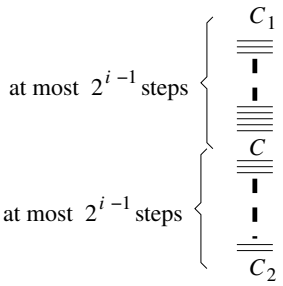
Recursive computation

It is possible to verify Reach recursively.

To check whether it is possible to reach C_2 from C_1 in at most 2^i steps we may check whether it is possible to reach some intermediate configuration C from C_1 in at most 2^{i-1} steps and then reach C_2 from C in at most 2^{i-1} steps. This process can be iterated until transforming the verification of Reach on two steps in two verifications of Reach of one step each. These can be easily implemented using the definition of N.



REACH (C_1, C_2, i)
  if $i = 0$
    if $C_1 = C_2$ or C_1 can be transformed into C_2 in one step
      return TRUE
    else return FALSE
  else
    for each C
      if REACH (C_1, C, $i-1$) and REACH (C, C_2, $i-1$)
        return TRUE

To complete the proof of the theorem we need one more bit of code.

The Turing machine N uses at most $c_1 s(n)$ tape cells.

$n \leftarrow |x|$
let C_0 be the initial ID of N on input x
if REACH (C_0, C_f, $c_1 s(n)$)
  return TRUE

Therefore non-determinism does not add any additional power to, say, polynomially space bounded computations.

Example

Assume N is a NTM that accepts $\{0^n 1^n : n \geq 1\}$, and assume N uses space n. Let q_0 (resp. q_f) be N's initial (unique final) state.

> Take N to be the deterministic TM accepting strings of the form $0^n 1^n$ (a deterministic TM is a particular case of a non-deterministic one!). Then $|Q| = 5$, $|\Gamma| = 5$ and $s(n) = n$.

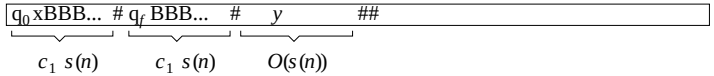
Let $x = 000111$.

Therefore $C_0 \equiv q_0\ 0\ 0\ 0\ 1\ 1\ 1$.

The simulating machine will start checking whether $C_f \equiv q_f\ B$ is reachable from C_0 in at most 2^y steps where

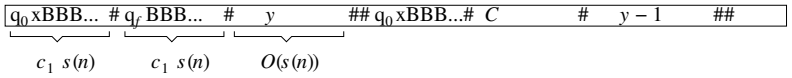
$$y = \lceil \log(|\Gamma|^{s(n)} s(n)|Q|) \rceil = \lceil \log(5^6 \times 6 \times 5) \rceil = 19$$

After some preprocessing the tape content of the simulating machine will be



The main **for** loop inside REACH will cycle through all possible strings of length $c_1 s(n)$ of the form $\alpha_1 q \alpha_2$ with $\alpha_i \in \Gamma^*$ and $q \in Q \setminus \{q_0\}$ and call itself recursively in each case. At some stage it will reach the right intermediate configuration C and then it will "behave correctly".

For instance if C_f is not immediately reachable from C_0 then the simulating machine tape content changes to:



and a recursive call to REACH can start on the right-most three items on the tape. If that fails, the next C can be generated and a new trial starts.

Complexity analysis

How much space does the simulating machine use?

1. At each level of the recursion the simulating machine will need to remember C and a counter (running up to $|\Gamma|^{s(n)} s(n)|Q|$). If the counter is binary, it will use $\log(|\Gamma|^{s(n)} s(n)|Q|) = O(s(n))$ space.
2. The stack of recursive calls will store few ID's and a counter (again this takes $O(s(n))$ space) and there are at most $O(s(n))$ nested calls, since the third parameter in REACH(C_1, C_2, i) is decreasing by one at each step.

Interactive proofs

Digression. What is a proof procedure? Perhaps the most natural definition is that of a sequence of statements written in a book that can then be read to convince the "verifier" of the validity of the argument.

A more general way of communicating a proof is based on the concept of interaction, and consists of explaining the proof to some recipients.

Take the teacher-student environment. The prover (that is, the teacher) can take full advantage of the possibility of interacting with the verifiers (the students). The latter may ask questions at crucial points of the explanation and receive answers. This make the prover's life much easier!

In a sense, writing down a proof that can be understood and checked by every verifier without interaction is a much harder task because, in some sense, the prover has to answer all possible questions in advance.

