Algorithmic Aspects of Game Theory

Tomasz Garbus

2019

1.1 Zero-player game

Let $G = \langle V, E \rangle$ be a legal graph and $R \subseteq E$ any set of edges in G. We shall define a sequence of moves in a zero-player game starting from configuration $\mathcal{G}_0 = \mathcal{G}, R_0 = R$ recursively:

$$R_{k+1} = \{(x,a) \in (E \setminus \bigcup_{i=0}^{k} R_i) : a \in fire_{\mathcal{G}_k}(\bigcup_{i=0}^{k} R_i)\}$$
$$E_{k+1} = (E_k \setminus R_{k+1}) \cup R_{k+1}^{op}$$
$$\mathcal{G}_{k+1} = \langle V, E_{k+1} \rangle$$

where $(x,y) \in R_{k+1}^{op} \Leftrightarrow (y,x) \in R_{k+1}$ and weights are unchanged.

Consider the following problem: given a legal graph $\mathcal{G} = \langle V, E \rangle$, a set of edges $R \subseteq E$ and an edge $e \in E$, does there exist a sequence of moves that reverses e?

1. Show that the above problem is P-complete.

1. $\in$ P

The game can be simulated in a polynomial time. It's easy to see that each timestep (i.e. computing of R_k, E_k, G_k for some k) can be done in polynomial time directly from the definition. The game ends after at most $|E|$ steps (since each edge can occur only in one set R_i, we know this directly from how R_{k+1} is defined). Thus the whole simulation can be performed in polynomial amount of time.

2. P-hard

Circuit value problem (CVP) can be reduced to zero-player flow game. In CVP, we are given a boolean circuit and its input and we are asked to compute the value of its top node. A boolean circuit consists of inputs, $\vee$ nodes and $\wedge$ nodes (there are no negations, since all negations can be moved to the inputs layer using de Morgan's laws). A $\vee$ node or $\wedge$ node has a positive in-degree and its value is either alternative ($\vee$) or conjunction ($\wedge$) of its inputs.

Let's denote the provided boolean circuit as a directed acyclic graph $C = \langle V_C, E_C, r \rangle$, where V_C is the set of vertices, E_C the set of directed edges and $r : V \to \{0, 1, \vee, \wedge\}$ is a funcion mapping a node to its type. If $r(v) \in \{0,1\}$ then $deg_{in}(v) = 0$ (it's an input node). The only node with $deg_{out} = 0$ is the one for which we want to compute the value.

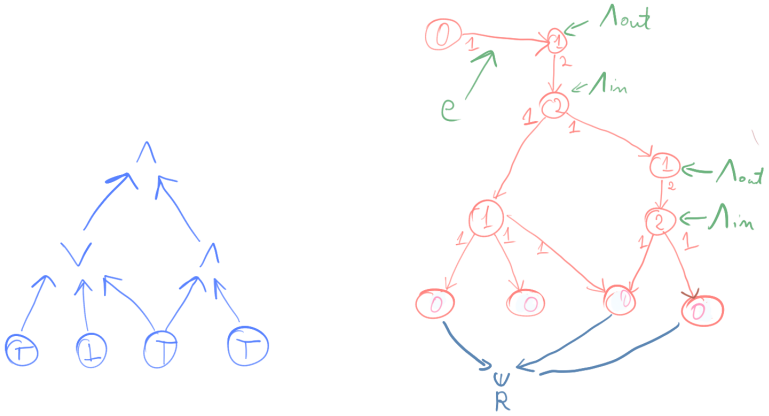
The constructed one-player flow game is $\mathcal{G} = \langle G = (V, E, w), R, e \rangle$, where:

- $V = \{v_u : u \in V_C, r(u) \in \{0,1,\vee\}\} \cup \{v_{u_{in}}, v_{u_{out}} : u \in V_C, r(u) = \wedge\} \cup \{v_{top}\}$ (we copy the input and $\vee$ nodes, split $\wedge$ nodes into two and add a dummy *top* node)
- $E = \{(v_a, v_b) : (b,a) \in E_C\} \cup \{(v_{u_{out}}, v_{u_{in}}) : u \in V_C, r(u) = \wedge\} \cup \{(v_{top}, v_u) : u \in V_C, deg_{out}(v) = 0\}$ (we reverse the edges from the circuit, add an edge from the queried node to *top* and reverse it, for each $\wedge$ node we add an edge $(\wedge_{out}, \wedge_{in})$)
- $w(a \in V) = \begin{cases} 0 & \text{if } a = v_{top} \\ 0 & \text{if } a = v_u, r(u) \in \{0,1\} \\ deg_{in}(u) & \text{if } a = v_{u_{in}} \text{ for some } u \in V_C \\ 1 & \text{otherwise} \end{cases}$

  $w((a,b) \in E) = \begin{cases} deg_{in}(u) & \text{if } a = v_{u_{out}}, b = v_{u_{in}} \text{ for some } u \in V_C \\ 1 & \text{otherwise} \end{cases}$
- $R = \{v_u : u \in V_C, r(u) \in \{1\}\}$, those inputs which are set to true
- $e = (v_{top}, v_{out})$, where *out* is the node to evaluate in CVP

Below is an example transformation of CVP to zero-player flow game.

