13. Alternation (Continuation)

Last time:

- Alternation = generalization of non-determinism
- If t is at least linear:

$$\text{ATIME}(t) \overset{1)}{\subseteq} \text{DSPACE}(t) \overset{2)}{\subseteq} \text{NSPACE}(t) \overset{3)}{\subseteq} \text{ATIME}(t^2)$$

1) via depth-first search (and storing choices as transitions on a stack)

2) clear

3) via "parallel" implementation of PATH applied to the configuration graph

13.3 From alternating space to deterministic time

Goal:

- Alternating space coincides with exponentially more deterministic time

Theorem:

Let $s : \mathbb{N} \to \mathbb{N}$ with $s(n) \geq \log(n)\ \forall n$. Then

$$\text{ASPSACE}\left(\mathcal{O}(s)\right) = \text{DTIME}\left(2^{\mathcal{O}(s)}\right)$$

Corollary:

$$\text{AL} = \text{P}, \qquad \text{APSPACE} = \text{EXP}$$

Proof of " $\subseteq$ " in theorem:

Idea: Simulate $d \cdot s(n)$-space-bounded ATM M_A by $2^{\mathcal{O}(s)}$-time-bounded DTM M_D that uses the configuration graph of M.

Construction:

- On input x (of size n), M_D constructs the configuration graph of M_A on x.
  - Nodes: configurations of M_A on x.
    ↳ One configuration uses at most $d \cdot s(n)$ space
  - Edge from c to c' if M_A can go from c to c' in a single step.
- After construction the graph, M_D repeatedly scans it to mark configurations as accepting.
  - Initially, $\wedge$-configurations without successors are marked.

- If all successors of an $\wedge$-configuration are already marked, mark it.
- If at least one successor of an $\vee$-configuration is marked, mark it.

- Repeat "scan & mark" until
  - either initial configuration is marked
    ↳ M_D accepts,
  - or no new marks since last iteration (fixed point reached)
    ↳ M_D rejects.

Time consumption:

- Construct configuration graph in $2^{\mathcal{O}(s)}$ time.
- One scan takes $2^{\mathcal{O}(s)}$ time.
- At most $2^{\mathcal{O}(s)}$ scans.
  (We either mark at least one new configuration per step or the fixed point is reached)

Total time:

$$2^{\mathcal{O}(s)} \cdot 2^{\mathcal{O}(s)} + 2^{\mathcal{O}(s)} \in 2^{\mathcal{O}(s)}$$

□

Proof of " $\supseteq$ " in theorem:

To do: Simulate a $2^{d \cdot s(n)}$-time-bounded DTM M_D by an ATM M_A with $\mathcal{O}(s)$ space.

Tricky: Compared to M_D's running time, we have very little space available.

Recall: As in the Ladner-Cook-Levin theorem, we can define the *computation matrix* of M_D of size $2^{d \cdot s(n)} \times 2^{d \cdot s(n)}$.
(i^{th} row = i^{th} step of computation, cell (i, j) = cell content of j in this step.)

Idea: Construct and evaluate the CVP instance of size $2^{d \cdot s(n)}$ *on the fly*.

M_A recursively guesses and verifies the values of the variables $P^a_{i,j}$ and $Q^p_{i,j}$ (cell content, head position and control state).

In each step, M_A will guess the content of a cell (i.e. $P^a_{i,j}$ and $Q^p_{i,j}$ for all p) for a tuple (i, j).

If i > 0, M_A verifies the guess as follows:

- M_A existentially ($\vee$) guesses the values of the parent cells (in row i - 1).

- M_A checks, whether the guessed values would yield the content of cell (i, j) (according to M_D's transition relation).
- M_A universally ($\wedge$) branches to recursively verify the guesses for the parent cells.

If $i = 0$ (first row), M_A can check the guesses directly because it knows M_D's initial configuration (compare input and control state).

We can assume that M_D has a single accepting configuration (accepting state, empty tape, head on \$).

Therefore, start by guessing the control state of lowest left-most cell ($2^{d \cdot s(n)}$, 1) to be q_{accept}. Verify the guess as above and accept if and only if verification is successful.

Space consumption: Only need a constant amount of pointers into the matrix. If we store pointers in binary, we can do it in

$$\log 2^{d \cdot s(n)} \in \mathcal{O}(s)$$

space.

