Complexity Theory
WS 2009/10

Prof. Dr. Erich Grädel

Mathematische Grundlagen der Informatik
RWTH Aachen

Contents

2 Nondeterministic complexity classes

2.1 Nondeterministic Turing machines

Nondeterministic Turing machines (NTM) are defined just as their deterministic counterparts except that the transition function generally allows several possible transitions.

Again, the most important model is the k-tape TM. The possible transitions are given by a function $\delta : Q \times \Sigma^k \to \mathcal{P}(Q \times \Sigma^k \times \{-1, 0, 1\}^k)$ (modified accordingly if the first tape is a read-only input tape). Again, we will mainly deal with *acceptors*, so that the set of final states is partitioned $F = F^+ \cup F^-$.

A configuration of a nondeterministic Turing machine M usually has several successor configurations. Let $\text{Next}(C) = \{C' : C \vdash_M C'\}$ be the set of successor configurations of C. For each NTM M there is an integer $r \in \mathbb{N}$ such that $|\text{Next}(C)| \leq r$ for all configurations C of M. Given an input x for a nondeterministic Turing machine M, instead of a (complete) sequential computation, we consider a *computation tree* $\mathfrak{T}_{M,x}$ defined as follows:

- the root of $\mathfrak{T}_{M,x}$ is the initial configuration $C_0(x)$;
- the children of each node C are the elements of $\text{Next}(C)$.

A computation path of M on x or, simply, a computation is a sequence $C_0, \ldots, C_t$ of configurations with $C_0 = C_0(x)$ and $C_{i+1} \in \text{Next}(C_i)$, that is, a path through $\mathfrak{T}_{M,x}$ starting at the root.

Definition 2.1. A nondeterministic Turing machine is T-time bounded (respectively S-space bounded) if *no* computation M on inputs of length n takes more than $T(n)$ steps (uses more than $S(n)$ fields).

Definition 2.2. Let M be a NTM and x its input. *M accepts x*, if there is at least one computation of M on x that stops in an accepting configuration. $L(M) = \{x : M \text{ accepts x}\}$ is the language accepted by M.

Definition 2.3.
$\text{NTIME}(T) := \{L : \text{there is a } T\text{-time bounded NTM with } L(M) = L\}$.
$\text{NSPACE}(S) := \{L : \text{there is an } S\text{-space bounded NTM with } L(M) = L\}$.
Other classes such as $\text{NTIME}_k(T)$ can be defined analogously.

Remark 2.4. In informal descriptions of nondeterministic Turing machines, nondeterministic steps are often called "guesses". Thus, "Guess a $y \in \Sigma^m$ " means: Perform a sequence of m nondeterministic steps so that in the ith step, the ith symbol of y is nondeterministically chosen from $|\Sigma|$. The (pseudo-)instruction is equivalent to a computation tree of depth m with $|\Sigma|$ many successors at all inner nodes and with $|\Sigma|^m$ leaves labelled with $y \in \Sigma^m$.

Example 2.5 (A nondeterministic algorithm for the Reachability problem). The following algorithm solves REACHABILITY nondeterministically:

Algorithm 2.1 Nondeterministic REACHABILITY

Input: $G = (V, E)$, a directed graph, $a, b \in V$ $(|V| = n)$
$x := a$
for *n steps* **do**
  if $x = b$ **then** accept **else**
    guess $y \in V$ with $(x, y) \in E$
    $x := y$
  endif
endfor
reject

If there is a path in G from a to b, then there is also one of length $\leq n$ (longer paths would include cycles). Therefore, the algorithm has an accepting computation iff there is a path from a to b. The required space is $\leq 3 \cdot \log n$ ($\log n$ for x, y and the counter). Hence, REACHABILITY belongs to the complexity class $\text{NLOGSPACE} = \text{NSPACE}(O(\log n))$. As we have seen in the exercises, REACHABILITY also belongs to $\text{DSPACE}(O(\log^2 n))$.

2.2 Elementary properties of nondeterministic classes

In order to compare deterministic and nondeterministic complexity classes, we often look at the configuration graphs of nondeterministic Turing machines.

Definition 2.6. Let M be a nondeterministic Turing machine and $s \in \mathbb{N} \cup \{\infty\}$. Then

$$\text{Conf}[s] := \{C : C \text{ is a configuration of } M \text{ using space at most } s\},$$

and the successor relation on configuration defines the (directed) *configuration graph* $G[M, s] = (\text{Conf}[s], \vdash_M)$.

For any S space bounded nondeterministic TM M and any input x (with $|x| = n$), we have:

- The computation tree $\mathfrak{T}_{M,x}$ corresponds to the unravelling of $G[M, S(n)]$ from the input configuration $C_0(x)$.
- M accepts x if there is a path from $C_0(x)$ to some accepting configuration C_a in $G[M, S(n)]$.