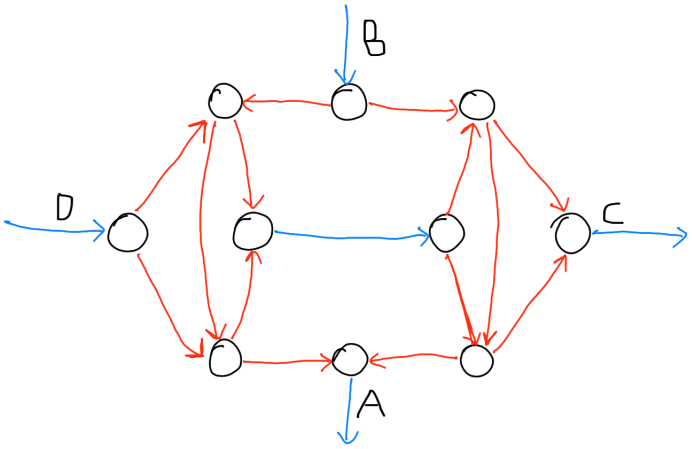
Fig. 1: An example transformation of CVP to zero-player flow game



Note that the edges coming to nodes set to false will never be reversed. True values will propagate "upwards" the graph. e will be reversed after some sequence of moves if and only if the circuit evaluates to true.

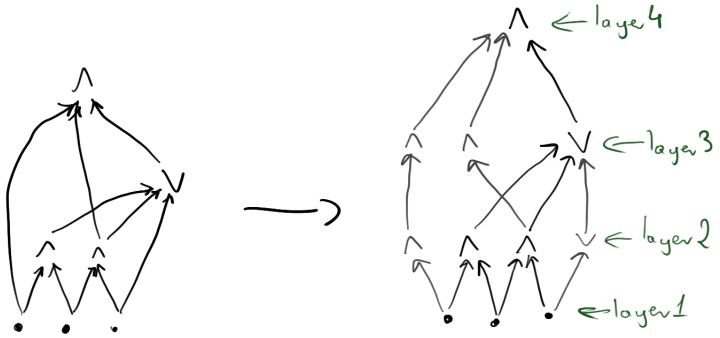
2. Show that it remains P-complete on planar graphs.
Of course it remains in P on planar graphs, so it suffices to show that it is still P-hard. Consider my solution from the first subproblem – I will show how to modify the constructed graph to make it planar.
As advised, I will use the crossover gadget. Consider the figure below, depicting the gadget.
Observation 1: The gadget, with the edges directed as shown on the figure is a valid subgraph (all vertices are firing).
Observation 2: The blue edge denoted B can be reversed if and only if A is reversed. Similarily, the blue edge D can be reversed if and only if C is reversed. Moreover, this will happen "automatically" (i.e. according to the rules of the zero-player games) – if $A \in R_k$ for some k, then $B \in R_{k+4}$. If $C \in R_k$ for some k, then $C \in R_{k+6}$.

Fig. 2: Crossover gadget with initial edges configuration. All nodes have weight 2, blue edges have weight 2, red edges have weight 1.



In order to effectively use this widget, I will put one more constraint on the input boolean circuit: all nodes are placed within "layers" and there are no connections skipping layers. This can be easily achieved by introducing dummy $\vee$ or $\wedge$ nodes with one input. See the figure below for an example of such transformation:

Fig. 3: Example transformation of the boolean circuit to get rid of skip-layer connections.

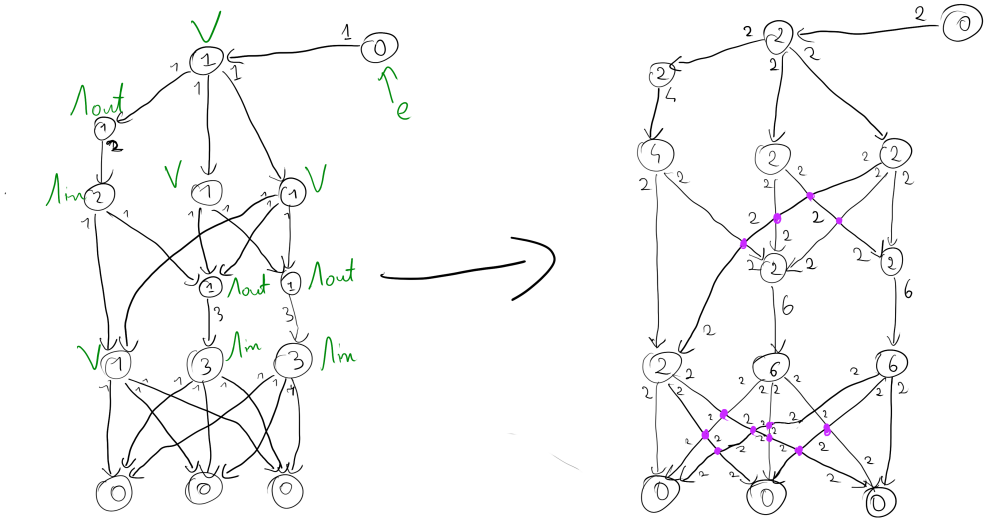


Now, having ensured that two nodes can be connected only if they are on two consecutive layers, we need to remove crossing edges. First, we multiply all weights in the graph (both for edges and vertices) by 2.
Then let's take any drawing of the graph on 2D plane such that:

- no three edges go through a single point (at most two edges may be intersecting at one point)
- two edges may intersect only if they have their ends in the same layers in graph

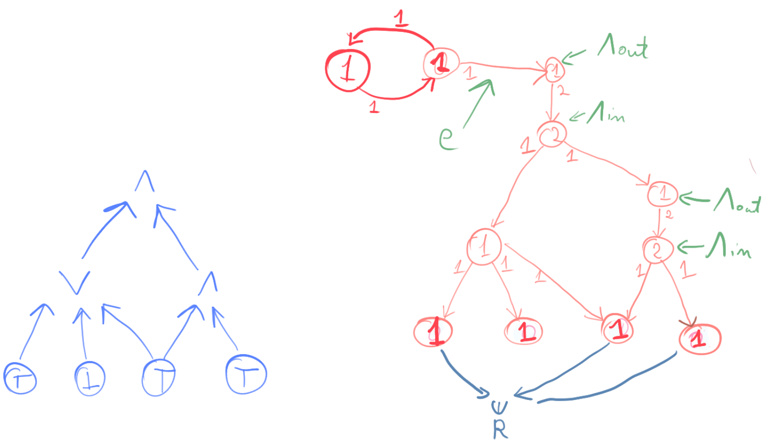
Then in every point, where two edges are intersecting, we place the crossover gadget. Let's assume we can draw the crossover gadget arbitrarily small, so that it does not introduce intersections with other edges. From Observation 2 it follows that if there was a sequence of moves winning for in original graph, then there also exists a (potentially longer) sequence of moves winning for in the graph with crossover gadgets, and vice versa.

Fig. 4: Example transformation of a graph created from a boolean circuit using the method from subproblem 1 to a planar graph. The purple dots each denote a crossover gadget.



3. Show that it remains P-complete if we restict to graphs whose vertices have degrees at most 3 and the possible weights of both vertices and edges are {1, 2}.
Here I will use the solution from the first subproblem but add one more assumption: that the provided boolean circuit for CVP has vertices with degrees at most 3, and both input degrees and output degrees are at most 2 for each node. It is clear that, with such assumption, my reduction to zero-player flow game from the first subproblem will produce nodes and edges with weights not larger than 2. The nodes with weight 0 (input nodes and the dummy top node) can be handled easily: input nodes' weights can be changed to 1 without rendering the graph illegal. For the dummy top node, it is enough to "plug" it to a small cycle with weights 1. See the illustration below:

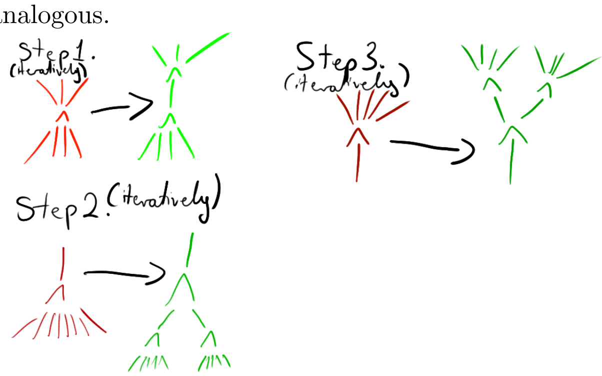
Fig. 5: A small modification of the reduction from the first subproblem to get rid of weights equal to 0.



I have made an assumption that all vertices in the provided circuit have a degree at most 3, and neither input degree nor output degree is larger than 2 for any node and I will justify that every circuit can be transformed to satisfy this assumption in reasonable (polynomial) time.

First, every node such that both input and output degree are larger than 1, is split into two, connected by a single edge. One node handles all input edges, the second – all output edges. Both nodes perform the same boolean operation as the original one. This procedure is performed while there is at least one vertex with input and output degree larger than 1.
Next, every node with input degree higher than 2 is split into 3 nodes, such that all input edges are divided evenly (±1) between two of them and the third one aggregates their results (refer to the drawing below). This procedure is performed iteratively while possible.
The last step is to perform analogous procedure for outgoing edges. It is also performed iteratively while possible.

Fig. 6: The three rules to apply iteratively (while possible) to reduce the input and output degrees of the nodes in given boolean circuit. For ∨ those rules are of course analogous.