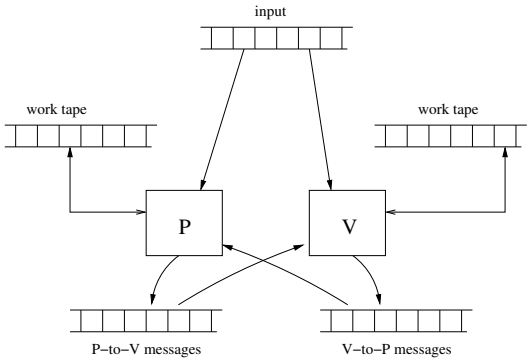
We will consider proving procedures, called interactive proof systems, in which a prover wants to convince a verifier of the correctness of a proof.

In typical complexity-theoretic fashion we shall view an interactive proof system as a new method for recognising languages.

We will define the model, introduce some relevant complexity classes and then state an interesting result relating the complexity of problems defined with respect to this model to standard TM complexity classes.

We will use interactive proof systems in the context of approximation algorithms in the last weeks of this course.

Interactive proof systems



An interactive proof system consists of two TM's, P and V. The two TM's can exchange messages. The exchange of messages takes place in the two *communication tapes* labelled $P \rightarrow V$ messages and $V \rightarrow P$ messages. The first tape is write-only for P and read-only for V. The second one is write-only for V and read-only for P. Both TM's have their own private working and input tapes. Additional assumptions include:

1. V is DTM working in polynomial time.
2. P is DTM with no limitation in time or space.
3. P and V take turns in being active and V starts the computation. When a machine is active, it can perform internal computation, read and write on the correct tapes and send a message to the other machine by writing on the appropriate communication tape.
4. Both the length and the number of messages exchanged between P and V are bounded by suitable polynomials in the input length.
5. V can, during its turn, terminate the interactive computation by entering either the accepting or the rejecting state.

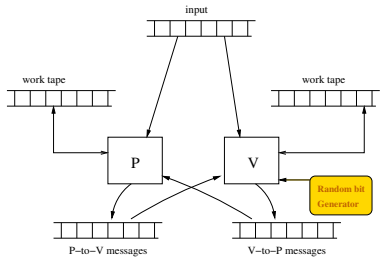
Acceptance

The acceptance criterion is straightforward. (P, V) accepts (rejects) input x if V halts in an accepting (rejecting) state. A language L *admits a deterministic interactive proof* if a verifier V exists such that

1. A prover P^* can be built such that (P^*, V) accepts all $x \in L$.
2. For all provers P, (P, V) rejects all $x \in L$.

Denote by DIP the class of languages which admit a deterministic interactive proof.

Interactive proof systems



The notion of deterministic prover can be extended by allowing V to toss coins[a]. In this case we will require that for each $x \in L$ the prover can convince the verifier with high probability, and, for all $x \notin L$, no prover can convince the verifier that $x \in L$ with better than negligible probability.

Denote by IP the class of languages which admit an interactive proof.

[a] In the simplest possible case this means that V can make moves at random.

Main results

- DIP = NP (relatively simple)
- IP = PSPACE (very deep result, we will not prove it)

(NP $\subseteq$ DIP) (Sketch) We first need to "believe" that $L \in$ NP if and only if there is a language $L_c \in$ P and a polynomial p such that

$$L \equiv \{x : \exists y (x, y) \in L_c \wedge |y| \leq p(|x|)\}$$

Given L the prover will send to V the right y and the verifier will in polynomial time check that $(x, y) \in L_c \wedge |y| \leq p(|x|)$.

(DIP $\subseteq$ NP) (Sketch) We need a non-deterministic simulation of the computation of a given deterministic interactive proof system.

Let $L \in$ DIP and let (P, V) be the proof system for L. Given an input x, the number of messages exchanged between P and V is polynomial. Thus an NTM exists which alternatively simulates V and guesses all possible messages of P.

Appendix: One more Turing machine example

Definition of a Turing machine that takes as input a binary string $x_1, \ldots, x_n$ and changes x_n to its complement.

State	0	1	B
q_0 (move right state)	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_1, B, L)
q_1 (flip state)	$(q_1, 1, R)$	$(q_2, 0, R)$	
q_2 (final state)			

Exercise. Modify the previous table so that the algorithm correctly computes the successor of the given binary number $x_1, \ldots, x_n$.

More Exercises

1. Complete the definition of the TMs for the string matching problem that were sketched last time.
2. Write a pseudo-code description of the algorithms implemented by the TMs mentioned in the previous exercise.
3. Simulate the deterministic machine for the existence of a valid shift problem on $T = 1101$ and $P = 00$.