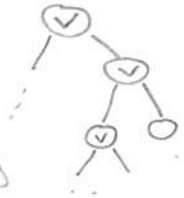
13. Alternation

Goal:
- Study a new resource: alternation
- Useful in understanding the relationship among complexity classes, more generally, between time and space.
- Useful to classify problems according to their complexity (alternation simplifies some proofs).

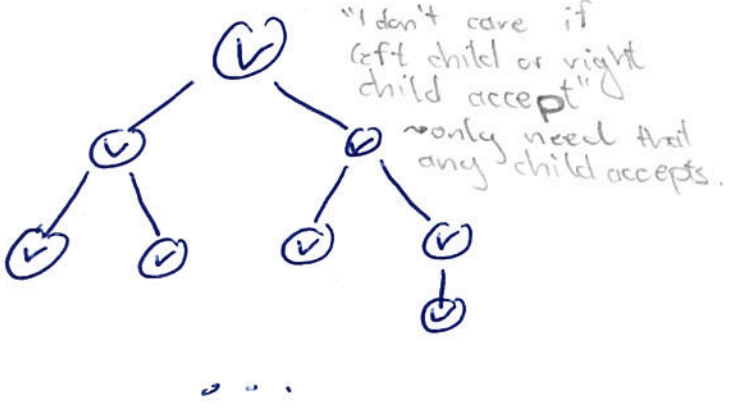
Idea:
- Alternation generalizes non-determinism.
- If non-deterministic algorithm branches a process into multiple sub-processes/child processes, (computation tree ←)
  - It accepts, if any of the child processes accept. (one is sufficient ←)
- An alternating algorithm also branches a process into child processes, but additionally specifies the mode of acceptance:
  - ↳ the current process should accept, if any of its children accept,
  - ↳ or it should accept, if all the children accept.

Illustration:
- Consider non-deterministic and alternating algorithms as (computation) trees that represent the branching structure of the spawned processes.
- Each node represents the configuration in a process.
  - ↳ In a non-deterministic algorithm, each node computes the ∨ of its children. (disjunction; it is suff. that one branch accepts)

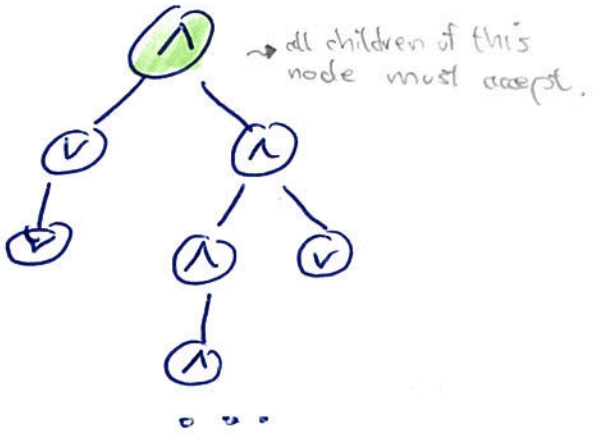


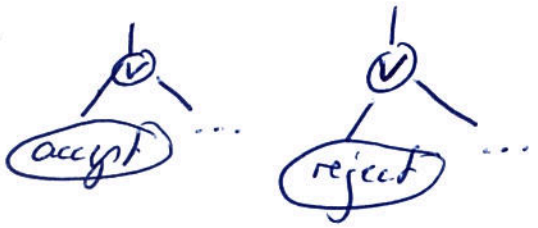
↳ In an alternating algorithm,
the nodes compute the $\wedge$ or the $\vee$,
as determined by the algorithm.

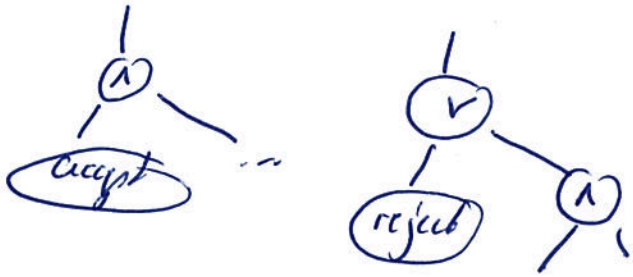
Non-determinism:



Alternation:







Goal: Study the relationship of alternating time and space classes to the known complexity classes.

→ to do so: need notion of an alternating TM

13.1 Alternating Time and Space

Definition:

- An alternating Turing machine (ATM) is a non-deterministic Turing machine with an additional feature:
  the set of states Q is divided into universal states and existential states.
- The semantics of an ATM is defined via its computation tree (that makes explicit all branching).