Theorem 2.7. $\text{DTIME}(T) \subseteq \text{NTIME}(T) \subseteq \text{NSPACE}(T) \subseteq \text{DTIME}(2^{O(T)})$ for all space-constructible $T : \mathbb{N} \to \mathbb{R}^+$ with $T(n) \geq \log n$.

Proof. The first inclusions $\text{DTIME}(T) \subseteq \text{NTIME}(T) \subseteq \text{NSPACE}(T)$ are trivial. To prove the remaining inclusion, let M be a nondeterministic, T-space bounded TM. Since every configuration uses at most $T(n)$ fields, $G[M, T(n)]$ consists of at most $2^{T(n)}$ different configurations. M accepts x iff there is a path in G from $C_0(x)$ to an accepting configuration. In time $2^{O(T(n))}$, a deterministic algorithm can

(a) construct G and
(b) decide for all accepting configurations C_a whether G contains a path from $C_0(x)$ to C_a.

This follows from the fact that REACHABILITY can be solved by a deterministic algorithm in polynomial time. Q.E.D.

Theorem 2.8 (Savitch's Theorem)**.** $\text{NSPACE}(S) \subseteq \text{DSPACE}(S^2)$ for any space constructible function $S(n) \geq \log n$.

Algorithm 2.2. **Reach**(C_1, C_2, k)

```
if k = 0 then
    if C_1 = C_2 ∨ C_2 ∈ Next(C_1) then return 1 else return 0
else // k > 0
    foreach C ∈ Conf[S(n)] do
        if Reach(C_1, C, k − 1) = 1 ∧ Reach(C, C_2, k − 1) = 1 then
            return 1
        endif
    endfor
    return 0
endif
```

Proof. Let M be a S-space bounded NTM. Then there exists a constant d such that M reaches at most $2^{dS(n)}$ different configurations on inputs of length n. If M has an accepting computation on x, then there is one that is reachable in at most $2^{d \cdot S(n)}$ steps. Furthermore, every configuration of M on x can be expressed by a word of length $c \cdot S(n)$, where c is a constant. Here, we use that $S(n) \geq \log n$.

Again, the idea is to search the configuration graph for reachable accepting configurations. Unlike in the previous argument, we cannot explicitly construct the whole configuration graph or maintain a complete list of reachable positions. However, we can solve the problem by an on-the-fly construction of $G[M, T(n)]$. We define a recursive, deterministic procedure **Reach**(C_1, C_2, k), see Algorithm 2.2, that, given two configurations $C_1, C_2 \in \text{Conf}[S(n)]$ and an integer $k \in \mathbb{N}$, computes the following output:

$$\textbf{Reach}(C_1, C_2, k) = \begin{cases} 1 & \text{if } M \text{ can reach configuration } C_2 \text{ from } C_1 \\ & \qquad \text{in less than } 2^k \text{ steps;} \\ 0 & \text{otherwise.} \end{cases}$$

Let $f(n,k) = \max\{\text{space}_{\text{Reach}(C_1,C_2,k)} : C_1, C_2 \in \text{Conf}[S(n)]\}$. We have

- $f(n,0) = 0$
- $f(n,k+1) \leq c \cdot S(n) + f(n,k)$, where $c \cdot S(n)$ is the space used to write C (space constructibility of S)

Algorithm 2.3. M_{det}

```
Input: x
foreach accepting configuration C_a ∈ Conf[S(n)] do
    if Reach(C_0(x), C_a, d · S(n)) = 1 then accept
endfor
reject
```

Therefore, $f(n,k) \leq k \cdot c \cdot S(n)$. $L(M)$ can be decided by Algorithm 2.3. Since $\text{space}_{M_{\text{det}}}(x) = O(S(n)) + f(n, d \cdot S(n)) = O(S^2(n))$, we conclude that $L(M) \in \text{DSPACE}(O(S^2)) = \text{DSPACE}(S^2)$. Q.E.D.

Corollary 2.9.

(i) NLOGSPACE $\subseteq$ P.

(ii) NPSPACE $=$ PSPACE.

(iii) NP $\subseteq$ PSPACE.

Proof.

(i) $\text{NLOGSPACE} := \text{NSPACE}(O(\log n)) \subseteq \text{DTIME}(2^{O(\log n)})$
$= \text{DTIME}(n^{O(1)}) = \text{P}$.

(ii) $\text{NPSPACE} := \bigcup_{d \in \mathbb{N}} \text{NSPACE}(n^d) \subseteq \bigcup_{d \in \mathbb{N}} \text{DSPACE}(n^{2d}) = \text{PSPACE}$.

