$ and furthermore, $n^i < 2^n$. Simulate $M_i^{\sqcup}$ on input 1^n, and each time $M_i^{\sqcup}$ queries the oracle with some y, if y's membership in A has already been established, answer accordingly, and otherwise declare y *not* in A.

If at the end $M_i^{\sqcup}$ rejects (all paths are rejecting), declare the remaining $\{0,1\}^n$ *not* in A. If $M_i^{\sqcup}$ accepts, find an accepting path and some $y \in \{0,1\}^n$ not queried on it, and declare it in A.

Exercise 4.4.3. Suppose that L is decided with space $S(n)$ and $A(n)$ many alternations, starting with an existential alternation. Let $R(C, C')$ be a predicate that is true iff C, C' are configurations which are not of the same type, meaning that one is existential and the other universal, and C' is reachable from C with a sequence of at most $2^{S(n)}$-many intermediate configurations where all these intermediate configurations are of the same type as C. Then, R is decidable in nondeterministic space $S(n)$, and hence by Savitch's theorem, in deterministic space $S(n)^2$. Now we implement the alternation with recursion as follows: check if there exists a C' (by cycling through all configurations lexicographically) such that $R(C_{\text{init}}, C')$ and furthermore if for all C'' we have $R(C', C'')$, then there exists a C''' such that $R(C'', C''')$ and .... The depth of this is $A(n)$, and from level to level we only need to record a single configuration, and to decide $R(C_1, C_2)$ we need $S(n)^2$ space that can be reused, so we need $A(n)S(n) + S(n)^2$ deterministic space.

4.7 Notes

For the Hierarchy Theorems we follow [Sip06b, §9.1]. See [Sip06b, theorem 9.15, pg. 344] for a proof of theorem 4.1.7. The proof (and statement) of Theorem 4.1.12 follows the exposition in [AB06]. Section 4.2.1 is based

on [SP95, Chapter 22]. The proof of theorem 4.5.2 can be found in [CN06, §3.4]. Exercise 4.4.3 is from [Koz06, homework 4, pg. 279]. The proof of theorem 4.5.2 can be found in [CN06].