Unbounded problem

Consider the following unbounded problem: given a legal graph $G = \langle V, E \rangle$, a set of edges $R \subseteq E$ and an edge $e \in E$, does there exist a sequence of moves that reverses e?
1. Show that the unbounded problem is PSpace-complete.
1. <u>$\in$ PSPACE</u>
There can be no more than $2^{|E|} \cdot 2^{|E|} \cdot 2^{|E|}$ different combinations of current state of the game:

- each edge can either be reversed or not
- each edge can either be in R_k or not, for current step number k
- each edge can either be in R_{k-1} or not, for current step number k

The exact value of k is irrelevant to the computation of sets E_{k+1}, R_{k+1}. Note that the above is an upper bound, perhaps a smaller number of different states can be proven. An example algorithm working in polynomial space is one that just simulates the zero-player unbonded game for at most $2^{|E| \cdot 3}$ steps, keeping a counter of steps in binary format ($3c|E|$ bits are needed). If edge e has

not been reversed during this number of steps, it never will (since some state of the game has reoccurred and the play will forever loop).

2. PSPACE-hard
I will show a reduction of QBF problem to the unbonded zero-player game. In fact, I will reuse my solution from one-player game.
The input to QBF problem is a formula in prenex normal form, where universal and existential quantifiers alternate, no $\rightarrow$ symbol occurs (implication can be rewritten as an alternative with negation) and all negations are applied directly to the variables (which can be ensured using de Morgan's laws). Such assumptions do not reduce the expressive power of possible input formulas.
The provided QBF can be decomposed into a tree where every node contains a formula created in a compositional way from its children.
Thus every subformula is of one of the following forms:

- x – a single free variable
- $\neg x$ – a negation of single free variable
- $\theta \vee \delta$, where θ and δ are subformulas with no quantifiers, no $\rightarrow$ symbols, such that every negation is applied directly to a variable
- $\theta \wedge \delta$, with the assumptions about θ and δ as above
- $\exists_x \psi$, where ψ is a formula, possibly with quantifiers and a potentially non-empty set of free variables
- $\forall_x \psi$, where ψ is a formula, possibly with quantifiers and a potentially non-empty set of free variables

I will show how to, for each of the above kinds of subformulas, construct a subgraph. The algorithm is be compositional, i.e. does not require any knowledge about nested subformulas. Another nice property is that the graph will stay valid at each step of the game.

TODO

1.2 One-player game

Let $\mathcal{G} = \langle V, E \rangle$ be a legal graph. A move in a one-player game in $\mathcal{G}$ is a reversal of a single edge, such that the resulting graph is legal.
Consider the following problem: given a legal graph $\mathcal{G}$ and an edge $e \in E$, does there exists a sequence of moves that reverse edge e?
We proved that if we restrict the above problem to the case where every edge can be reversed at most once, then the problem is NP-complete (reduction from 3CNF)
Show that the above problem is PSpace-complete.
1. $\in$ PSPACE
I will use Savitch's theorem here, more concretely its corollary NPSPACE = PSPACE.
Let $G = \langle V, E \rangle$ be the graph on which the only player plays the game and let e be the winning objective – the single edge to reverse in order to win the game. There can only be $2^{|E|}$ different states of the game, so if there exists a sequence of moves reversing edge e, there also exists such sequence not longer than $2^{|E|}$.
We can use a totally nondeterministic algorithm, which in each timestep chooses nondeterministically any edge in the graph such that it can be reversed without invalidating the graph (this can be checked in polynomial time, even linear $O(|V| + |E|)$ by checking weights inequalities for each node) and reverses it. This procedure stops after $2^{|E|}$ steps or if no edge can be reversed without invalidating the graph. If (and only if) any of the nondeterministic runs finds a sequence reversing e, then the player has the winning strategy.

2. PSPACE-hard
I will show a reduction from QBF in to one-player flow game. I will assume three things about the formula:

- It is in prenex normal form (i.e. all quantifiers preceed the portion containing an unquantified Boolean formula). Moreover, let's assume that the existental and universal quantifiers alternate – if it is not the case in the original input formula, we can introduce quantifiers with dummy variables, not used anywhere in the formula. For instance, $\exists_{x_1} \exists_{x_2} \phi(x_1, x_2) \mapsto \exists_{x_1} \forall_{y_1} \exists_{x_2} \phi(x_1, x_2)$ (y_1 is a "dummy" variable).
- There are no $\rightarrow$ symbols in the "body" of the formula. Every implication can be transformed into an alternative with negation ($a \rightarrow b$ is $\neg a \vee b$).
- All negations in the formula are applied directly to the variables (this can be easily ensured with de Morgan's laws).

Let $\forall_{x_1} \exists_{x_2} \forall_{x_3} ... \exists_{x_n} (y_{1,1} \vee ... \vee y_{1,k_1}) \wedge (y_{2,1} \vee ... \vee y_{2,k_2}) \wedge ... (y_{m,1} \vee ... \vee y_{m,k_m})$ be the given quantified boolean formula.
I will show a method for constructing the game graph by treating the formula as a composition of smaller formulas. The graphs created from any formula will always conform to two invariant assumptions:

- All the free variables are provided to the formula as nodes with weight 1. There will also be nodes for negations of the free variables. A variable can be set to true by firing its corresponding node. It can be set to false by firing its negation's vertex.
- A graph for formula δ contains a node with an outgoing edge of weight 1 which can be reversed (following the games rules) if and only if the formula δ is satisfiable. Moreover, this edge will have weight 1 if $FV(\delta) \neq \emptyset$ and 0 if $FV(\delta) = \emptyset$ ($FV(\delta)$ is the set of free variables in δ).