(iii) $\text{NP} := \bigcup_{d \in \mathbb{N}} \text{NTIME}(n^d) \subseteq \text{NPSPACE} = \text{PSPACE}$. Q.E.D.

2.3 The Theorem of Immerman and Szelepcsényi

Definition 2.10. Let $\mathcal{C}$ be a class of languages (e.g., a complexity class). Then, we define the class $\text{co}\mathcal{C} := \{\overline{L} : L \in \mathcal{C}\}$, where $\overline{L}$ is denotes the complement of L.

The deterministic complexity classes $\text{DTIME}(T)$ and $\text{DSPACE}(T)$ are obviously closed under the following operations:

- Union: $L, L' \in \mathcal{C} \implies L \cup L' \in \mathcal{C}$;
- Intersection: $L, L' \in \mathcal{C} \implies L \cap L' \in \mathcal{C}$;
- Complement: $L \in \mathcal{C} \implies \overline{L} \in \mathcal{C}$, i.e., $\mathcal{C} = \text{co}\mathcal{C}$.

The nondeterministic complexity classes $\text{NTIME}(T)$ and $\text{NSPACE}(S)$ are also closed under union and intersection. However, the closure under complement is not obvious and possibly incorrect in many instances. Actually, it is conjectured that the complexity class $\text{NTIME}(T)$ is not closed under complement. For $\text{NSPACE}(S)$, this conjecture had been standing for a long time when Immerman and Szelepcsényi presented the following surprising result in 1988.

Theorem 2.11 (Immerman und Szelepcsényi)**.**

$$\text{NSPACE}(S) = \text{coNSPACE}(S)$$

for any space constructible function $S(n) \geq \log n$.

The main idea of the proof is to "count inductively" all reachable configurations. Once the number $R_x(t)$ of configurations that can be reached in t steps is known, we can decide for every configuration C whether it is reachable in $t+1$ steps. If so, this can be verified by guessing an appropriate computation for C. Otherwise, we can verify that $C \notin \text{Next}(D)$ for all $R_x(t)$ configurations of D that are reachable in t steps. More generally, for a nondeterministic decision procedure of L, we only require that $x \in L$ iff there is an accepting computation of M on x. In particular, there can be rejecting computations on x although $x \in L$. To sharpen our terminology accordingly, we introducing the notion of an *error-free* nondeterministic computation or decision procedure.

Definition 2.12. An *error-free* nondeterministic computation procedure for a function f is a nondeterministic Turing machine M with the following properties:

(i) every computation of M on x stops with output either $f(x)$ or ? ("I don't know");

(ii) at least one computation of M produces the result $f(x)$.

An error-free nondeterministic decision procedure for a language L is an error-free nondeterministic computation procedure for its characteristic function χ_L.

We will now prove the following theorem which implies Theorem 2.11.

Theorem 2.13. Let $S(n) \geq \log n$ be space constructible. Then, for every $L \in \text{NSPACE}(S)$ there is an error-free S-space bounded decision procedure.

In particular, this implies that such a decision procedure also exists for $\overline{L}$ and, consequently, $\overline{L} \in \text{NSPACE}(S)$.

Proof. Let M be a S-space bounded NTM that decides L, $C_0(x)$ the initial configuration of M on x and $\text{Conf}[S(n)]$ the set of configurations of M with space usage $\leq S(n)$. As $S(n) \geq \log n$, every configuration $C \in \text{Conf}[S(n)]$ can be described by a word of length $S(n)$. Let

$$\text{Reach}_x(t) := \{C \in \text{Conf}[S(n)] : C \text{ is reachable from } C_0(x) \text{ in} \leq t \text{ steps}\}$$

and set $R_x(t) := |\text{Reach}_x(t)|$.

(1) There is a nondeterministic procedure M_0 with input x, r, t, C, where x is the input of M, $r, t \in \mathbb{N}$ and $C \in \text{Conf}[S(n)]$ $(n = |x|)$, such that if $r = R_x(t)$, then M_0 decides error-free in space $O(S(n))$ whether $C \in$

Algorithm 2.4. $M_0(x, r, t, C)$

```
m := 0
foreach D ∈ Conf[S(n)] do
   /* simulate (nondeterministically) at most t steps of M on x */
   C' = C_0(x)
   for t times do
      if C' ≠ D then
         guess C'' ∈ Next(C')
         C' := C''
      endif
   endfor
   if C' = D then                                   /* D was reached */
      m = m + 1
      if C ∈ Next(D) then output 1
   endif
endfor
if m = r then output 0 else output ?
```

Algorithm 2.5. M_1

Input: x
$r := 1$
for $t = 0$ **to** $t(|x|)$ **do**
  $m := 0$
  foreach $C \in \mathit{Conf}[S(n)]$ **do**
    $z := M_0(x, r, t, C)$ `/* Call of nondet. procedure` M_0 `*/`
    if $z = 1$ **then** $m := m + 1$
    if $z =$? **then output** ?
  endfor
  $r := m$
endfor
output r

$\mathrm{Reach}_x(t+1)$. It does not matter how M_0 operates on (x, r, t, C) with $r \neq R_x(t)$.

