3 Time and Space Complexity

So far, we have only studied decision problems with respect to their computability. In this section we will look at the problem of how much space and/or time it takes to solve certain decision problems, and whether there are space and time hierarchies of decision problems.

3.1 Gaps and Speed-ups

First, we show that in certain situations there can be large gaps between complexity classes.

Theorem 3.1 *For every computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ there are monotonically increasing functions $s, t : \mathbb{N} \rightarrow \mathbb{N}$ with*

$$\begin{aligned} \mathrm{DSPACE}(s(n)) &= \mathrm{DSPACE}(f(s(n)) \quad \textit{and} \\ \mathrm{DTIME}(t(n)) &= \mathrm{DTIME}(f(t(n))) \,. \end{aligned}$$

The theorem implies that there are functions t, s with

$$\begin{aligned} \mathrm{DSPACE}(s(n)) &= \mathrm{DSPACE}(2^{2^{s(n)}}) \quad \text{and} \\ \mathrm{DTIME}(t(n)) &= \mathrm{DTIME}(2^{2^{t(n)}}) \end{aligned}$$

At first glance, this seems to be quite surprising. However, it has been shown, for instance, that $\mathrm{DSPACE}(o(\log\log n)) = \mathrm{DSPACE}(1)$, which explains why such gaps can occur. We will see that for "well behaved" functions s and t it is not possible to create such gaps.

Another phenomenon is that it is quite easy to achieve constant improvements in space or time.

Theorem 3.2 *If L can be decided by an $s(n)$ space-bounded Turing machine, then L can be also decided by an $s(n)/2$ space-bounded Turing machine.*

Proof. Let M be any $s(n)$ space-bounded Turing machine that decides L, and let Γ be the tape alphabet of M. In order to obtain an $s(n)/2$ space-bounded Turing machine for L, simply extend the alphabet by $\Gamma \times \Gamma$. This will allow to encode two cells of M in one cell and therefore to reduce the space requirement by a factor of two. □

Theorem 3.3 *If L can be decided by a $t(n)$ time-bounded Turing machine, then L can be also decided by an $n + t(n)/2$ time-bounded Turing machine.*

Proof. The proof will be an assignment. □

Next we will show that there are hierarchies of complexity classes. Recall that we defined the complexity classes over multi-tape Turing machines.

3.2 A space hierarchy

First, we prove the existence of a space hierarchy. For this we will need the following lemma.

Lemma 3.4 *For any function $s : \mathbb{N} \to \mathbb{N}$, any $O(s(n))$ space bounded multi-tape Turing machine can be simulated by a single-tape Turing machine with $O(s(n))$ space.*

Proof. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be an arbitrary k-tape Turing machine. Our aim will be to simulate M by a single-tape Turing machine $M' = (Q', \Sigma', \Gamma', \delta', q_0', F')$ that uses (asymptotically) the same amount of space. To achieve this, we choose a tape alphabet of $\Gamma' = (\Gamma \cup \{\uparrow\})^{2k}$ ("$\uparrow$" is a symbol that is not in Γ). This is possible, because k is a constant. The alphabet allows us to view the tape of M' as consisting of $2k$ tracks. Let these tracks be numbered from 1 to $2k$. The purpose of the odd tracks is to store the contents of the tapes of M, and the purpose of the even tracks is to store the positions of the heads of M:

track 1	contents of tape 1
track 2	$\uparrow$ (position of head 1)
track 3	contents of tape 2
track 4	$\uparrow$ (position of head 2)
...	...

Let $Q' = Q \times \Gamma^k$. This set of states allows M' to store the current state of M plus the k symbols that are read by the heads of M. In this case, a single step of M can be simulated by M' in two phases. In the first phase, the head of M' searches for the positions of the heads of M and stores the symbols that are read by the heads in its state. In the second phase, M replaces the symbols and moves the head pointers according to the transition function δ and stores in its state the next state of M. Since the space required by M' is obviously bounded by the space required by M, the lemma follows. $\square$

Given a function $s : \mathbb{N} \to \mathbb{N}$, the language L_s is defined as

$$L_s = \{\langle M \rangle w \mid M \text{ is a single-tape TM that started with } \langle M \rangle w \text{ uses at most } s(|\langle M \rangle w|) \text{ cells}\} .$$

The next theorem gives a lower bound on the amount of space necessary to decide L_s.