We label each configuration of the computation tree
by $\wedge$ or $\vee$,
depending on whether the ATM is
- in an universal state → $\wedge$
- or in an existential state → $\vee$

a config. in an univ. state

An $\wedge$-configuration is accepting,
if all its children are accepting.

a config. in an existstate

An $\vee$-configuration is accepting,
if some child is accepting.

The input is accepted, if the initial configuration
is found to be accepting.

- Note that an ATM does not need to make explicit accept and reject states:
  - ↳ An $\vee$-configuration without children is rejecting
  - ↳ An $\wedge$-configuration without children is accepting.

if has no child which could acc.

all children accept

Note:
There are alternative definitions of ATMs in the literature that do contain $\neg$ as a labelling of states. → But this model does not have more power
($\neg$-configurations are supposed to have a single successor).

Lemma:
Every ATM M with negation can be converted
into an ATM M' without negation
without overhead in terms of time and space.

Proof (Sketch):
- Remember from logic the concept of dualizing a formula F

by turning $\wedge$ to $\vee$ and $\vee$ to $\wedge$.

Then, if we also invert the valuation ℓ to

$$\mathcal{U}(a) := 1 - \ell(a),$$ (for any var.)

we obtain

$$\mathcal{U}(\text{dual}(F)) = 1 - \ell(F).$$

- With the same trick ($\wedge \rightsquigarrow \vee$, $\vee \rightsquigarrow \wedge$), we can dualize the NTM M into M_d. (also has negation)
- We moreover replace every negation $\cdot$ in M by $\vee$ and $\cdot$ in M_d by $\wedge$.
- Then M' is the union of M and M_d.

Upon every former negation, M moves to M_d (by $\vee$) and M_d moves to M (by $\wedge$). (if we see a neg and are in M → go to M_d)

□

The time and space requirements of NTMs are defined like for NTMs.

↳ An NTM is $t(n)$-time-bounded, if all paths in the computation tree are of length $\leq t(n)$.

↳ An NTM is $s(n)$-space-bounded, if all paths in the computation tree use at most $s(n)$ work tape cells (per configuration).

Definition: Let $s, t : \mathbb{N} \to \mathbb{N}$.

↳ $\text{NTIME}(t(n)) := \{L(M) \mid M \text{ is a } t(n)\text{-time-bounded NTM (that is a decider)}\}$

- $\exists SPACE(s(n)) := \{L(M) \mid M$ is an $s(n)$-space-bounded ATM (that is a decider)$\}$

There are alternating analogues of the resp. complexity classes:

$$\exists L := \exists SPACE(O(\log n))$$

$$\exists P := \bigcup_{k \in \mathbb{N}} \exists TIME(O(n^k))$$

$$\exists PSPACE := \bigcup_{k \in \mathbb{N}} \exists SPACE(O(n^k))$$

$$\exists EXP := \bigcup_{k \in \mathbb{N}} \exists TIME(2^{O(n^k)})$$

13.2 From Alternating TIME to Deterministic Space

Goal:
- Show that, up to a polynomial, alternating time coincides with deterministic space.
- We add O for not having to reason about compression and speed-up for ATMs.

Theorem: Let $t(n) \geq n$.

$$\exists TIME(t(n)) \overset{(1)}{\subseteq} DSPACE(t(n)) \overset{(2)}{\subseteq} \exists TIME(O(t(n)^2)).$$

In particular, $\exists P = PSPACE$.

$$AP = \bigcup_{ls} ATIME(\mathcal{O}(n^k)) \subseteq \underbrace{\bigcup_{ls} DSPACE(\mathcal{O}(n^k))}_{PSPACE} \subseteq \bigcup_{ls} ATIME(\mathcal{O}(n^{2k})) \subseteq AP$$

Proof:

(1): We convert a $t(n)$-time-bounded ATM M into a [redacted]-space-bounded DTM M' s.th. $L(M)=L(M')$.
- On input x, the DTM M' performs a depth-first search on M's computation tree,

to determine which configurations are accepting.
It accepts, if the root node (start configuration of M) is found to be accepting.
We have seen a similar construction when showing $NTIME(t(n)) \subseteq DSPACE(t(n))$.

paths in tree are of length s(n)

↳ Initially, it seems we need an $t(n)$-hight stack with entries that are configurations of length $t(n)$, hence we need $O(t(n)^2)$ space.

and we need to store the other children are acc. out degree $\leq |\delta|$ → $|\delta| \cdot t(n) \in O(t(n))$

↳ Then we find that we only have to store the non-deterministic choices. ⇝ $O(t(n))$ space
(Here, it does not matter whether we have $O(t(n))$ space, because for regular DTMs we have compression).

(2): We actually show the stronger inclusion

(DSPACE(t(n)) ⊆) $NSPACE(t(n)) \subseteq ATIME(O(t(n)^2))$.