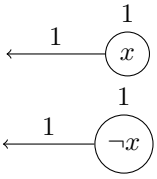
The provided QBF can be decomposed into a tree where every node contains a formula created in a compositional way from its children.
Thus every subformula is of one of the following forms:

- x – a single free variable
- $\neg x$ – a negation of single free variable
- $\theta \vee \delta$, where θ and δ are subformulas with no quantifiers, no $\rightarrow$ symbols, such that every negation is applied directly to a variable
- $\theta \wedge \delta$, with the assumptions about θ and δ as above
- $\exists_x \psi$, where ψ is a formula, possibly with quantifiers and a potentially non-empty set of free variables
- $\forall_x \psi$, where ψ is a formula, possibly with quantifiers and a potentially non-empty set of free variables

I will show how to, for each of the above kinds of subformulas, construct a subgraph.

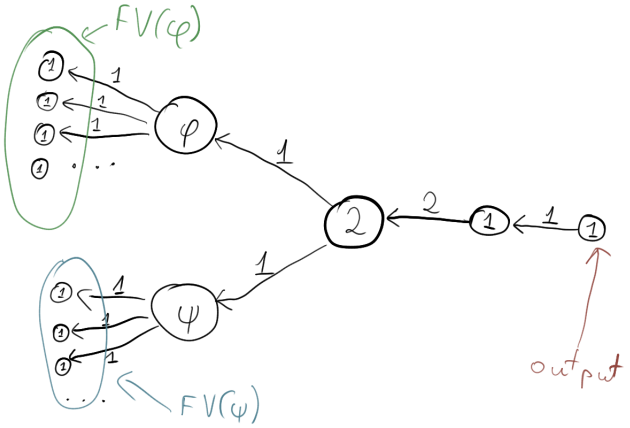
$x, \neg x$
From our assumption that the free variables are provided as nodes with weight 1, also the second invariant follows – the provided node has weight 1 and is firing if and only if the variable (or its negation) is set to true.

Fig. 7: Node marked x ($\neg x$) is firing if and only if the provided value for the free variable x is true (false).



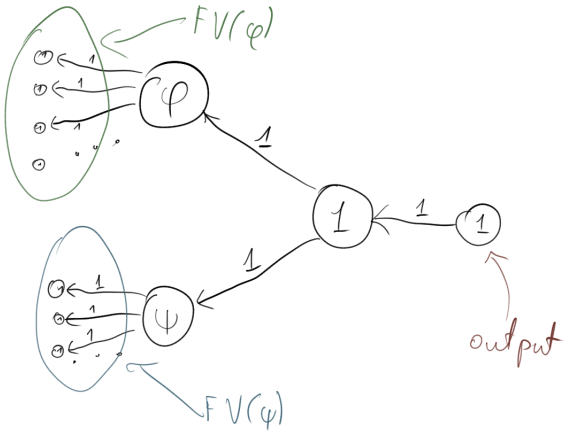
$\phi \wedge \psi$

Fig. 8: The node marked *output* is firing if and only if $\phi \wedge \psi$ is satisfied with regard to the provided free variables. The depicted state of edges corresponds to the initial state of the game. Circles with ϕ and ψ denote the graphs build for formulas ϕ and ψ. The edges drawn as incoming to ϕ or are routed to the nodes corresponding to free variables and their negations, the edges coming out from ϕ are edges from its output node. I will use this convention with all such nested graphs. Note that $FV(\phi)$ and $FV(\psi)$ do not have to be disjoint. There can also be multiple edges incoming into nodes providing the free variables values.



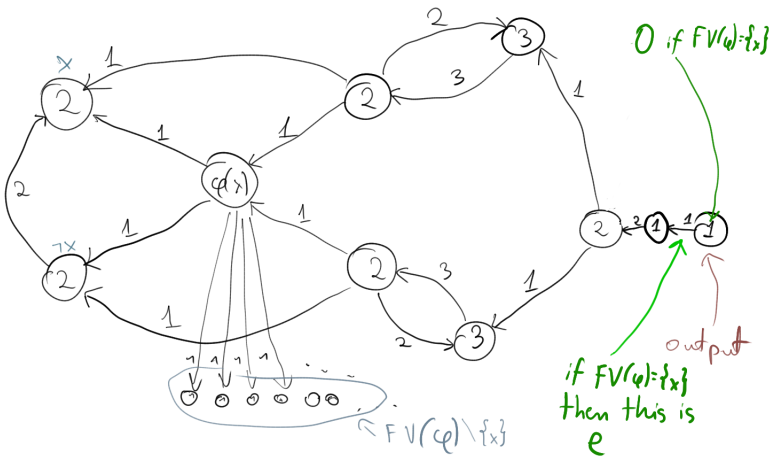
$\phi \vee \psi$

Fig. 9: The node marked *output* is firing if and only if $\phi \vee \psi$ is satisfied with regard to the provided free variables.