Theorem 3.5 *For any function $s : \mathbb{N} \to \mathbb{N}$, $L_s \notin \mathrm{DSPACE}(o(s(n)))$.*

Proof. Suppose on the contrary that there is a deterministic Turing machine that decides L_s with $o(s(n))$ space. Then, according to Lemma 3.4, there is also a deterministic single-tape Turing machine that decides L_s with $o(s(n))$ space. Let $\tilde{M}$ be any one of these machines. We use $\tilde{M}$ to construct a "mean" single-tape Turing machine M that works as follows:

- simulate $\tilde{M}$ on the given input
- if $\tilde{M}$ accepts, then use infinitely many tape cells
- otherwise, halt

What does M when started with $\langle M \rangle w$? In order to find out, we distinguish between two cases.

1. $\tilde{M}$ accepts $\langle M\rangle w$. Then M started with $\langle M\rangle w$ should use at most $s(|\langle M\rangle w|)$ space. However, M will use in this case infinitely many tape cells.

2. $\tilde{M}$ does not accept $\langle M\rangle w$. Then M started with $\langle M\rangle w$ should use more than $s(|\langle M\rangle w|)$ space. However, since $\tilde{M}$ only uses $o(s(|\langle M\rangle w|))$ space, there must be an n_0 such that for all w with $|w| \geq n_0$, the space used by $\tilde{M}$ is less than $s(|\langle M\rangle w|)$. In this case, also M will use less than $s(|\langle M\rangle w|)$ tape cells.

Since for both cases we arrive at a contradiction, the theorem follows. □

Next we show that, when enough space is available, L_s can be decided. We start with an auxiliary result.

Lemma 3.6 *If a single-tape Turing machine M started on input x uses at most $s(|x|)$ tape cells for more than $|Q| \cdot (s(|x|)+1) \cdot |\Gamma|^{s(|x|)}$ time steps, then it uses at most $s(|x|)$ tape cells for all time steps.*

Proof. We count the number of configurations that M started with x can reach. If we restrict ourselves to configurations (q, w_1, w_2) with $|w_1 w_2| \leq s(|x|)$, then

- there are at most $|Q|$ many states,
- there are at most $|\Gamma|^{s(|x|)}$ many ways of choosing $w_1 w_2$, and
- there are at most $s(|x|)+1$ many ways of separating $w_1 w_2$ into w_1 and w_2 (and thereby determining the head position).

Therefore, there can be at most $N = |Q| \cdot (s(|x|)+1) \cdot |\Gamma|^{s(|x|)}$ different configurations. If M started with x uses at most $s(|x|)$ many cells for more than N steps, then it must have visited a configuration at least twice. Since M is deterministic, this means that M will be in an infinite loop, and thus will never use more than $s(|x|)$ tape cells. □

A function $s(n)$ is said to be *space constructible* if there is an $O(s(n))$ space-bounded Turing machine M that on input x computes the binary representation of $s(|x|)$. It is not difficult to check that all standard functions such as $\log n$, n^k, and 2^n are space constructible. Using this definition, we can prove the following result.

Theorem 3.7 *For any space constructible function $s : \mathbb{N} \to \mathbb{N}$ with $s(n) = \Omega(n)$, $L_s \in \mathrm{DSPACE}(s(n))$.*

Proof. The following $O(s(n))$ space-bounded 2-tape Turing machine $\tilde{M}$ decides L_s.

1. Test whether the input x on tape 1 is of the form $\langle M\rangle w$, where M is a single-tape Turing machine. (This can be done with $O(|x|) = O(s(|x|))$ space.) If not, reject; otherwise continue.

2. Compute $s(|x|)$. (Since s is space constructible, this can be done with $O(s(|x|))$ space.)

3. Compute $T = |Q| \cdot (s(|x|)+1) \cdot |\Gamma|^{s(|x|)}$ on tape 1, where Q is the set of states and Γ is the tape alphabet of M. (This can be done with $O(s(|x|))$ space, because $\log T = O(s(|x|))$.)

4. Simulate on tape 2 M started with $\langle M\rangle w$ for $T+1$ steps. Accept if and only if M uses at most $s(|x|)$ many cells during these steps.

Obviously, the space needed by $\tilde{M}$ is bounded by $O(s(|x|))$. For the correctness we see that

$$\begin{aligned}
\langle M\rangle w \in L_s \quad &\Leftrightarrow \quad M \text{ started with } \langle M\rangle w \text{ uses at most } s(|\langle M\rangle w|) \text{ cells} \\
&\Leftrightarrow \quad M \text{ started with } \langle M\rangle w \text{ uses at most } s(|\langle M\rangle w|) \text{ cells during the first} \\
& \quad T+1 \text{ steps} \\
&\Leftrightarrow \quad \langle M\rangle w \text{ is accepted by } \tilde{M}
\end{aligned}$$

□

Combining Theorem 3.5 and Theorem 3.7 we obtain the following result.