□

14. The polynomial-time hierarchy

Goal:

- Introduce the *polynomial-time hierarchy (PH)*
  - Hierarchy of complexity classes between P and PSPACE
  - Introduced by Stockmeyer
  - Not known whether the inclusions are strict
- Here: Definition using ATMs
- Complete problems for each level
- Later: Definition using oracles

14.1 PH defined via ATMs

Idea:

- NP defined by polytime NTM = polytime ATM with only $\vee$-states (no alternation)
- PSPACE = AP defined by polytime ATM with arbitrary "mode"-alternation
  (In fact, we have seen that PSPACE is about alternation before.)
- Get hierarchy in between by limiting the number of alternations

Definition:

An ATM M is *k-alternation-bounded* (for $k \in \mathbb{N}, k \geq 1$) if for every input x and every path $c_1, ..., c_n$ in the corresponding configuration tree, there are at most k - 1 positions where the mode of c_i is not equal to the mode of c_{i+1} (i.e. $q_i \in Q_\vee, q_{i+1} \in Q_\wedge$ or vice versa).

An ATM M is

- a Σ_k*-machine* if it is k-alternation-bounded and the initial state is existential ($q_0 \in Q_\vee$),
- a Π_k*-machine* if it is k-alternation-bounded and the initial state is universal ($q_0 \in Q_\wedge$).

By convention, Π_0-machines = Σ_0-machines = DTMs.

Remark:

When we defined alternating Turing machines, we said that we do not need to explicitly specify an accepting / rejecting state, because a universal ($\wedge$) configuration without successors is accepting and an existential ($\vee$) configuration without successors is rejecting.

This will now be problematic: depending one the mode of the current state, rejecting or accepting will now introduce an alternation. To avoid this, we again explicitly specify states q_{accept} and q_{reject} such that

- any configuration in which the control state is q_{accept} or q_{reject} has no successors,
- q_{accept} and q_{reject} are considered neither universal nor existential.
  In particular, switching to them does not introduce an alternation.

Now our Turing machine can either accept by going to a universal state with no successors or by going to q_{accept}. It can reject by going to an existential state with no successors or by going to q_{reject}.

Example:

- Σ_1-machines = NTMs.
- The number of alternations of the examples we have seen (parallel PATH, on-the-fly CVP) grows with the input, i.e. those machines are not alternation-bounded.

Definition:

The complexity classes Σ_k^p and Π_k^p of problems decidable by a polytime k-alternation-bounded machine are defined as follows:

$$\begin{aligned}\Sigma_k^p &= \{\mathcal{L}(M) \mid M\ \Sigma_k\text{-machine, t-time-bounded for some } t \in \mathcal{O}(n^m), m \in \mathbb{N}\}\\ \Pi_k^p &= \{\mathcal{L}(M) \mid M\ \Pi_k\text{-machine, t-time-bounded for some } t \in \mathcal{O}(n^m), m \in \mathbb{N}\}\end{aligned}$$

Note that

- $\Sigma_0^p = \Pi_0^p = P$,
- $\Sigma_1^p = NP$,
- $\Pi_1^p = coNP$.

Lemma:

For all $k \in \mathbb{N}$:

$$\text{a)} \quad \Pi_k^p = co\Sigma_k^p = \left\{\overline{\mathcal{L}} \mid \mathcal{L} \in \Sigma_k^p\right\}$$

$$\text{b)} \quad \Pi_k^p \cup \Sigma_k^p \subseteq \Pi_{k+1}^p \cap \Sigma_{k+1}^p$$

$$\text{c)} \quad \bigcup_{k\in\mathbb{N}} \Sigma_k^p = \bigcup_{k\in\mathbb{N}} \Pi_k^p \subseteq PSPACE$$

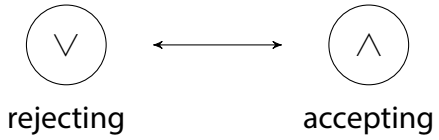
Proof:

a) Given a machine M, we will construct a machine accepting the complement language $\overline{\mathcal{L}(M)}$ within the same time bounds.

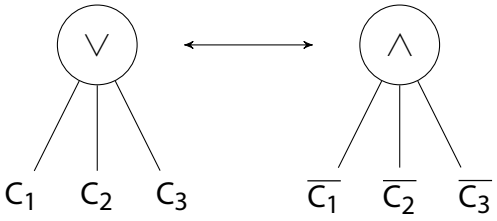
We define the *dual machine* M_d: It behaves like M, just the modes of the states are swapped, i.e. existential states of M are universal states of M_d and vice versa. Whenever M would go to q_{accept}/q_{reject}, M_d will go to q_{reject}/q_{accept}.

One can prove using induction on the structure of computation trees that M_d accepts the complement language of $\mathcal{L}(M)$.

Base case:



Induction step:



Note that the existential ($\vee$) state in the induction step is accepting if and only if at least one of the child nodes C_i is accepting. But in this case, the inverted child $\overline{C_i}$ is rejecting, and so is the universal ($\wedge$) state.

