# Childhood theory

Oleg Telyatnikov

2019-04-06

theory of reality-based computations
capdoki@gmail.com

## Contents

# 1 Introduction

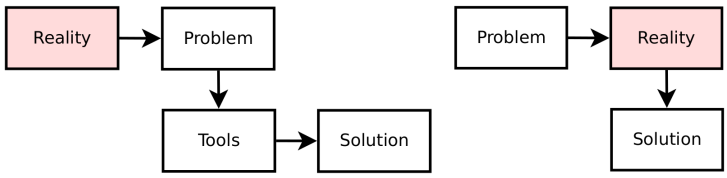
**Looking back at our childhood...** Remember yourself as a kid.

I guess, it was a great time, a lot of imagination were involved in everything, since you had lack of rational explanations of the world, and a lot of time you've spent on games, probably. It's very important for further understanding of this research paper, so I want to return you to this brilliant era of youth for a bit.

Why? Because games can be very important, they have a very interesting side - you build your own realities while playing, with laws coming directly from your imagination, such an impressive skill!

This can help you to step into wonderful world of non-Turing computations and probably can become a solution to one of Millenium problems with help of fastest pathfinding algorithm. Queer!

**Different approach** Imagine, that you can ask reality to work for your needs. Sounds interesting? I want to introduce way how you can do it on your own.




(a) Usual way of solving. (b) Reality-based way.

Figure 1: Different approaches.

Well, it includes some interesting changes in perspective. Originally we use tools to solve problems of reality (fig. 1a), but this time we will try using reality to answer questions, stated by some problem (fig. 1b).

Imagine some problem you want to solve, for example we would take problem of solving sudoku.

At the start you have rules and fields, which are following the rules. How do you want to solve it? Usually, you can suggest some algorithms, but let's try different way, according to idea, named Childhood theory[1].

On next pages base of this theory would be described, as well as realisation of this particular example.

[1] Obviously, solutions based on this theory have a lot of common with games, dive into constructed worlds for instance, you can think about it as of grown-up game, if you wish, simply with quite advanced tools.

**Main concept** Sudoku - is a system, our problem (problem of solving sudoku) described inside of this system and defines it, so the idea - is what every system has its own reality, as showed here. (fig. 2)

If problem exists - it exists somewhere, attached to something (system), what has rules. And problem defines some initial parameters of our system, so we also have a state. At this point we can perform natural transformation, presuming, that using state and rules we can describe a reality.

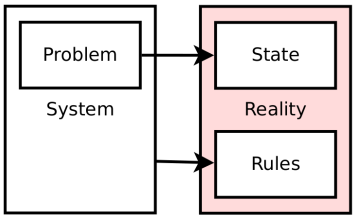



Figure 2: Relationships between problem & reality.

Probably we can insert system inside reality's block, showing how deep their dependency is, but for our needs it's not necessary and it's even more comfortable to leave system outside, so it wouldn't steal our attention from reality, which is more important here.[2]

And what we've got from this constructed reality? At this point it may not make any sense, but there is a major idea, which will make everything clear.

Take a look back on traditional Turing computers, what they do - they work with a little part of information at once, but when you are looking at some problems - like labyrinth in book for children, you probably work with great part of information at the same time, giving answer almost immediately, just with one glance you are able to load system's state and lead the way through.

If you have image of labyrinth in your mind, you have some sort of model, which is made of possible solutions. (let's presume, that an "abstract" of something, which describes how thing works, has every possible parameter and state of a system, theoretically - informational field of solutions)

Since you have this informational field, all what you need - is to find some way to extract answer in which you are really interested, we would use some kind of probes for it, like in chemistry (indicators) or physics (probe charge), and as we work with reality - it has it's own laws (or rules, it's same thing for us), which are forming probes for us. (fig. 3)

Let's imagine, that you have bottle of unknown liquid (which would represent informational field of our reality) and you want to check presense of some element inside it. Instead of checking every atom inside this bottle you can apply indicator to it, which reacts to targeted element, extracting needed answer in evident form of color. So far so good!

[2]If it sounds not enough familiar, think about it for some time, these principles are very important for further understanding.