The idea is to use a ~~parallel~~ parallel implementation of Savitch's procedure

$sav(\alpha, \beta, k)$

that determines whether configuration α can go to β in $\leq k$ steps.

↳ if $k=0$ or $k=1$, we directly check $\alpha = \beta$ or $\alpha \to \beta$.

↳ if $k \geq 2$,

how many conf. can a NTM which is t-time-bounded visit? ⇝ $s(n) = 2^{t(n)}$

· we guess $\gamma \in \Delta^{s(n)}$ using $\vee$-branching
Note that the guessing cannot be done instantaneously, but needs $s(n)$ steps. → we compute $t(n)$ configurations.

one path should be accepting

- The result is $|\Delta|^{s(n)}$ independent sub-processes, one for each γ.
  For each γ, we check in parallel
  $\text{sav}(\alpha, \gamma, \lceil \frac{t(n)}{2} \rceil)$ and $\text{sav}(\gamma, \beta, \lceil \frac{t(n)}{2} \rceil)$
  using $\wedge$-branching. → both parts of path must accept
  paths in comp. tree of length $O(t(n)^2)$.

This (alternating) procedure can be implemented in $O(s(n)^2)$ alternating time. □

3.3 From Alternating Space to Deterministic Time

Goal: Show that alternating space coincides with exponentially more deterministic time.

Theorem: Let $s(n) \geq \log n$.

$$ASPACE(O(s(n))) = DTIME(2^{O(s(n))}).$$

$$APSPACE = \bigcup_{k} ASPACE(O(n^k)) = \bigcup_{k} DTIME(2^{O(n^k)}) = EXP$$

In particular, $AL = P$ and $APSPACE = EXP$.

Proof: $AL = ASPACE(O(\log n)) = DTIME(2^{O(\log n)}) = \bigcup_k DTIME(O(n^k)) = P$

$\subseteq$: We construct an $2^{O(s(n))}$-time-bounded DTM M' that simulates an ATM M that is $O(s(n))$-space bounded.

- On input x, M' constructs the configuration graph of M on x
  - The nodes are the configurations of M on x that use at most $d \cdot s(n)$ space (d appropriate for $O(s(n))$)
  - Edges go from a configuration α to a configuration β, if α can go to β in a single step.

- After constructing the graph, M' repeatedly scans it to mark configurations as accepting:
  - Initially, only the $\wedge$-configurations without successors are marked. ~ these are accepting
  - An $\wedge$-configuration with only accepting successors is marked accepting
  - An $\vee$-configuration with some accepting successor is also marked accepting.
- M' accepts, if the start configuration is marked.
  It rejects, if no more new nodes are marked (fixed point). as accepting
- The configuration graph can be constructed in $2^{O(s(n))}$ time.
  Scanning each step costs $2^{O(s(n))}$ time.
  There are at most $2^{O(s(n))}$ scans.
  Hence, the time bound is

$$\underbrace{2^{O(s(n))} \cdot 2^{O(s(n))}}_{\text{scanning } 2^{O(s(n))} \text{ times}} + \underbrace{2^{O(s(n))}}_{\text{constructing}} = 2^{O(s(n))}.$$

(2") We show how to simulate a $2^{O(s(n))}$-time-bounded DTM M by an RTM M' with $O(s(n))$ space.
The simulation is tricky because the space available to M' is so much less than M's compute time.
Essentially, the space is just sufficient to store a pointer into the computation matrix of M from the Ladner-Cook-Levin theorem.

Indeed, we will build on Ladner's construction and show how to construct and evaluate on-the-fly

a CVP-instance of length $2^{O(s(n))}$.

- Machine M' recursively guesses and verifies the values of the variables $P^a_{i,j}$ and $Q^p_{i,j}$ (in step i, we see a in cell j; in step i, the TM's head is in cell j and the TM is in state p)

To verify the content of a cell i,j with $i>0$:

↳ M' existentially (∨) guesses the values of the parent cells (for computation step $i-1$), $j, j-1, j+1$

↳ checks whether the guessed content would yield the content of cell i,j according to M's transition relation, and ($P^a_{ij} = 0 \wedge Q^p_{i,j-1} 1 \ldots 1 \ldots$) $\vee(\ldots)$ $\vee(\ldots)$

↳ universally (∧) branches to recursively verify these guesses.

For a cell in the first row ($i=0$), M' can verify the guess directly, because it knows M's start configuration.

- We can assume M has a single accept configuration where the head is on the left-end marker of the tape. Hence, M' can check whether M accepts x by checking the valuation of the lowest leftmost cell of the computation matrix. □
- Since M' never needs to store more than a single pointer to a variable in the matrix, and since such pointers can be stored in binary, M' needs $\log 2^{O(s(n))} = O(s(n))$ space.

□