Furthermore, the shape of the computation trees and configurations of M and M_d are the same. In particular: If M was a polynomial time-bounded Σ_k-machine, M_d will be a polynomial time-bounded Π_k-machine. This proves the claim.

b) We need to show $\Pi_k^p \cup \Sigma_k^p$.

$\Pi_k^p \subseteq \Pi_{k+1}^p$ (and $\Sigma_k^p \subseteq \Sigma_{k+1}^p$) is clear.

To show $\Pi_k^p \subseteq \Sigma_{k+1}^p$ (and analogously $\Sigma_k^p \subseteq \Pi_{k+1}^p$), we introduce an auxiliary initial state.

If M is a Π_k-machine, we create a Σ_{k+1} machine by introducing a new existential control state q_0'. We furthermore add a transition that changes the state from the new initial state $q_0' \in Q_\vee$ to the old initial state $q_0 \in Q_\wedge$.

All computations of this new machine will lead to the same result, they are just prolonged by the additional step in the beginning.

c) We need to show

$$\bigcup_{k\in\mathbb{N}} \Sigma_k^p \overset{1)}{=} \bigcup_{k\in\mathbb{N}} \Pi_k^p \overset{2)}{\subseteq} \text{PSPACE}$$

The equality 1) follows using b):

Suppose without loss of generality that there is a language $\mathcal{L}$ with $\mathcal{L} \in \bigcup_{k\in\mathbb{N}} \Sigma_k^p$ but $\mathcal{L} \notin \bigcup_{k\in\mathbb{N}} \Pi_k^p$. Then there is some k' such that

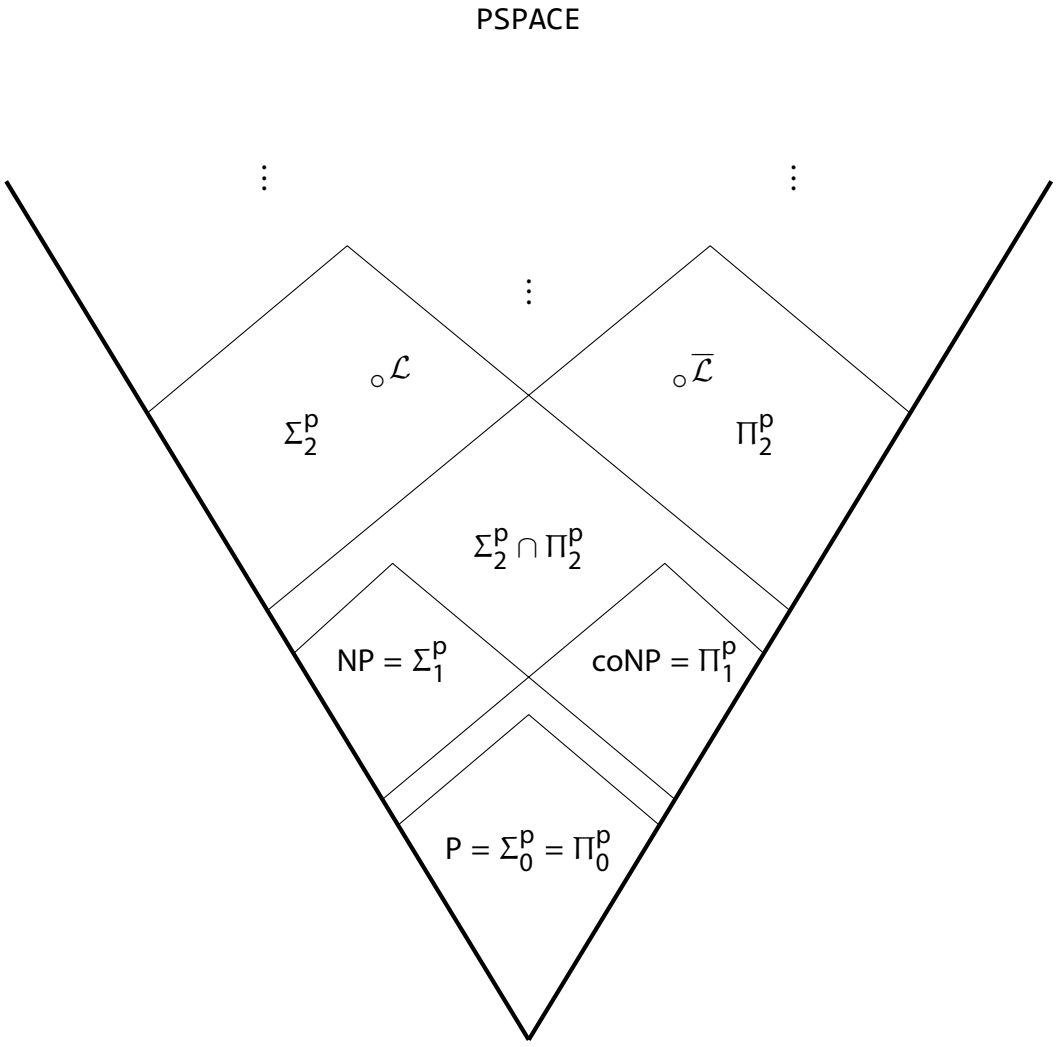
$$\mathcal{L} \in \Sigma_{k'}^p \subseteq \Pi_{k'+1}^p \subseteq \bigcup_{k\in\mathbb{N}} \Pi_k^p.$$

This is a contradiction to the assumption.