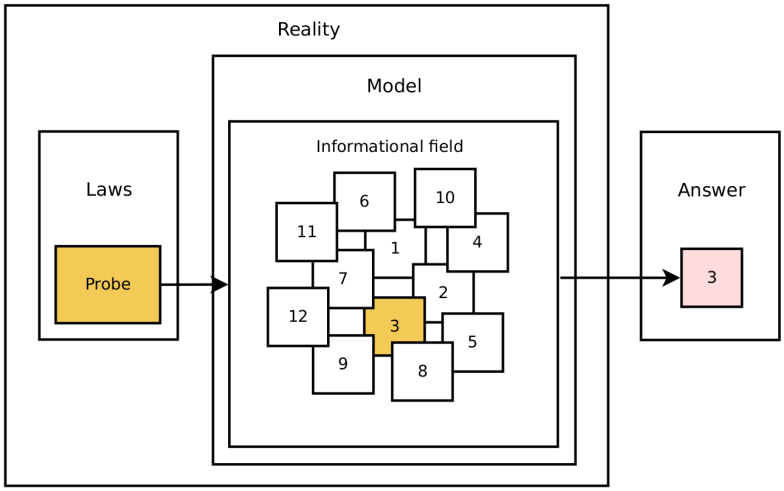



Figure 3: Interaction with reality.

## 2 Actual implementation

**An example** In case of sudoku, you can construct a reality of fields, they obey rules of being right (rows and colums have non-repeating numbers in them, simplified), as you done that, you have a reality of a problem, a structure, which has all answers for all possible questions.

All you need - is to ask a certain question to your constructed reality, or in other words - you need to use a probe, by which you can interact with your reality, then system would give appropriate solution to initial inputs of unsolved sudoku, which you defined.

Let's move to next iteration of understanding our sudoku model.

**Essence of sudoku's reality** Basically, by giving "one row/column can't have two numbers of same type" property to each segment of our model, we developed laws of sudoku's perception, generation rules and life cycle of components, very similar to laws of physics in our present reality, in which you are reading this paper right now. As we have done it, we have an uncertain reality, like our world exactly before creation, if it took place in history.

Now reality tries to reach stable state in which there will be no violations of rules and if we assume, that it takes zero time for each segment to switch from one state to another, it reaches stable and certain state very quickly.

We would swap some of components to a constant. As you know (or as you can predict and imagine) probability, that system would move from unstable state to a stable is much higher, than same thing reversed, based on use of Second Law of thermodynamics, reality of sudoku would evolve to a something stable, which also is a solution for our question.

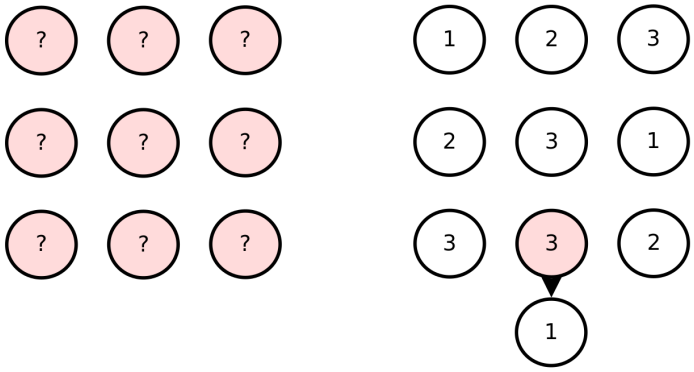



Figure 4: Uncertainty and probable way how enthropy increases.

**Implementation** Of course, it all sounds well, but we need a proof, that it all really works.

We would use simplified version of task, so we won't use 9x9 sudoku with numbers from 1-9, rather we are going to work with 3x3 version and three types of numbers, encoded binary way:

$$\begin{array}{lll} 01 & (equals & 1) \\ 10 & (equals & 2) \\ 11 & (equals & 3) \end{array} \begin{bmatrix} 01 & 10 & 11 \\ 11 & 01 & 10 \\ 10 & 11 & 01 \end{bmatrix}$$

Let's see, which behavior each of segments must have to correlate with rules of playing sudoku. Because we don't want to check diagonals, we would check only rows and colums. So each element would check values of elements in its row/column, like this:

| $elementA$ | $elementB$ | $-> elementC$ |
|---|---|---|
| 11 | 10 | 01 |
| 11 | 01 | 10 |
| 01 | 10 | 11 |

where "elementA", "elementB" and "elementC" are elements of one row/column.