Corollary 3.8 *For any space constructible function $S : \mathbb{N} \to \mathbb{N}$ with $S(n) = \Omega(n)$ and any function $s : \mathbb{N} \to \mathbb{N}$ with $s(n) = o(S(n))$ it holds that*

$$\mathrm{DSPACE}(s(n)) \subset \mathrm{DSPACE}(S(n)) \,.$$

Hence, we have an infinite space hierarchy of decision problems.

3.3 A time hierarchy

Next we prove the existence of a time hierarchy. For this we need the following lemma.

Lemma 3.9 *For any function $t : \mathbb{N} \to \mathbb{N}$, any $O(t(n))$ time bounded multi-tape Turing machine M_1 can be simulated by a 2-tape Turing machine M_2 in $O(t(n) \log t(n))$ time.*

Proof. The first storage tape of M_2 will have two tracks for each storage tape of M_1. For convenience, we focus on two tracks corresponding to a particular tape of M_1. The other tapes of M_1 are simulated in exactly the same way. The second tape of M_2 is used only for scratch, to transport blocks of data on tape 1.

One particular cell of tape 1, known as B_0, will hold the storage symbols read by each of the heads of M_1. Rather than moving head markers as in the proof of Lemma 3.4, M_2 will transport data across B_0 in the direction opposite to that of the motion of the head of M_1 being simulated. This enables M_2 to simulate each move of M_1 by looking only at B_0. To the right of B_0 will be blocks $B_1, B_2, \ldots$ of exponentially increasing length; that is, B_i is of length 2^{i-1}. Likewise, to the left of B_0 are blocks $B_{-1}, B_{-2}, \ldots$ with B_{-i} having a length of 2^{i-1}. The markers between blocks are assumed to exist, although they will not actually appear until the block is used.

Recall that we concentrate on a particular head of M_1. Let a_0 be the contents initially scanned by this head. The initial contents of the cells to the right of this cell are $a_1, a_2, \ldots$, and those to the left are $a_{-1}, a_{-2}, \ldots$. At the beginning, the upper track of M_2 is empty, while the lower tracks holds all the a_i. Hence, the two tracks look as follows.

…																…
…	a_{-7}	a_{-6}	a_{-5}	a_{-4}	a_{-3}	a_{-2}	a_{-1}	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	…
	B_{-3}				B_{-2}		B_{-1}	B_0	B_1	B_2		B_3				

After the simulation of each move of M_1, the following has to hold:

1. For any $i > 0$, either B_i is full (both tracks) and B_{-i} is empty, or B_i is empty and B_{-i} is full, or the bottom tracks of both B_i and B_{-i} are full, while the upper tracks are empty.

2. The contents of any B_i or B_{-i} represent consecutive cells on the tape of M_1. For $i > 0$, the upper track represents cells to the left of those of the lower track, and for $i < 0$, the upper track represents cells to the right of those of the lower track.

3. For $i < j$, B_i represents cells to the left of those of B_j.

4. B_0 always has only its lower track filled, and its upper track is specially marked.

To see how data is transferred, imagine that the tape head of M_1 in question moves to the left. Then M_2 must shift the corresponding data right. To do so, M_2 moves the head of tape 1 from B_0 and goes to the right until it finds the first block, say B_i, that does not have both tracks full. Then M_2 copies all the data of $B_0, \ldots, B_{i-1}$ onto tape 2 and stores it in the lower track of $B_1, B_2, \ldots, B_{i-1}$ plus the lower track of B_i, assuming that the lower track of B_i is not already filled. If the lower track of B_i is already filled, the upper track of B_i is used instead. In either case, there is just enough room to distribute the data. Note that the operations described can be performed in time proportional to the length of B_i.

Next, in time proportional to the length of B_i, M_2 can find B_{-i} (using tape 2 to measure the distance from B_i to B_0 makes this easy). If B_{-i} is completely full, M_2 picks up the upper track of B_{-i} and stores it on tape 2. If B_{-i} is half full, the lower track is put on tape 2. In either case, what has been copied to tape 2 is next copied to the lower tracks of $B_{-(i-1)}, B_{-(i-2)}, \ldots, B_0$. (By rule 1, these tracks have to be empty, since $B_1, \ldots, B_{i-1}$ were full.) Again, note that there is just enough room to store the data, and all the above operations can be carried out in time proportional to the length of B_i.

