The 0-1- principle

"A tool for building / proofing sorting algorithms"

by

Stefan Schwetschke

stefan @ schwetschke.de

Based on papers / books I ♥:

[Knu73] Donald E. Knuth: The Art of Computer-programming, Vol 3 - Sorting & Searching, Addison-Wesley (1973)

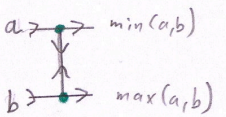
[Lei92] Frank Thomson Leighton: Introduction to parallel algorithms & architectures: arrays, trees, hypercubes, Morgan Kaufmann (1992)

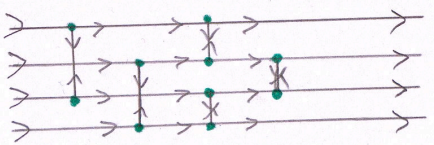
Prerequesites

- sorting network [Knu 73]

  A method to sort a fixed number of input values using a fixed order of compare-exchange (*) operations

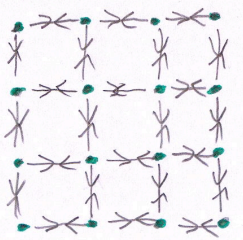
*) compex





- oblivious sorting algorithm [Leig2]

  An algorithm consisting of prespecified (compex) (**) operations. Which cells are compared in each step may not depend on the results of the previous operations. The result of each operation may only depend of the content of the compared cells.

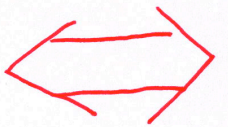


**) more general operations are allowed in certain extensions of the concept

The 0-1-Lemma

For every sorting network / oblivious sorting algorithm

If there is an input sequence that the algorithm doesn't sort correctly, then there is also an input sequence consisting only of the two values **0** and **1** that the same algorithm[*] also doesn't sort correctly.



If the algorithm sorts every input consisting only of the values **0** and **1** correctly, this algorithm sorts <u>every</u> input correctly

*) if needed: replace compex operator so that "$0 \leq 1$"

$\neg B \Rightarrow \neg A$

$A \Rightarrow B$

Sketching a proof

- Part I: Some tools -

1) Def.: $f: A \xrightarrow{f} B$ is called monoton, if A is totally ordered regarding "$\leq$" and if $\forall a,b \in A$:

$a \leq b \Rightarrow f(a) \leq f(b)$

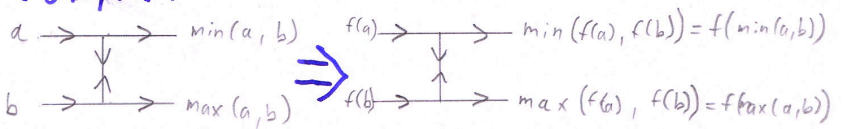
→ f preserves the ordering regarding "$\leq$"

This leads to the following conclusions:

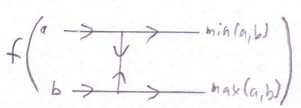
2) $f(\min(a,b)) = \min(f(a), f(b))$

$f(\max(a,b)) = \max(f(a), f(b))$

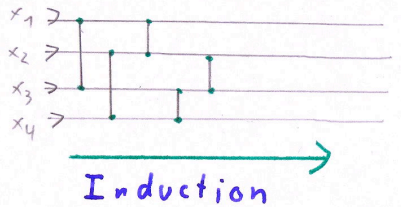
3) Compex:



→ Works "the same" for a, b and $f(a), f(b)$



4) Sorting network:



→ Works "the same" for input $x_1, x_2, \ldots$ and $f(x_1), f(x_2), \ldots$

5) Oblivious algorithms

→ Works "the same" ...

But proof can be a little bit "messy" or very "technical"

Sketching a proof

– Part II: Putting the pieces together –

$\neg B \Rightarrow \neg A$

We assume for our sorting network or oblivious sorting algorithm there is some input sequence

$$x_1, x_2, \dots, x_i, \dots, x_j, \dots, x_n$$

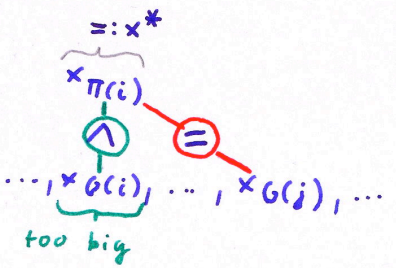
which should be correctly sorted as

$$x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(i)}, \dots x_{\pi(j)}, \dots, x_{\pi(n)}$$

but our sorting algorithm sorts it to the incorrect order

$$x_{G(1)}, x_{G(2)}, \dots, x_{G(i)}, \dots, x_{G(j)}, \dots, x_{G(n)}$$

with



We define:

$$f(x) = \begin{cases} 0, & \text{if } x \leq x^* \\ 1, & \text{otherwise } (x > x^*) \end{cases}$$