Inclusion 2) follows from PSPACE = AP , since number of alternation in AP is not bounded.

□

Picture:


PSPACE
$\circ\mathcal{L}$
$\circ\overline{\mathcal{L}}$
Σ_2^p
Π_2^p
$\Sigma_2^p \cap \Pi_2^p$
$NP = \Sigma_1^p$
$coNP = \Pi_1^p$
$P = \Sigma_0^p = \Pi_0^p$

14.2 A generic complete problem

Goal:

- For each $k > 0$, find a Σ_k^p-complete language.

Definition:

Given any ATM M and natural numbers $k \geq 1, m \in \mathbb{N}$, we define the machine M_k^m. Initially, it behaves like M, but it is artificially restricted to m steps and k alternations. Furthermore, it always starts in an existential state.

- M_k^m has an additional alternation counter with values in $\{0, ..., k - 1\}$.
  (Since k is fixed, this can be stored in the control state.)
- M_k^m has an additional step-counter with values in $\{0, ..., m\}$
- Initially, M_k behaves like M and the counter values are 0.
- If the initial state is universal ($q_0 \in Q_\wedge$), introduce a new artificial initial state $q_0' \in Q_\vee$ (as in the proof of the Lemma).
- Every time M changes from an $\vee$-state to an $\wedge$-state or vice versa, the alternation counter is increased by one.
- Every time M does a step, the step-counter is increased by one.
- If a transition would increase the alternation counter beyond k - 1, it is not possible to take it.
- If a transition would increase the step counter beyond m,it is not possible to take it.

The computation tree of M_k^m one some input is a restricted version of the computation tree of M on this input. More precisely, branches that have length greater than m or more than k - 1 alternations are cut off so that they respect the bounds.

Note: M_k^m is always a Σ_k-machine.

Definition:

For each $k \geq 1$, we define

$$H_k = \{e\#x\#^m \in \{0, 1, \#\}^* \mid m \in \mathbb{N}, e \text{ is the encoding of an ATM M, } M_k^m \text{ accepts } x\}.$$

Note that we can encode an ATM similar to DTMs. We just need to additionally encode the modes of the states.

Theorem:

For all $k \geq 1$, H_k is Σ_k^p-complete with respect to logspace-many-one reductions.

Proof (sketch):
To show *membership*, use a "universal ATM"-construction.

We construct an alternating Turing machine M' deciding H_k. It will first check that the input is of the correct shape. In particular, it will check that e is the encoding of an alternating Turing machine M. It will then simulate M_k^m on x and accept if and only if x is accepted by M.

Note that we can use existential and universal states of M' machine to simulate existential and universal states of M. Since we do not need an alternation to check whether the input has the correct shape, M' is k-alternation bounded. The simulation takes only m steps, which is linear in the size of the input, since m is a part of the input. The simulation causes polynomial overhead and checking that the input is of the correct shape can also by done in polynomial time.

Overall: M' is a polynomial time-bounded Σ_k machine deciding H_k.

To show *hardness*, assume $A \in \Sigma_k^p$, i.e. $A = \mathcal{L}(M)$ where M is n^c-time-bounded for some c and k-alternation bounded.

On input x of length n, choose $m = n^c$, then the computation trees of M and M_k^m are essentially the same. In particular:

$$\text{M accepts x} \quad \text{iff} \quad M_k^{|x|^c} \text{ accepts x} \quad \text{iff} \quad \text{enc(M)\#x\#}^{|x|^c} \in H_k$$

Therefore, we can reduce A to H_k: Given an input x, the reduction will print the encoding of M followed by an #, print the input x and finally print $\#^{|x|^c}$.

Since M is independent of the input, printing its encoding requires constant space and time. To print the input, the transducer computing the reduction can just copy it from its own input tape. To print $\#^{|x|^c}$, we need a binary counter with values in $\{1, ..., |x|^c\}$. Storing it needs $\log(|x|^c) \in \mathcal{O}(\log(|x|))$ space.

Overall, the reduction can be computed using logarithmic space. □

Corollary:

For every $k \geq 1$, $\overline{H_k}$ is Σ_k^p-complete with respect to logspace-many-one reductions.