We call all that we have described above a B_i-*operation*. The case in which the head of M_1 moves to the right is analogous. Note that for each tape of M_1, M_2 must perform a B_i-operation at most once per 2^{i-1} moves of M_1, since it takes this long for $B_1, B_2, \ldots, B_{i-1}$ (which are half empty after a B_i-operation) to fill. Also, a B_i-operation cannot be performed for the first time until the 2^{i-1}th move of M_1. Hence, if M_1 operates in time $t(n)$, M_2 will perform only B_i-operations with $i \leq \log t(n) + 1$.

We have seen that M_2 needs at most $O(2^i)$ steps to perform a B_i-operation. If M_1 runs for $t(n)$ steps, M_2 needs at most

$$t_1(n) = \sum_{i=1}^{\log t(n)+1} O(2^i) \cdot \frac{t(n)}{2^{i-1}} = O(t(n) \cdot \log t(n))$$

steps for the simulation of one tape of M_1. Since the simulation of k tapes only increases the simulation time by a factor of k, the lemma follows. □

Given a function $t : \mathbb{N} \to \mathbb{N}$, the language L_t is defined as

$$\begin{aligned} L_t \;=\; & \{\langle M \rangle w \mid M \text{ is a 2-tape TM that started with } \langle M \rangle w \text{ halts after at most} \\ & t(|\langle M \rangle w|)/|\langle M \rangle| \text{ time steps}\} \,. \end{aligned}$$

We again start with a negative result.

Theorem 3.10 *For any function $t : \mathbb{N} \to \mathbb{N}$ with $t(n) = \Omega(n)$, $L_t \notin \mathrm{DTIME}(o(t(n)/\log t(n)))$.*

Proof. Suppose on the contrary that there is a deterministic Turing machine that decides L_t in $o(t(n)/\log t(n))$ time steps. Then, according to Lemma 3.9, there is a deterministic 2-tape Turing machine that decides L_t in $o(t(n))$ time steps. Let $\tilde{M}$ be any one of these machines. We use $\tilde{M}$ to construct a "mean" 2-tape Turing machine M that works as follows:

- simulate $\tilde{M}$ on the given input
- if $\tilde{M}$ accepts, then run forever
- otherwise, halt

What does M when started with $\langle M\rangle w$? In order to analyze this, we distinguish between two cases.

1. $\tilde{M}$ accepts $\langle M\rangle w$. Then M started with $\langle M\rangle w$ should halt after at most $t(|\langle M\rangle w|)/|\langle M\rangle|$ time steps. However, M will run in this case forever.

2. $\tilde{M}$ does not accept $\langle M\rangle w$. Then, according to L_t, M started with $\langle M\rangle w$ should run for more than $t(|\langle M\rangle w|)/|\langle M\rangle|$ time steps. However, since $\tilde{M}$ only uses $o(t(|\langle M\rangle w|))$ time steps, there must be an n_0 such that for all w with $|w| \geq n_0$, the time used by $\tilde{M}$ is less than $t(|\langle M\rangle w|)/|\langle M\rangle|$. In this case, M will use at most $t(|\langle M\rangle w|)/|\langle M\rangle|$ time steps.

Since for both cases we arrive at a contradiction, the theorem follows. □

Next we prove an upper bound on the time necessary to decide L. A function $t : \mathbb{N} \to \mathbb{N}$ is called *time constructible* if there is a $t(n)$ time-bounded Turing machine that for an input x computes the binary representation of $t(|x|)$. With this definition we can show the following result.

Theorem 3.11 *For any time constructible function $t : \mathbb{N} \to \mathbb{N}$ with $t(n) = \Omega(n)$, $L_t \in \mathrm{DTIME}(t(n))$.*

Proof. The following $O(t(n))$ time-bounded multi-tape Turing machine $\tilde{M}$ decides L_t.

1. Test whether the input x on tape 1 is of the form $\langle M\rangle w$. (This can be done in $O(|x|) = O(t(|x|))$ time steps.) If not, reject. Otherwise continue.

2. Compute $t(|x|)/|\langle M\rangle|$ on tape 2. (Since t is time constructible, this can be done in $O(t(|x|))$ time steps.)