Important: f is monotonic

$\Rightarrow$ Algorithm behaves the same for input sequence

$$f(x_1), f(x_2), \dots, f(x_i), \dots, f(x_j), \dots, f(x_n)$$

$$\downarrow$$

$$f(x_{G(1)}), f(x_{G(2)}), \dots, f(x_{G(i)}), \dots, f(x_{G(j)}), \dots, f(x_{G(n)})$$

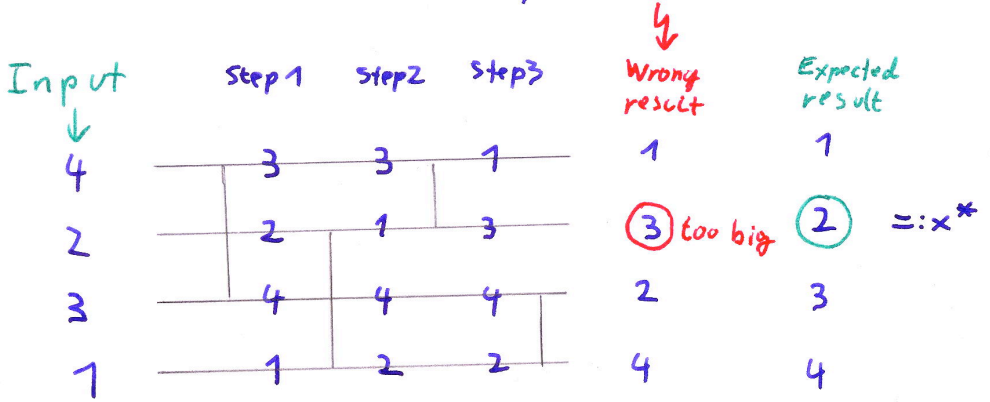
$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$0, 0, \dots, 1, \dots, 0, \dots,$$

$\Rightarrow$ 0-1-sequence $f(x_1), \dots, f(x_n)$ is also sorted wrong $\lightning$

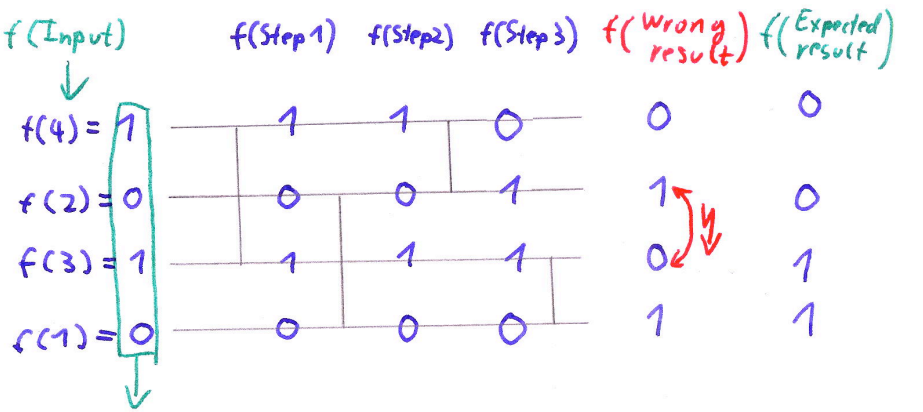
Example

→ Given: A sorting network that has a bug: it sorts incorrectly



$$f(x) := \begin{cases} 0, & \text{if } x \leq 2 \\ 1, & \text{otherwise} \end{cases}$$

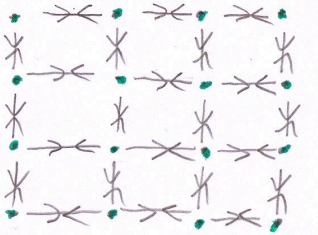
x	1	2	3	4
f(x)	0	0	1	1



We constructed an input consisting of 0 and 1, that also leads to a wrong result

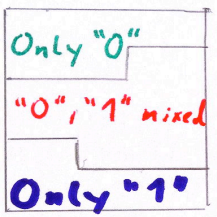
Applications in parallel sorting

0-1-Lemma motivates some very useful concepts



} Sorting grid (→ e.g. on a GPU/ graphic card)

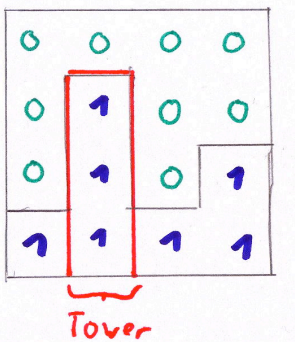
Dirty region



} dirty

→ Example: Share sort

Towers



→ Example: Schnorr - Shamir - Sort

Grid: $3\sqrt{n} + o(\sqrt{n})$

Torus: $2.5\sqrt{n} + o(\sqrt{n})$