# Analysis: GAT
# Gates

**HISTORY:**

dokument systemu SINOL 1.7.2

## 1 Problem analysis

In task "Gates" our goal is to find such a configuration of switches that closes all channels. If such a configuration exists we have to find a state of every switch in such valid configuration.

Let us denote by $a_i$ the sentence "Switch number $i$ is on.". We can think about $a_i$ as a logical variable (whether the sentence is true or false). Using this approach, all we have to do in this task is to find valuation of these variables fulfilling task conditions. Moreover we can easily transform the input file to the language of logic. I will show it using the example from the task:

3 2
1 0 2 1
1 0 2 0
1 1 2 1

In this example we have two switches. As a result, we will have two variables $a_1$ and $a_2$ respectively. If we look at the channel number one we will see that: switch number one has to be closed or switch number 2 has to be opened. Similarly we can transform next two channels. We obtain logical formulas:

- Channel 1: $\neg a_1 \vee a_2$
- Channel 2: $\neg a_1 \vee \neg a_2$
- Channel 3: $a_1 \vee a_2$

Since we are looking for a configuration that closes all channels we have to make a conjunction of mentioned logical formulas:

$$(\neg a_1 \vee a_2) \wedge (\neg a_1 \vee \neg a_2) \wedge (a_1 \vee a_2)$$

Our task is to find a valuation of variables $a_1$ and $a_2$ that satisfies formula above. In general we have to find a valuation of variables $a_1, a_2, \ldots, a_m$ satisfying a certain logical formula. In general problem of satisfying a given logical formula is well known to be NP-complete. Fortunately, the formula obtained in this task is of a very special form:

$$(x_1^1 \vee x_1^2) \wedge (x_2^1 \vee x_2^2) \wedge \ldots \wedge (x_n^1 \vee x_n^2)$$

where $x_i^1$ and $x_i^2$ are literals and each of them stands for some variable $a_j$ or negated variable $\neg a_j$. The literals $x_i^1$ and $x_i^2$ correspond to the conditions of $i-th$ channel. The formula is a conjunction of alternatives of exactly two literals. Therefore is said to be in the $second\ conjunctive\ normal\ form$ (2-CNF). We shall show that this problem can be solved in a polynomial (even linear) time.

For the rest of this problem discussion I will assume that our goal it to find correct valuation of variables $a_1, a_2, \ldots, a_m$ in formula:

$$(x_1^1 \vee x_1^2) \wedge (x_2^1 \vee x_2^2) \wedge \ldots \wedge (x_n^1 \vee x_n^2)$$

## 2 2-CNF formulas and graphs

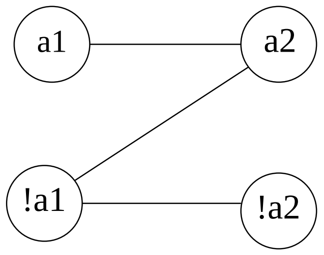
Let us consider an undirected graph $G = (V, E)$ with vertices corresponding to all possible literals:

$$V = \{a_1, \neg a_1, a_2, \neg a_2, \ldots, a_m, \neg a_m\}$$

and edges connecting pairs of literals which appear in alternatives of the formula

$$E = \{(l_i^1, l_i^2) : i = 1, 2, \ldots, n\}$$

Graph $G$ from our example is shown on the following picture.



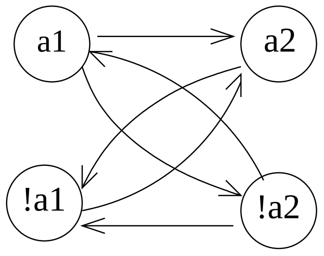
Our goal is to select a subset $W \subset V$ containing vertices corresponding to these literals which are true when the formula is satisfied. For that the following conditions have to be satisfied:

1. either $a_i \in W$ or $\neg a_i \in W$ (exclusively), for $i = 1, 2, \ldots, m$
2. for any edge $(u, v) \in E$, $u \in W \vee v \in W$

To do that let us construct directed graph $G_1 = (V, E_1)$ using definition of graph $G$:

$$E_1 = \{(\neg u, v) : (u, v) \in E \vee (v, u) \in E\}$$

We will call it the *inference graph*, because it can be used to find which vertices one has to choose to the set $W$, provided that some given vertex has already been chosen. The inference graph has $2m$ vertices and at most $2n$ edges. The inference graph of $G$ is shown below.



Looking at this graph it is easy to notice that every $W \subset V$ which is a correct solution of our problem must satisfy the condition:

$$(w \in W \wedge (w, v) \in E_1) \Rightarrow v \in W$$

This logical formula is equivalent to the previous condition number 2. As a result we can search for $W \subset V$ using inference graph and not contradicting the exclusive condition number 1.