Hopefully, it doesn't require very complicateted logic beneath, I used NAND elements, working with high and low bits separately. As you can see,

- NAND(1,1) equals 0
- NAND(1,0) equals 1
- NAND(0,1) equals 1

what is actually what we need. But our schematics can output quite strange numbers at the start, so we need to make this states unstable and then they would perish after some time, because system in general tries to reach stable state and probably not interested in this kind of things.

| $elementA$ | $elementB$ | $-> elementC(nand)$ |
|---|---|---|
| 11 | 11 | $00(stable)$ |
| 01 | 01 | $10(unstable)$ |
| 10 | 10 | $01(unstable)$ |
| 00 | 00 | $11(stable)$ |
| 00 | 11 | $11(stable)$ |
| 00 | 01 | $11(unstable)$ |
| 00 | 10 | $11(unstable)$ |

Easiest way (and probably one of the most efficient ways to minimize number of wrong stable states, while working with high and low bits them separately) is to add an implication to the output of higher bit of each segment, so 00 state would be impossible, because every 00 state would transform to 10 (what later leads to combinations of numbers without rules violation), leaving to us only one wrong stable state, 11 11 10 (and all transpositions: 11 10 11, 10 11 11).

**Schematics** As board is pretty complex, it was devided onto 3 different boards, 2 from them are standing for rows and colums (fig. 5), mounted then on some kind of motherboard (fig. 6), which is stands for rows/colums interaction and control over segments, giving opportunity to inderect changing of their values[3].

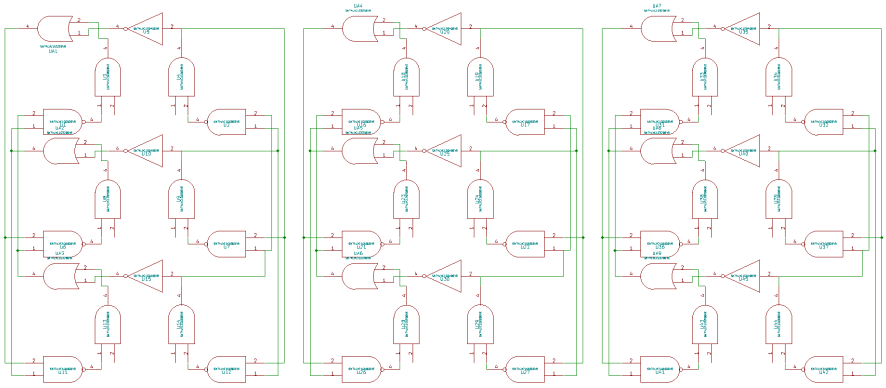


Figure 5: Rows/Columns board.

[3]What is particulary interesting about inderect interaction is that you change values through AND gates on Rows/Columns board, persuading another elements in the row/column what this element has different value, by changing one of bits to zero, and then output of NAND gate (which is used like current state of sudoku's segment) changes to another value, because it hasn't got different choise, since another elements were changed accordingly to what they've seen after AND gate.

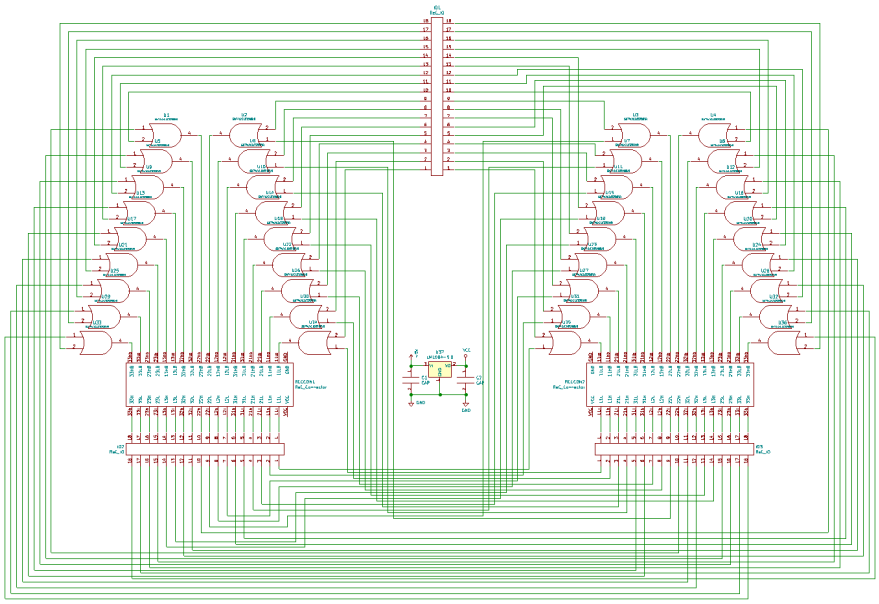


Figure 6: Motherboard.