$\underline{\forall_x \phi}$

Fig. 10: This one is more complex. In order to satisfy the entire formula (fire the *output* node), the player must separately choose x and $\neg x$ to be true, satisfy ϕ with this choice (i.e. fire the output node of ϕ nested graph) and fire the corresponding node with weight 3 with a single edge of weight 3. Only then can the player propagate the firing towards the *output* node. What's important in this construction is that there is only one copy of graph created for ϕ, so the entire graph will not grow exponentially with the number of quantifiers. Note that if $FV(\phi) = \{x\}$ then the *output* node should have weight 0 and its only adjacent edge is the edge e that is the objective of the game.



$\underline{\exists_x \phi}$

Fig. 11: The construction is similar as in the previous case ($\forall_x \phi$), but to fire the *output* node it suffices that the player satisfies ϕ with only one choice from $\{x, \neg x\}$.

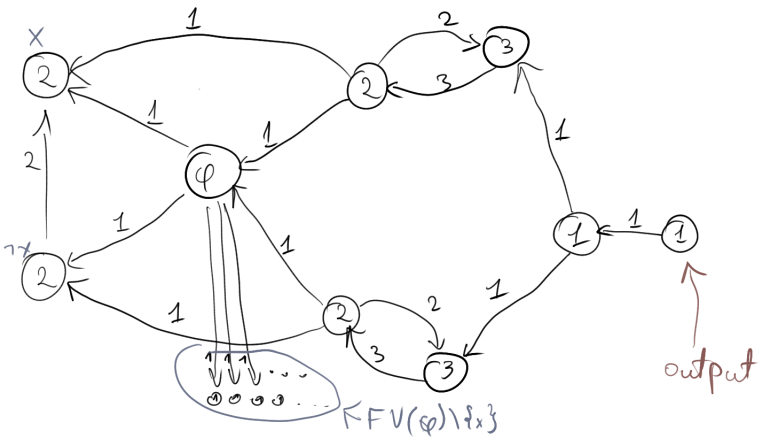
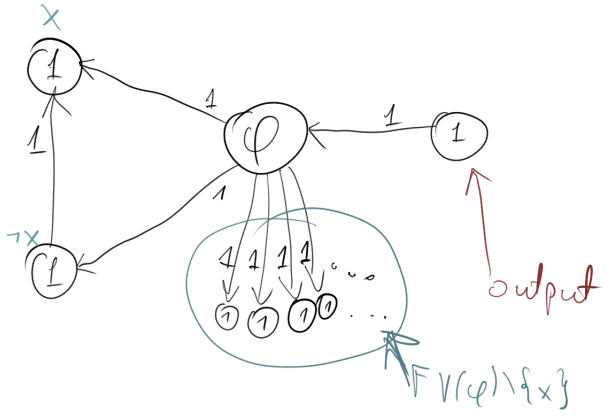


Fig. 12: This is an alternative and simpler way to construct the graph for $\exists_x \phi$ formula.



Above I have presented a compositional and recursive way for constructing a graph G and selecting and edge e such that the only player has a winning strategy in one-player flow game $\langle G, e \rangle$ if and only if the quantified boolean formula is true.

5 Multi-reachability games

In a reachability games there is a set of vertices which the first player wants to reach. In multi-reachability games (MRG) there is a family of sets of vertices and the first player wins if every set in the family has been visited at least once.

1. Show that MRG game is PSpace-complete.

1. $\in$ PSPACE

I will use the fact that a normal reachability game is solvable in polynomial time. I will also use Savitch theorem, in particular its corollary, stating that PSPACE = NPSPACE.

Let $G = \langle V, E, v_I, S \rangle$ be the MRG, where V is the set of vertices, E is the set of edges, v_I is the starting vertex and S is the family of sets the first player wants to visit.

The first player can nondeterministically choose an ordered sequence of vertices $v_1, v_2, ..., v_k$ such that $\forall_{T \in S} \exists_{1 \leqslant i \leqslant k} (v_i \in T)$. Then we want to check in polynomial space whether the first player can visit selected vertices in the specified order, regardless of what the second player is doing. This can be done by considering around $4k$ normal reachability games. For each $i, 1 \leqslant i \leqslant k$, we create 4 reachability games:

- $\langle V', E', (v_{i-1}, 0), \{(v_i, 0)\} \rangle$
- $\langle V', E', (v_{i-1}, 0), \{(v_i, 1)\} \rangle$
- $\langle V', E', (v_{i-1}, 1), \{(v_i, 0)\} \rangle$
- $\langle V', E', (v_{i-1}, 1), \{(v_i, 1)\} \rangle$

where:
$V' = V \times \{0, 1\}$
$E' = \{((u, 0), (v, 1)), ((u, 1), (v, 0)) \mid (u, v) \in E\}$
$v_0 = v_I$
Player p starts, where $(v_{i-1}, p) = v_I$ (not necessarily the first player from our MRG)