Let us denote

$$Induced(u) = \{v : u \mapsto v\}$$

where $u \mapsto v$ means that there exists a path from vertex u to vertex v in graph $G_1$. Using this definition we can easily rewrite previous condition as:

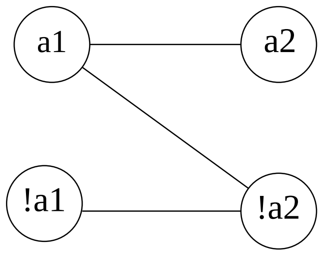
$$w \in W \Rightarrow Induced(w) \subseteq W$$

If a set $Induced(v)$ contains two opposite vertices $(w \in Induced(v) \land \neg w \in Induced(v))$ we will call vertex $v$ problematic.

Let us introduce another kind of undirected graph - conflict graph. Using the second condition we can say that for every edge $(u,v) \in E$ of graph G it is not possible to $\neg u$ and $\neg v$ both be in set $W$. We can say that these vertices are in conflict. Therefore, we can build a conflict graph $G_2 = (V, E_2)$ where:

$$E_2 = \{(\neg u, \neg v) : ((u,v) \in E) \lor ((v,u) \in E)\}$$

Conflict graph for our example is shown below.



Using these definitions and facts we can show a solution for the problem.

## 3 Solution

### 3.1 Naive solution

The first naive solution is to solve this problem using backtracking. In the algorithm for each vertex which is yet neither accepted nor rejected we try to decide if it is possible to find a solution with this vertex accepted and if it is not if it is possible to reject this vertex. Unfortunately the time complexity of this algorithm is exponential, so it is not an efficient solution.

### 3.2 Polynomial solution

We will try to show better algorithm. Using previously introduced definitions we can solve this problem using the following algorithm:

1. $W = \emptyset$
2. while $|W| < m$ repeat
    - let $x$ be the vertex that $x \notin W$ and $\neg x \notin W$
    - if both $x$ and $\neg x$ are problematic then "SOLUTION DOES NOT EXIST" (stop the algorithm)

- let $v$ be a non-problematic vertex $x$ or $\neg x$
- $W := W \cup Induced(v)$

3. $W$ is correct solution for our problem

It is not obvious that this algorithm really returns a correct answer. To see that let us show the lemma.

***Lemma***
Let $A$ be a set of vertices that neither they nor the vertices opposite to them are in the $Induced(v)$. If the vertex $v$ is non-problematic then there is no edge in the conflict graph between A and $Induced(v)$.
***Proof***
Let $a$ be the vertex from $A$ $(a \in A)$ and $u$ be the vertex from $Induced(v)$ $(u \in Induced(v))$. If there were a conflict between $a$ and $u$ there would have to be an edge $(u, \neg a)$ in inference graph $G_2$. As a result there would have to be: $\neg a \in Induced(u) \subseteq Induced(v)$. But $\neg a \notin Induced(v)$ by the definition of $A$. ♠

This Lemma shows the correctness of presented algorithm — setting the value of a certain variable to in a way not leading to a contradiction does not affect the vertices not in $Induced(v)$ and thus leads to a proper solution, if one exists. This solution has the overall time complexity $O(m * (n + m))$ as checking if a vertex is problematic may take time proportional to the size of the graph.

## 3.3 Model solution

Let us think about strongly connected components in our inference graph. (Two vertices $u$ and $v$ belong to the same strongly connected component if and only if there is a path from $u$ to $v$ as well as from $v$ to $u$). Strongly connected components of an inference graph have a very useful property for us: for any component $C \subseteq V$, either $C \subseteq W$, or $C \bigcap W = \emptyset$. As a result, we can consider the graph of components $G_c = (V_c, E_c)$, whose vertices are strongly connected components of the graph $G_1$ and edges are inherited from that graph in a natural way. There is a simple algorithm calculating strongly connected components in a graph in time $O(n + m)$.

Obviously the graph of components is a directed acyclic graph. We shall sort it topologically and consider its vertices in not-ascending order, according to the graph of components. This can also be done in a linear time.

We say that we accept a component when that component is chosen and included to the set $W$ while performing the algorithm. Similarly, we say that we reject a component if we decide not to choose the component anyway. It is easy conclusion that if we reject a component we have to reject all its ”predecessors” and if we accept a component we have to accept all components that are induced by our component.

Still thinking about the previous algorithm it leads to another, more efficient solution of the problem:

1. read the input file and generate the inference graph $G_1$
2. find strongly connected components of $G_1$ and build a graph of components $G_c$
3. if there are two opposite vertices in a component reject this component (and all its ”predecessors”)
4. sort topologically the components, and process them in descending order:
   - if current component has not been rejected so far, accept it
   - for each vertex in the accepted component reject the component containing the opposite vertex (and consequently all its ”predecessors”)
5. if exactly $m$ vertices have been accepted, then they form a correct solution, otherwise solution does not exist.