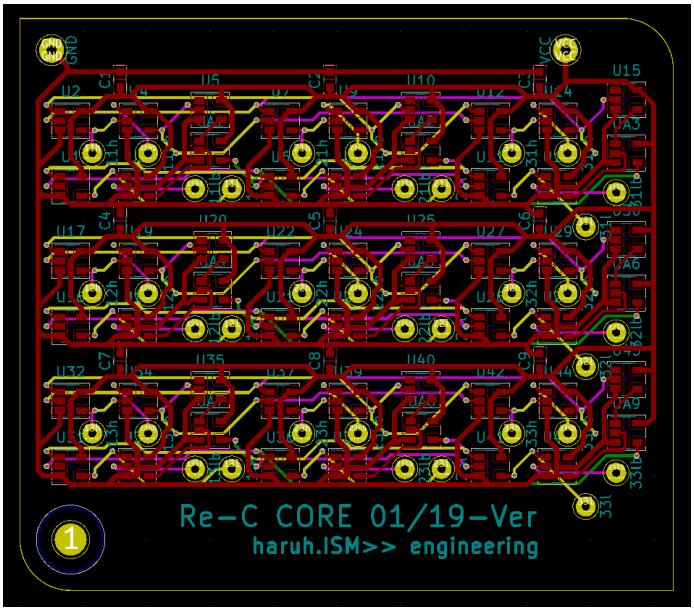



Figure 7: Rows/Columns board PCB.

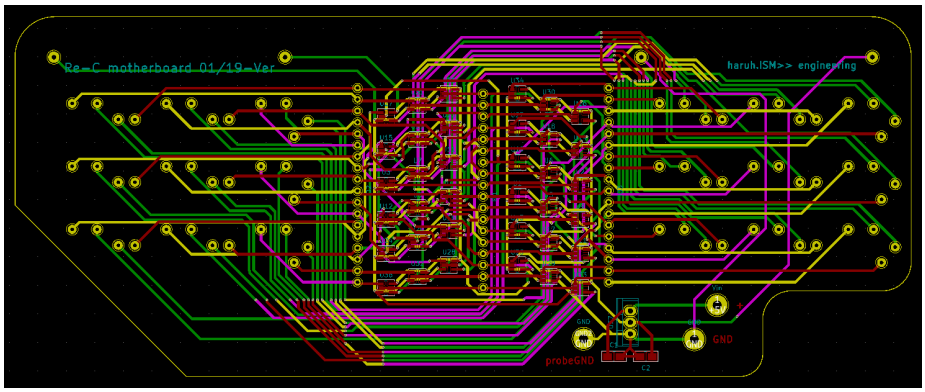



Figure 8: Motherboard PCB.

How logic elements work here, briefly:

- NAND is most crucial part here, as showed before.
- AND watches, what number on rows and colums are equal.
- NOT and OR - just an implication, unstabilizing wrong states.

Motherboard links bytes of rows/columns through AND gate, one input of AND gate is a control input, logic LOW means what this byte can only contain 0, with logic HIGH byte can contain 1 and 0 as well. Let's head to results, since basic principles of this were described before and this schematics is only an implementation of them.

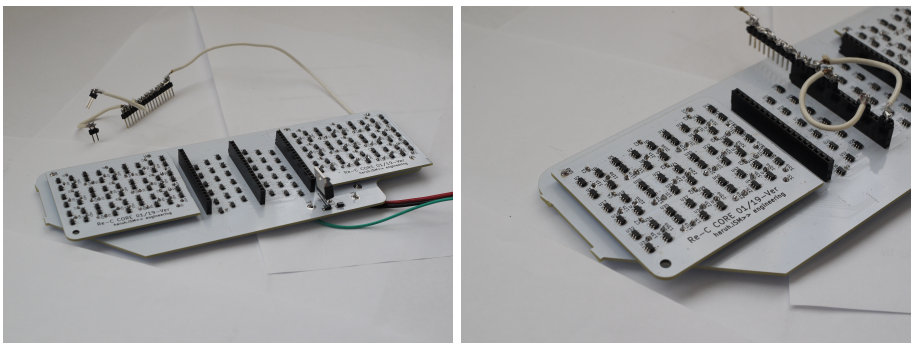


Figure 9: Complete build.

