5. Savitch's Theorem

The most important question
in Theoretical Computer Science
is

$$P = NP.$$

For space, the corresponding question
was solved very early.

Theorem (Walter Savitch, 1970):

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

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

In particular, $NPSPACE = DPSPACE$
and typically referred to as $PSPACE$.

Idea:

- Generalize acceptance to the problem PATH (in a directed graph)
  defined as follows:

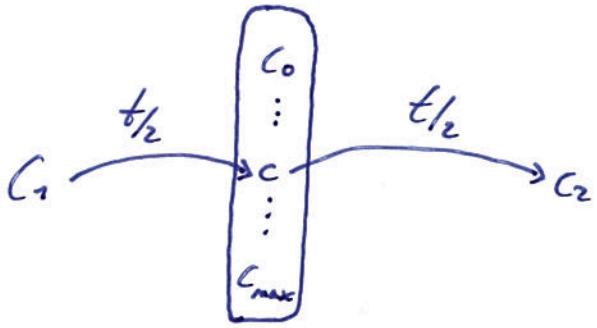
  Given: Configurations c_1, c_2, and a time bound t.
  Question: Can we get from c_1 to c_2 in at most t steps.

- Our goal is to solve PATH
  in a deterministic way in $s(n)^2$ space.
  (When we think about graphs, we solve
  reachability in a directed graph with n nodes
  deterministically in $(\log n)^2$ space.)

- The idea is to search for an intermediate configuration c and recursively check
  (1) whether c_1 can get to c in $t/2$ time and
  (2) whether c can get to c_2 in $t/2$ time.

  Reusing the space for each of the checks gives a significant saving of space.

Illustration:



- The algorithm needs space for storing the stack.
  Each stack frame has to hold:
  ↳ c_1, c_2, the currently enumerated configuration c, and
  ↳ the counter t stored in binary.
  Such a stack frame can be stored in $O(s(n))$.
- The depth of the recursion is $\log t$,
  where t is the maximum time the NTM uses on any branch.
  We discussed in the previous lectures
  that t is bounded by $c^{s(n)}$ for some c
  and thus $\log t = O(s(n))$.
- Together, the simulation needs

$$\underbrace{O(s(n))}_{\substack{\text{stack}\\\text{height}}} \cdot \underbrace{O(s(n))}_{\substack{\text{size of}\\\text{each}\\\text{stack frame}}} = O(s(n)^2).$$

Proof:

We assume that $s(n)$ is space constructible.
Otherwise we do the enumeration trick from the previous lecture.

- Since the given NTM is $s(n)$-space bounded,
  we know it is $c^{s(n)}$-time bounded
  by a lemma from the last lecture.
- We discussed that we can encode the configurations of M
  as strings over an alphabet Γ
  of length $= d \cdot s(n)$ // One can choose an encoding so that equality holds.
- If $\alpha, \beta \in \Gamma^{d \cdot s(n)}$, we write

  $\alpha \xrightarrow{\leq k} \beta$ to mean:

  α and β represent configurations of M
  and α can go to β in at most k steps
  and without exceeding the space bound $s(n)$.

The deterministic algorithm will check

$$c_{init} \xrightarrow{\leq k} c_{accept}$$

where c_{init} and c_{accept} are
the initial and the accepting configurations of the given NTM.
Indeed, we can assume accept to be unique
(by deleting the tape content, moving left, and
only then accepting).

- The deterministic algorithm to check $\alpha \xrightarrow{\leq k} \beta$
  is implemented by the following function $\text{sav}(\alpha, \beta, k)$:

```
bool sav(α, β, k) {
  if k=0 then return α=β;
  if k=1 then return α → β;
  if k>1 then {
    for all γ ∈ Γ^{d·s(n)} enumerated in lexicographic order
                          and checked for being a configuration {
      bool a1 := sav(α, γ, ⌈k/2⌉);
      bool a2 := sav(γ, β, ⌊k/2⌋);
      if a1 ∧ a2 then return true;
    } end for all
    return false;
  } end if
}
```

Correctness: $sav(\alpha, \beta, k) = true$ iff $\alpha \xrightarrow{\leq k} \beta$.

Space requirement:

- The depth of the recursion is $\log(c^{s(n)}) = O(s(n))$.
- Each stack frame contains
  - the configurations α, β, γ, and
  - the value k stored in binary.

  This needs $3 \cdot d \cdot s(n) + \log(c^{s(n)}) = O(s(n))$.
- Together, we obtain $O(s(n)^2)$.
- We can get rid of the constant using tape compression. □