HW3 Randomized algorithms & Random structures

MPRI 1.24 Tue. Nov. 22, 2016 - **Due on /**

You are asked to complete the exercise(s) marked with a [★] and to send me your solutions at:
nicolas.schabanel@cnrs.fr
(or drop it in my mail box at the 4th floor of Sophie Germain) on /.

■ **Exercise 1 (NAND-tree evaluation).** Let us consider a complete binary tree of height h, whose internal nodes are labeled NAND and whose 2^h leaves are assigned unknown binary values. The value of the tree is recursively defined as the NAND (where $x \operatorname{nand} y = \operatorname{not}(x \operatorname{and} y)$) of the values of its two subtrees. We want to compute the value of tree while inspecting as few leaves as possible.

► ***Question* 1.1)** *Compute the value of the tree in Fig. 1. Show that it is not necessary to inspect all the leaves to compute this value.*

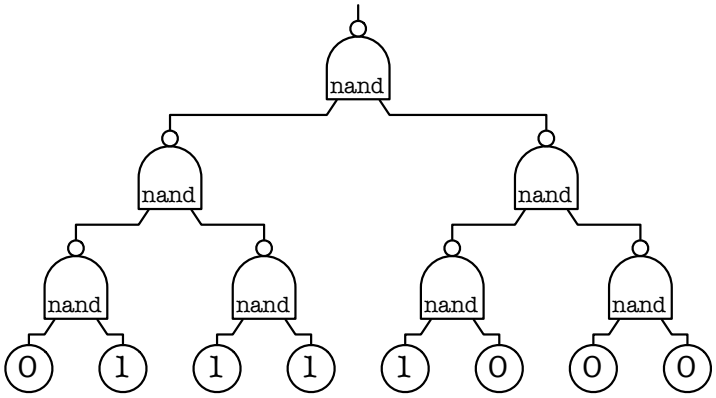

Figure 1: A nand-tree.

► ***Question* 1.2)** *Show that for every deterministic algorithm, there exists a assignment of the leaves that forces the algorithm to inspect all the leaves.*
▷ Hint. *Show by recurrence on h that an adversary can assign values of the leaves as they are inspected by the algorithm in such a way that the value of the tree depends on the value of the last leaf inspected.*

We consider the following simple randomized algorithm, which evaluates each subtree recursively in random order (either left then right or right then left) and evaluates the second one only if the value is not yet determined:

► ***Question* 1.3)** *May this algorithm fail?*

Given an assignment $\sigma : \{\text{leaves of } T\} \to \{0,1\}$, we denote by $T(\sigma)$ the resulting value of T. In order to evaluate how many leaves are inspected on expectation for the worst assignment σ, it is convenient to introduce two functions: $Q_0(h)$ and $Q_1(h)$, where $Q_v(h)$ is the

Algorithm 1 Randomized NAND-tree evaluation

```
Procedure NANDTreeEvaluation(T)
if T is a leave of value x then
        return x
else
        Let r be a uniform random bit.
        Let (T_1, T_2) = (if r = 0 then (T.left, T.right) else (T.right, T.left))
        if NANDTreeEvaluation(T_1) = 0 then
                return 1
        else
                return not NANDTreeEvaluation(T_2)
```

worst possible expected number of leaves inspected by Algorithm 1 on a tree of height h over all assignments σ such that $T(\sigma) = v$. The expected number of leaves inspected by the algorithm for the worst assignment σ is then $\max(Q_0(h), Q_1(h))$.

► ***Question* 1.4)** *Show that for all $h \geqslant 1$, $Q_1(h) = Q_0(h-1) + \frac{1}{2}Q_1(h-1)$ and $Q_0(h) = 2Q_1(h-1)$.*
▷ Hint. *Remark that the algorithm take advantage of the fact that: if the tree T evaluates to 1 then at least one of its subtrees evaluates to 0 which is favorable; and if T evaluates to 0, then both of its subtrees evaluate to 1 which are both favorable.*

Then, for all $h \geqslant 2$, $Q_1(h) = \frac{1}{2}Q_1(h-1) + 2Q_1(h-2)$. Since $Q_1(0) = Q_0(0) = 1$, and $Q_1(1) = 3/2$, it follows that (admitted): $Q_1(h) = \frac{33+5\sqrt{33}}{66}\alpha^h + \frac{33-5\sqrt{33}}{66}\beta^h = \Theta(\alpha^h)$, where $\alpha = \frac{1+\sqrt{33}}{4} \doteq 1.686$ and $\beta = \frac{1-\sqrt{33}}{4} \doteq -1.186$. If $N = 2^h$ denote the number of leaves, the randomized algorithm inspects thus $\Theta(N^{\log_2 \alpha}) = \Theta(N^{0.753\ldots}) \lll N$ leaves on expectation in the worst case. The randomized algorithm is then much more efficient than the deterministic algorithm in the worst case.

We will now use Yao's principle to prove that this algorithm is indeed to optimal among the randomized algorithms that evaluate the trees recursively (which is in fact optimal for nand-trees — admitted). Recall that Yao's principle states that "*the expected value of a randomized algorithm on its worst instance equals the worst possible expected value of the best deterministic algorithm for the worst distribution of instances*".

In order to get a lower bound that matches our algorithm, we need to force the deterministic algorithm to look half the time at both children of a node that evaluates to 1. So, rather than choosing the values independently, we build our instance from the top down. We start by flipping a coin and setting the root to 0 or 1 with equal probability. Then we descend the tree, determining the values of the nodes at each level. If a node gets value 0, we set both its children to 1; if a node gets value 1, we set one of its children to 1 and the other to 0, flipping a fair coin to decide which is which. Then the truth values of the nodes are correlated in such a way that the easy case never happens.

► ***Question* 1.5)** *Show that any deterministic algorithm that evaluates the tree recursively (i.e., which computes the value of one subtree before inspecting the leaves of the other subtree) inspects at least $\Omega(\alpha^h)$ leaves on expectation for this distribution of instances.*