If you want to put it together too, you can download Gerber files for PCB manufacturing here: Archive on Google Drive

**Results** Hooray! It solves everything as needed, so there is no problem with what. Some outputs:

$$\begin{bmatrix} 11 & 10 & 01 \\ 01 & 11 & 10 \\ 10 & 01 & 11 \end{bmatrix} \begin{bmatrix} 01 & 10 & 11 \\ 10 & 11 & 01 \\ 11 & 01 & 10 \end{bmatrix}$$

However, I was surprised, that it likes awkward states very much, for example: it seemed almost impossible for me, what it would evolve into state, where there would be rows/columns with 11 11 10 stable state (only one possible wrong state described before), because to maintain stability it would require at least two rows like this, but it did it pretty frequently. Another interesting thing was this situation: I wanted to see how it would behave with one of bytes changed externaly, changing high byte which showed in brackets here:

$$\begin{bmatrix} (0)1 & 11 & 10 \\ 10 & 01 & 11 \\ 11 & 10 & 01 \end{bmatrix} \begin{bmatrix} (1)1 & 11 & 10 \\ 10 & 01 & 11 \\ 11 & 10 & 01 \end{bmatrix}$$

As you can see, there is no change in other numbers! But why? Well, it evolved into state, where changed byte doesn't defines anything because 11 11 10 (and all possible transpositions of that) also seen like stable state, and we have this in first row and first column as well. Board done it each time I tried to do this. Lazy thing! Persuading it to do something requires more bytes to change...

Then, what about perfomance? Well, it's not so easy to tell, since it's hard to compare it to traditional computer but let's try! Here we have nice logic gates with switching time between 0.8 and 4.7 nanoseconds at 3.3V VCC. So, as we have rows & colums, signal goes from one board to another and then spends some time on NAND-AND-NOT-OR sequence, in total our board reacts to change after switching time multiplied by 5. To be fair to the maximum level, we would take 0.8 nanoseconds switching time, so fastest possible reaction time now is 4 nanoseconds, what we can somehow compare to one tact of traditional processor, as it determines operation speed.

It was a failure to try to measure speed with 100MHz MCU, so I used oscilloscope for that purpose (fig. 10), bandpass isn't so great, but we can presume, what one part (highlighted in red) is a transition process and another part is a converging oscillations. As you can see (fig. 10) it takes only 56 "tacts" to give answer, that is actually pretty quick! For example, you need at least 18 tacts to check, what it solved correctly, comparing numbers two times in each row/column and writing result into some register of your MPU. If medium switching time wasn't so good as we assumed before (for example, 2 ns), it's even more impressing.

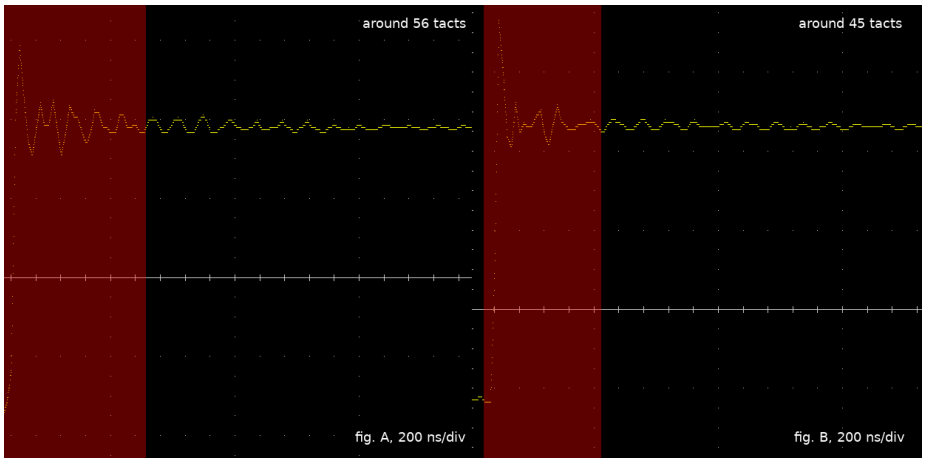



Figure 10: Unstable to stable transition.

Of course, this perfomance measurements are only for the sake of curiosity, if you really want to see how it stands against more traditional methods of computation. By the way, it's too different from everything to be compared: it has different structure, behaves as it wants and even more! However, results are good, it works right after assembly and works quickly.