Remark. The nondeterministic simulation of at most t steps, for $t = 2^{O(S(n))}$, can be done in space $O(S(n))$, e.g., by guessing a path step by step.

Let $r = R_x(t)$. We obtain:

- If $C \in \mathrm{Reach}_x(t+1)$, there is a computation with output 1. Furthermore, there is no computation with output 0 since no computation passes through all configurations within $t+1$ steps without reaching C at least in the $(t+1)$st step.
- If $C \notin \mathrm{Reach}_x(t+1)$, there is a computation of M_0 that outputs 0. This is the one that follows all computation paths of length at most t, checking for every configuration D met on such a path whether $D \notin \mathrm{Next}(C)$. Moreover, no computation returns 1.

(2) Clearly, there is a function $t(n) = 2^{O(S(n))}$ such that M either halts after $t(n)$ steps or it enters a loop.

Lemma 2.14. There is an error-free nondeterministic $O(S(n))$-space bounded computation procedure for the function $x \mapsto R_x(t(|x|))$.

Proof. Algorithm 2.5 describes the procedure M_1 which calls the nondeterministic procedure M_0 (usually several times) and is therefore

nondeterministic itself. Each time $M_0(x, r, t, C)$ is called by M_1, we have $r = R_x(t)$ for the current values of r and t because:

- $t = 0 : r = 1 = R_x(0)$
- $t > 0 : r = |\{C :$ there is a computation of M_0 on input $(x, R_x(t-1), t-1, C)$ with output $1\}| = R_x(t)$.

In particular, the value of r at the end of a successful computation of M_1 equals $R_x(t(|x|))$. Since there is a computation of M_0 on (x, r, t, C) for all r, t with $r = R_x(t)$ that results in ?, there is also a computation of M_1 that computes the number $R_x(t(|x|))$. This proves the lemma. Q.E.D.

(3) Finally, Algorithm 2.6 specifies an error-free nondeterministic decision procedure for $L = L(M)$.

- Let $x \in L$. Hence, there is a computation of M_1 that results in $r = R_x(t(|x|))$. Then there exists an accepting configuration $C_a \in \mathrm{Reach}_x(t(|x|))$ and therefore a computation of M_0 on $(x, r, t(|x|), C_a)$ with output 1. Therefore, there is a computation of $\tilde{M}$ with output "$x \in L$". On the other hand, it is clear that the answer "$x \in L$" is produced only if there is an accepting configuration C_a with $C_0(x) \vdash_M^x C_a$, that is, if indeed $x \in L$. We have thus shown: $x \in L$ iff there is a computation of $\tilde{M}$ with answer "$x \in L$".

Algorithm 2.6. $\tilde{M}$

Input: x
$r := M_1(x)$ `/* Call of` M_1 `*/`
if $r =$? **then output** ? **else**
  foreach *accepting* $C_a \in \mathit{Conf}[S(n)]$ **do**
    $z := M_0(x, r, t(|x|), C_a)$
    if $z = 1$ **then output** "$x \in L$"
    if $z =$? **then output** ?
  endfor
endif
output "$x \notin L$"

- Let $x \notin L$: Again, there is a computation of M_1 resulting in $r = R_x(t(|x|))$. As no accepting configuration C_a is reachable from $C_0(x)$, for every C_a there is a computation of M_0 on $(x, r, t(|x|), C_a)$ resulting in 0. Therefore, there is a computation of $\tilde{M}$ with answer "$x \notin L$". On the other hand, this answer is given only if M_0 has returned 0 for each C_a, that is, if no C_a is reachable from $C_0(x)$ or, in other words, if $x \notin L$.

Thus, we have shown that $\tilde{M}$ is an error-free nondeterministic decision procedure for $L = L(M)$ and therefore also for $\overline{L}$. Obviously, $\tilde{M}$ is $O(S(n))$-space bounded. By the Space Compression Theorem (Theorem 1.18), we obtain $\overline{L} \in \text{NSPACE}(S)$. Q.E.D.

In particular, it follows that coNLOGSPACE = NLOGSPACE.