Such extended set of vertices V' carries, together with the vertex, the information whose move it is. Having solved the single reachability games described above, we can perform an algorithm in a dynamic-programming-manner as follows:

```
wins: bool[k][2][2] = given      # wins[i][ps][pe] iff the first
                                 # player has a winning strategy in
                                 # a game <V', E', (v_{i-1}, ps), (v_i, pe)>

can_reach[][] = false * [k][2]   # At the end of the algorithm
                                 # can_reach[i][p] = true iff
                                 # first player can force the game
                                 # to reach vertex v_i and player
                                 # number p has the next move.

can_reach[0][0] = true           # First player starts.

for i := 1 to k:
  for ps := 0 to 1:
    for pe := 0 to 1:
      if can_reach[i-1][ps] and wins[i][ps][pe]:
        can_reach[i][pe] = true
```

If either `can_reach[k][0]` or `can_reach[k][1]` then the first player has a winning strategy.

2. PSPACE-hard
I will show a reduction of QBF problem to an MRG game.
An input to the QBF problem is a formula, about which I will make two assumptions:

- It is in prenex normal form (i.e. all quantifiers preceed the portion containing an unquantified Boolean formula). Moreover, let's assume that the existental and universal quantifiers alternate – if it is not the case in the original input formula, we can introduce quantifiers with dummy variables, not used anywhere in the formula. For instance, $\exists_{x_1}\exists_{x_2}\phi(x_1, x_2) \mapsto \exists_{x_1}\forall_{y_1}\exists_{x_2}\phi(x_1, x_2)$ (y_1 is a "dummy" variable).
- The "body" of the formula is in conjunctive normal form.

Note that the above assumptions do not reduce the expressive power of input formulas. Every possible formula can be represented in the described format. QBF problem for such normalized formulas is still PSPACE-complete.
Let $\forall_{x_1}\exists_{x_2}\forall_{x_3}...\exists_{x_n}(y_{1,1}\vee...\vee y_{1,k_1})\wedge(y_{2,1}\vee...\vee y_{2,k_2})\wedge...(y_{m,1}\vee...\vee y_{m,k_m})$ be the input QBF formula, where $y_{...} \in \{x_1, ..., x_n, \neg x_1, ...\neg x_n\}$.
The created multi-reachability game is $G = \langle V, E, v_I, S\rangle$, where:

- V is the set of vertices. Vertices are indexed by all variables bound by quantifiers and their negations, plus there is the initial vertex. $V = \bigcup\limits_{1\leqslant i\leqslant n} \{v_{x_i}, v_{\neg x_i}\} \cup \{v_I\}$. Note that the vertices are positions and:
  $Pos = V$
  $v_x \in Pos_\exists$ iff x_i is bound by an existential quantifier
  $v_x \in Pos_\forall$ iff x_i is bound by an universal quantifier
- E is the set of edges. $E = \{(v_I, v_{x_1}), (v_I, v_{\neg x_1})\} \cup \bigcup\limits_{2\leqslant i\leqslant n} \{(v_{x_{i-1}}, v_{x_i}), (v_{x_{i-1}}, v_{\neg x_i}), (v_{\neg x_{i-1}}, v_{x_i}), (v_{\neg x_{i-1}}, v_{\neg x_i})\}$
- v_I is a starting vertex.
- S is the family of sets of vertices that the first player wants to reach. It is created directly from the CNF formula, i.e. $S = \bigcup\limits_{1\leqslant i\leqslant m} \{\{v_{y_{i,j}} \mid 1 \leqslant j \leqslant k_i\}\}$

The first player is the existential player if the formula starts with an existential quantifier. Otherwise the universal player starts. The QBF formula is satisfiable iff the first player has a winning strategy.

2. Show that one-player MRG is NP-complete.
1. $\in$ NP

The algorithm is as follows:

1 We make a nondeterministic selection of one element of every set of vertices to be visited and we also choose nondeterministically a permutation of those verices – the order in which we want to visit them. If some vertex was chosen more than once, we only leave one copy of it in the chosen order.

2 It is possible to check in polynomial time whether the chosen vertices can be visited in the chosen order. Let $v_1, v_2, ..., v_k$ be the chosen sequence of vertices. For any pair of vertices u, w, it can be decided in polynomial time whether w is reachable from u (e.g. a simple depth-first search algorithm). Let $v_0 = v_I$, the starting vertex of the game. The player has a winning strategy if $\forall_{1 \leqslant i \leqslant k}(v_i \text{ is reachable from } v_{i-1})$.

2. NP-hard
I will show the reduction of 3SAT problem (satisfiability of CNF formula where each clause has at most 3 literals) to one-player MRG. The resulting game will be very similar as in previous subproblem.
Let $(y_{1,1} \vee y_{1,2} \vee y_{1,3}) \wedge ... \wedge (y_{m,1} \vee y_{m,2} \vee y_{m,3})$ be the input formula, where $\underset{\substack{1 \leqslant i \leqslant m \\ 1 \leqslant j \leqslant 3}}{\forall} y_{i,j} \in \bigcup_{1 \leqslant k \leqslant n} \{x_k, \neg x_k\}$.