# 3 Different ways

**Sudoku and similar** Sudoku is just an example, every task where you have some target and you want to find parameters which are leading to that results, you can make a reality, where this target would evolve to your complete answer. So, this approach is very good at generation of predicted outputs for unpredicted inputs, when you don't know how to make it via algorithms fast enought.

**Pathfinding** Of course, there is a lot of ways how to construct reality, which are helpful in solving another kinds of tasks, as an example we would take pathfinding, it's a hard algorithmic task and has a lot of practical use.

At start, we have a number of verticles, which we are going to recreate in our reality, they have edges connected to them, with some weight (as we are talking about pathfinding, we would't think about directional edges, but we can assume further, that it's not a problem for this method anyway)

Since we have some abstract map of labyrinth, we have all solutions and all shortesth paths in one place, all we need - is to dig through it, using laws of reality, and under using laws we mean, what we are using properties of reality to solve our task. At this point it's very important to understand, what it doesn't matter, which properties of which reality we are using, but it's very helpful to use real world, because you don't need to recreate rules on your own and you use something, that works as fast as time flows (delay in used components, if

you want to create your own rules, defines a speed on which your handmade reality works, and as components are usually not as perfect as you can wish, it will affect your reality badly)

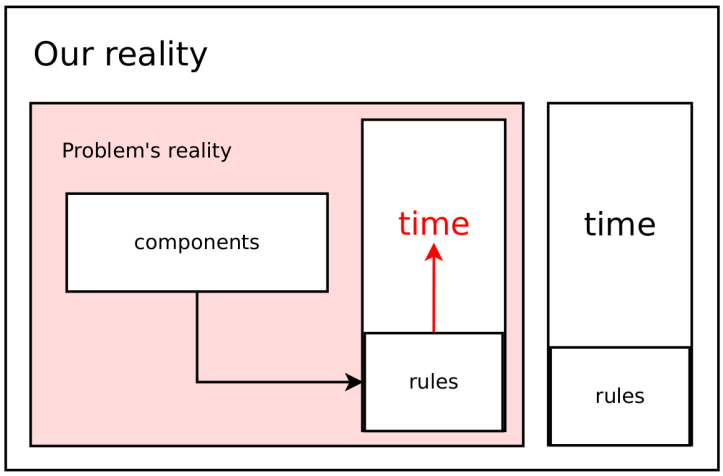
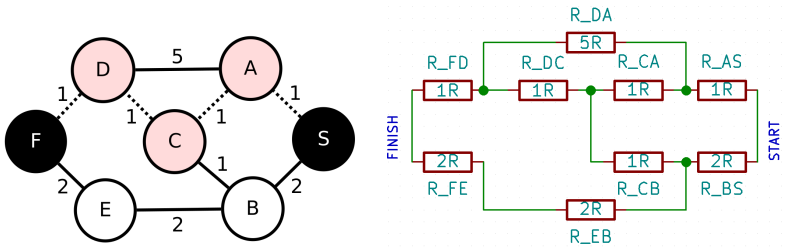



Figure 11: Dependency between components and timeline.

Let's do some natural moves (probably, they are not so evident, but they have logic beneath), firstly encoding weights into resistance, so now, when all resistances are connected according to map of our labyrinth, we have a model.It's time to inject information about our desired way through. We would apply probe (electric potential) to extract specific information. Current flows through every path from start point to end point, last thing we need to do - is to find which of this ways is shortest. Amazing thing! We really have rule in our world, which finds shortest ways, if you would apply electric potential to conductor in two points, biggest current would flow through path with lowest resistance, it would find a way for you, spotlighting shortest path like this:




(a) Graph. (b) Graph's schematics.

Figure 12: Graph to circuit conversion.

Shortest path here (fig. 12a) is S-A-C-D-F ("S" and "F" stands for "START"

and "FINISH"), after applying voltage to START and FINISH nodes (fig. 12b), you can recreate shortest path precisely, "walking" by the way of highest current:

1. We are "standing" at the starting point
2. Current flowing through R_AS is higher than current through R_BS
3. We are moving along R_AS to next node
4. Current flowing through R_CA is higher than current through R_DA
5. We are moving along R_CA to next node
6. Than, basing on direction and amount of current flowing through R_CB, we deside between R_CB and R_DC, choosing R_DC
7. We are moving along R_DC to next node
8. Than, as on step 6, basing on direction and amount of current flowing through R_DA, we deside between R_DA and R_FD, choosing R_FD, obviously
9. We are at the FINISH node now, so our shortest path is R_AS - R_CA - R_DC - R_FD

What we done there, by "walking", is just recreation of solution in useful form of storaging information, answer for our question existed even before that, without inductance it happens wery fast.