Note that because of topological order of components each time we accept a component $C$, all components induced by $C$ have already been accepted. Indeed, if one had been rejected before, then $C$ would also have been rejected as a ”predecessor” of it. It is also clear that set $W$ does not contain any conflicts. As a result, if it returns an answer it will be correct. All we have to do is to show that if solution exists, this algorithm will find it. But it is a consequence of previous algorithm and Lemma from section ”Polynomial solution”, as we reject all problematic vertices in step 3 of our algorithm.

## 4 Tests

I have prepared 4 ”ocen” tests for competitors and 17 groups of tests designed for checking contestants solutions.

Exact description of tests is shown below. Each test is described by values of $n$ and $m$ and the number of switches which are really used. Moreover in the first two brackets there is running time of both model solutions. Explanations of used signatures are shown below.

- `gat0.IN` (0.03 sek.) (0.00 sec.) first example from the task description
- `gat0a.IN` (0.03 sek.) (0.00 sec.) second example from the task description
- `gat1a.IN` (0.03 sek.) (0.00 sec.) n=10, m=10, used=10, test P, 3 chains
- `gat1b.IN` (0.02 sek.) (0.00 sec.) n=2, m=2, used=1, test I
- `gat2a.IN` (0.03 sek.) (0.00 sec.) n=20, m=20, used=20, test P, 1 chain
- `gat2b.IN` (0.03 sek.) (0.00 sec.) n=2, m=20, used=1, test I
- `gat3.IN` (0.03 sek.) (0.00 sec.) n=20, m=7, used=7, test R
- `gat4.IN` (0.04 sek.) (0.00 sec.) n=100, m=115, used=100, test P, 30 chains
- `gat5.IN` (0.04 sek.) (0.00 sec.) n=200, m=100, used=100, test R
- `gat6.IN` (0.04 sek.) (0.00 sec.) n=500, m=500, used=500, test P, 72 chains
- `gat7.IN` (0.05 sek.) (0.00 sec.) n=700, m=1243, used=940, test R
- `gat8.IN` (0.05 sek.) (0.00 sec.) n=900, m=354, used=354, test R
- `gat9a.IN` (0.05 sek.) (0.00 sec.) n=1000, m=1500, used=1000, test P, 3 chains
- `gat9b.IN` (0.05 sek.) (0.00 sec.) n=1000, m=2000, used=1239, test I
- `gat10.IN` (0.08 sek.) (0.03 sec.) n=10000, m=37651, used=10000, test P, 2 chains
- `gat11a.IN` (0.32 sek.) (0.24 sec.) n=90000, m=40000, used=40000, test R
- `gat11b.IN` (0.55 sek.) (0.37 sec.) n=100000, m=100000, used=86402, test I
- `gat12.IN` (0.58 sek.) (0.44 sec.) n=140000, m=60000, used=60000, test R
- `gat13.IN` (0.52 sek.) (0.44 sec.) n=150000, m=160000, used=150000, test P, 6 chains
- `gat14.IN` (0.74 sek.) (0.56 sec.) n=200000, m=70000, used=7000, test R

- `gat15a.IN` (1.05 sek.) (0.72 sec.) n=250000, m=250000, used=250000, test P, 15 chains
- `gat15b.IN` (1.44 sek.) (0.94 sec.) n=200000, m=400000, used=252990, test R
- `gat16.IN` (1.12 sek.) (0.87 sec.) n=250000, m=500000, used=250000, test P, 10
- `gat17a.IN` (1.00 sek.) (0.92 sec.) n=250000, m=500000, used=250000, test P , 2 chains
- `gat17b.IN` (1.05 sek.) (0.73 sec.) n=250000, m=500000, used=500000, test H

Explanations:

- P - inference graph consisting of few chains ending with problematic vertices
- R - random test with existing solution
- I - test checking correctness of solutions - answer IMPOSSIBLE
- H - maximal test

## 5 Limits

I suggest to set about 4· running time of the slowest model solution `prog/gat1.cpp` as a time limit for this task. Using this selection, statistics turn out to be as mentioned below:

- Model solutions `prog/gat.cpp` , `prog/gat1.cpp` , `prog/gat2.pas` get 100 points.
- Slower solutions `prog/gats1.cpp` , `prog/gats2.pas` get about 55 points.
- The slowest solution `prog/gats0.cpp` gets about 16 points.
- The incorrect solution `prog/gatb1.cpp` gets about 6 points.

As a memory limit I suggest 96MB.