3. Simulate on tapes 3 and 4 M started with $\langle M\rangle w$ for $t(|x|)/|\langle M\rangle|$ time steps. Accept if and only if M halts at one of these time steps. (Since the lookup of the next transition in $\langle M\rangle$ takes $O(|\langle M\rangle|)$ steps and therefore the simulation of a single step of M takes $O(|\langle M\rangle|)$ steps, the simulation of $t(|x|)/|\langle M\rangle|$ time steps of M takes $O(t(|x|))$ steps.)

From the comments above it follows that the time needed by $\tilde{M}$ is bounded by $O(t(|x|))$. The correctness is straightforward. □

Combining Theorem 3.10 and Theorem 3.11 we obtain the following result.

Corollary 3.12 *For any time constructible function* $T : \mathbb{N} \to \mathbb{N}$ *with* $T(n) = \Omega(n)$ *and any function* $t : \mathbb{N} \to \mathbb{N}$ *with* $t(n) = o(T(n))$ *it holds that*

$$\mathrm{DTIME}(t(n)/\log t(n)) \subset \mathrm{DTIME}(T(n)) .$$

Hence, there is also an infinite time hierarchy of decision problems. Using involved methods, one can even prove the following result.

Theorem 3.13 *For any time constructible function* $T : \mathbb{N} \to \mathbb{N}$ *with* $T(n) = \Omega(n)$ *and any function* $t : \mathbb{N} \to \mathbb{N}$ *with* $t(n) = o(T(n))$ *it holds that*

$$\mathrm{DTIME}(t(n)) \subset \mathrm{DTIME}(T(n)) .$$

3.4 Relationships between complexity classes

We start with a definition of one more complexity class. The class EXP is defined as

$$\mathrm{EXP} = \bigcup_{k \geq 1} \mathrm{DTIME}(2^{n^k}) .$$

The following theorem gives some easy relationships.

Theorem 3.14 *For any function* $f : \mathbb{N} \to \mathbb{N}$ *it holds:*

1. *If* $L \in \mathrm{DTIME}(f(n))$*, then* $L \in \mathrm{DSPACE}(f(n))$*.*
2. *If* $L \in \mathrm{NTIME}(f(n))$*, then* $L \in \mathrm{DSPACE}(f(n))$*.*
3. *If* $L \in \mathrm{DSPACE}(f(n))$ *and* $f(n) \geq \log n$*, then there is a constant* c *(depending on* L*) such that* $L \in \mathrm{DTIME}(c^{f(n)})$*.*

Proof. Proof of 1): If a Turing machine M runs for no more than $f(n)$ time steps, then it cannot scan more than $f(n) + 1$ cells on any tape.

Proof of 2): Given a non-deterministic Turing machine M that decides L in time $O(f(n))$, we construct a deterministic Turing machine M' that decides L in the following way: go in a depth-first-search manner through all the computational paths of M. If an accepting path is found, then accept. Otherwise reject. The space necessary to store the directions taken at any of the $O(f(n))$ time steps is at most $O(f(n))$, since M only has a constant number of choices for each time step. Furthermore, since M only runs for $O(f(n))$ steps, it follows from 1) that it only uses $O(f(n))$ space. Hence, the total amount of space needed by M' to simulate M is $O(f(n))$.

Proof of 3): From Lemma 3.6 we know that if an $f(n)$ space-bounded Turing machine M that decides L can run for at most $|Q| \cdot f(n) \cdot |\Gamma|^{f(n)}$ time steps on any input of size n (otherwise, it would run in an infinite loop). Since $f(n) \geq \log n$, there is some constant c such that for all $n \geq 1$, $c^{f(n)} \geq |Q| \cdot f(n) \cdot |\Gamma|^{f(n)}$. Thus, the runtime of M can be bounded by $c^{f(n)}$. $\square$

Theorem 3.14 and Assignment 1 together yield the following relationships.

Corollary 3.15

$$\mathrm{P} \subseteq \mathrm{RP} \subseteq \mathrm{BPP} \subseteq \mathrm{PSPACE}$$

and

$$\mathrm{RP} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP} \ .$$

Next we will study the relationship between PSPACE and NPSPACE.