And of course it doesn't mean, that it's impossible to work with very big graphs using this method, we can compress them like showed here:

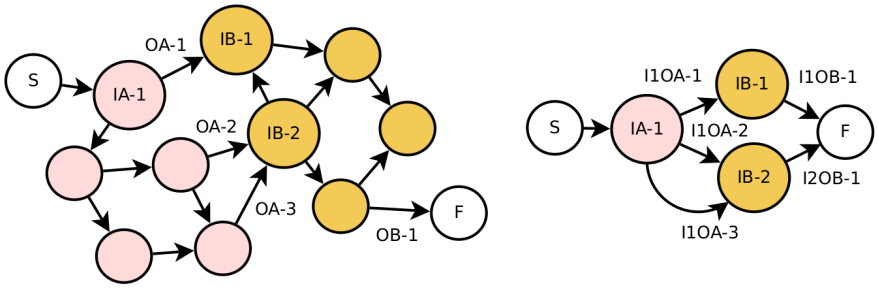



Figure 13: Graph compression.

By dividing original graph onto sections, A (pink) and B (orange), we can transform inputs of each section (IA-1, IB-1, ...) into verticles of compressed graph (IA-1, IB-2, ...) and shortest paths from inputs to outputs (OA-1, OA-2, ...) into links (I1OA-1, I1OA-2, ...)[4]

[4]IXOS-Y, where X - input number, S - section letter, Y - output number

Firstly finding shortest paths between inputs and outputs of sections, secondary we can proceed everything on compressed graph, unrolling every compressed part of path afterwards. If it's necessarily we can even compress everything two and more times!

**Millenium problem?** Where is some questions around there, named "Millenium problems". One of them - is a question, stated like "P versus NP". May time needed to solve task be lesser than time needed to check what task solved correctly? Definite answer to this question is very important to cryptography, as to many another fields too, because it would mean, what some things can be solved on very extraordinary speeds. And it seems, what Childhood theory, as a different approach to solving, possibly could give this speeds on some tasks, proving, what it's possible.

Then, answer to one of Millenium problems depends only on how people would estimate answers given by this method: if they don't have to be checked - $P = NP$, otherwise time to check answer by traditional methods would be even greater than time to solve this way, because to get answer on Pathfinding example, you only need to walk through shortest path once, and to prove what it's shortest - you definitely need to compare it's lenght to lenghts of every path you have, so it's evident what you would need to make more steps to prove, than steps to solve (to get answer from the system), since it's not necessarily to even visit every verticle of your graph to get shortest path by this method.

# 4 Summary and thoughts

**Method** You can create your own machines following simple sequence:

1. Create reality of your task
2. Find a proper probe
3. Interact with reality using this probe
4. Load answer somehow (using Turing-type computer, for example)

Of course it requires skills from many different fields at that moment, we don't have any kind of universal machine, which can easily configure realities for your needs similar to programming of traditional computer, but as showed - it's not that hard to develop certain implementations, it works and it can be very useful.

**Philosophy** Machines based on this theory - is the answer generating machines, which can work with any type of data. Principles lying under them - working with informational field, which contains answers, so if you know how to dig through it - you would succeed. (of course, if reality is made properly)

Answer - is a state of problem's reality, which corresponds to all reality's laws. There is some very cute philosophy about that - digging into reality in search of answers, I personally really like it :)

**Thoughts** What is all that exactly?
If you would look closer to Turing computer, you would definitely discover something common, processor - is like a reality, but a small one - which gives you a predictable output for your input, based on laws of math and logic, you are sending inputs in needed order via program. So, Childhood theory-based computer is a computer working with information in more general sense. Probably, most universal storage and processor for information - is a reality, and this is a step forward to make use of it.

**Future** And what about future of this theory?
Very good question! All in our hands, literally. Of course, there may be plenty of greater applications than described here, but it all requires further research. If science community would find it useful - pretty sure, what everything would evolve in numerous great tools. Time'll show everything.