Theorem 3.16 (Savitch) *Let $s : \mathbb{N} \to \mathbb{N}$ be space constructible. Then*

$$\mathrm{NSPACE}(s(n)) \subseteq \mathrm{DSPACE}(s(n)^2) \ .$$

Proof. Let M be an $s(n)$ space-bounded nondeterministic Turing machine. W.l.o.g. M has only one accepting configuration $\tilde{C}_1$ and one rejecting configuration $\tilde{C}_2$. We first describe a recursive algorithm that does the following: given an input (C_1, C_2, ℓ, t) with $\ell, t \in \mathbb{N}$ and C_1 and C_2 being configurations of length at most ℓ, the algorithm TEST checks, whether $C_1 \stackrel{\leq t}{\to} C_2$ using at most ℓ tape cells. (W.l.o.g. we will a assume that t is a power of 2.)

Function TEST(C_1, C_2, ℓ, t):boolean
if $t = 1$ then
  $Test = (C_1 \to C_2) \ \vee \ (C_1 = C_2)$
else
  $C_3 = (q_0, \epsilon, B \ldots B)$ (ℓ many B's)
  $Test = false$
  repeat
    $Test = \mathrm{TEST}(C_1, C_3, \ell, t/2) \ \wedge \ \mathrm{TEST}(C_3, C_2, \ell, t/2)$
    C_3 = lexicographically next configuration of C_3
  until $Test$ is true or no more configurations possible for C_3
return $Test$

It is not difficult to check that the algorithm above computes the truth value of $C_1 \stackrel{\leq t}{\to} C_2$.

Lemma 3.17 *If C_1 and C_2 have a length of at most ℓ, then $TEST(C_1, C_2, \ell, t)$ can be computed by a deterministic Turing machine using at most $(3\ell + \log t)(\log t + 1)$ tape cells.*

Proof. The trick to keep the space low is to save space by processing the recursive calls on the same part of the tape. We prove the space bound by induction on t.

$t = 1$: we have to test whether $C_1 \to C_2$. This can obviously be done with at most $3\ell \leq (3\ell + \log t)(\log t + 1)$ cells.

$t > 1$: the tape is used in the following way:

C_1	C_2	C_3	bin(t)	place R for recursive calls

The Turing machine for TEST works as follows:

1. Copy C_1, C_3, and $t/2$ to the begin of R.
2. Process $\mathrm{TEST}(C_1, C_3, \ell, t/2)$ on R.
3. Also process $\mathrm{TEST}(C_3, C_2, \ell, t/2)$ on R.
4. If both Tests return "true", then return "true".
5. Otherwise, replace C_3 by the lexicographically next configuration and go to 1).

This results in a space requirement for $\mathrm{TEST}(C_1, C_2, \ell, t)$ of at most

$$\begin{aligned} 3\ell + \log t + \underbrace{(3\ell + \log(t/2))(\log(t/2) + 1)}_{\text{hypothesis}} &= 3\ell + \log t + (3\ell + \log t - 1)((\log t - 1) + 1) \\ &= 3\ell + \log t + (3\ell + \log t - 1)\log t \\ &\leq (3\ell + \log t)(1 + \log t) \; . \end{aligned}$$

The algorithm uses recursive operations. However, it is also possible to construct an iterative algorithm with essentially the same space requirement. □

In order to prove the theorem, we use the result of Lemma 3.6, adapted to non-deterministic Turing machines: if on input x there is an accepting computation of M with space ℓ, then there is an accepting computation with space ℓ and a length of at most $|Q| \cdot (\ell + 1) \cdot |\Gamma|^\ell = 2^{O(\ell)}$. Hence, only $2^{O(\ell)}$ steps have to be checked by TEST to find an accepting computation.

Now we are ready to construct an $O(s(n)^2)$-space bounded Turing machine M' for M. Let $s'(n) = O(s(n))$ be a function so that the space used by M on any input x is at most $s'(|x|)$. (Such as function must clearly exist.) M' works as follows when started with input is x.

- Set T to the power of 2 closest from above to $|Q| \cdot (s'(|x|) + 1) \cdot |\Gamma|^\ell$.
- Execute $\mathrm{TEST}((q_0, \epsilon, x), \tilde{C}, s'(|x|), T)$.
- If TEST returns *true* then accept and otherwise reject.

From Theorem 3.14(3) it follows that M' accepts input x if and only if M has an accepting computation using space at most $s'(|x|)$, which is true if and only if $x \in L(M)$. Hence, $L(M') = L(M)$. Furthermore, Lemma 3.17 implies that the space needed by M' is $O(s'(|x|)^2) = O(s(|x|)^2)$. □

The theorem has the following important consequence.

Corollary 3.18

$$\mathrm{PSPACE} = \mathrm{NPSPACE} \; .$$

3.5 References

- J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, Reading, 1979.
- W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computers and System Sciences* 4:177-192, 1970.