We create a one-player MRG as follows:
$G = \langle V, E, v_I, S \rangle$, where:

- $V = \bigcup_{1 \leqslant i \leqslant n} \{v_x, v_{\neg x}\} \cup \{v_I\}$ is the set of vertices. Each vertex (except for the initial one) corresponds to some variable or its negation. Visiting a vertex is synonymous to assigning a value to the variable.
- E is the set of edges. $E = \{(v_I, v_{x_1}), (v_I, v_{\neg x_1})\} \cup \bigcup_{2 \leqslant i \leqslant n} \{(v_{x_{i-1}}, v_{x_i}), (v_{x_{i-1}}, v_{\neg x_i}), (v_{\neg x_{i-1}}, v_{x_i}), (v_{\neg x_{i-1}}, v_{\neg x_i})\}$.
- v_I is the initial vertex.
- S is the family of sets of vertices that the player wants to reach. It is constructed directly from the CNF formula, i.e. $S = \bigcup_{1 \leqslant i \leqslant m} \{\{v_{y_{i,j}} \mid 1 \leqslant j \leqslant 3\}\}$

The formula is satisfiable iff the player can find a path in the graph, starting from v_I and visiting at least one vertex from each set in S. Moreover, the indices of visited vertices form a satisfying valuation of the variables occuring in the input formula.

3. Show that if the sets are singletons then MRG is P-complete.

1. $\in$ P
I will again use the fact that a single reachability game is decidable in polynomial time. Let $G = \langle V, E, v_I, S = \{\{v_1\}, \{v_2\}, ..., \{v_k\}\}\rangle$. Just like in the first subproblem, let's create some single reachability games:
For each $i, 1 \leqslant i \leqslant k$, we create 4 reachability games:

- $\langle V', E', (v_{i-1}, 0), \{(v_i, 0)\}\rangle$
- $\langle V', E', (v_{i-1}, 0), \{(v_i, 1)\}\rangle$
- $\langle V', E', (v_{i-1}, 1), \{(v_i, 0)\}\rangle$
- $\langle V', E', (v_{i-1}, 1), \{(v_i, 1)\}\rangle$

where:
$V' = V \times \{0, 1\}$
$E' = \{((u, 0), (v, 1)), ((u, 1), (v, 0)) \mid (u, v) \in E\}$
$v_0 = v_I$
Player p starts, where $(v_{i-1}, p) = v_I$ (not necessarily the first player from our MRG)

If (and only if) the first player wins a game $\langle V', E', (u, p), \{(v, q)\}\rangle$, it means that, starting from node u in the original game, if it's turn of player p, the first player has a strategy for the game to always reach the node v in a way that player q moves next.
Having solved the above games, we can create a relation R such that $((u, p), (v, q)) \in R$ iff first player has a winning strategy in game $\langle V', E', (u, p), \{(v, q)\}\rangle$. Note that this relation is transitive. We turn this relation into a graph, such that the nodes are pairs (v_i, p) $(0 \leqslant k, p \in \{0, 1\})$ and the directed edge between two nodes iff they are in relation R. Then we turn this graph into a DAG of strongly connected components. Since the relation is transitive, every strongly connected component will also be a clique, so finding a Hamiltonian cycle is trivial inside each component. There exists a Hamiltonian path in the graph if and only if the DAG of strongly connected components is a single path. The first player has the winning strategy in the original MRG if and only if there exists a

Hamiltonian path in the constructed DAG. Every step of this method is polynomial in time.

2. P-hard
I will show a reduction of circuit value problem (CVP) to an MRG game where the sets are singletons. The input to the problem is a boolean circuit and an input to the circuit. The first layer consists of inputs and their negations (every boolean circuit can be transformed to a form where the only negations are the once in the input layer). Every node in further layers is either a conjunction or disjunction of its inputs.

The general idea is that we will treat the circuit as a graph and reverse the edges. In $\vee$ nodes the first player moves, in $\wedge$ nodes, the second player moves. The first player wins iff they have a strategy for reaching any input set to `true`, which can be expressed by creating a new vertex w such that all inputs set to true have an outgoing edge to w. The first player therefore wants to reach w.

In order to transform this problem to an MRG game with singleton sets, we first treat the given circuit as a graph $G_{cir} = \langle V_{cir}, E_{cir}, r\rangle$, where V_{cir} is the set of vertices, E_{cir} the set of edges and $r : V_{cir} \to \{\wedge, \vee\} \cup \{in, \neg in\} \times \mathbb{N}$ is a function describing a vertex's role. $\wedge$ and $\vee$ of course mean that the vertex is a conjunction/disjunction of its inputs, (in, k) means that it is the k-th input value, $(\neg in, k)$ means that it is the negation of k-th input value. Let s be the input sequence and $n = |s|$.

The game we construct is $\mathcal{G} = \langle Pos = Pos_\exists \cup Pos_\forall, Moves, win\rangle$, where:

- win is the set of singletons the first player wants to reach. $win = \{\{w\}\}$, where w is a new vertice such that $\forall_{p \in Pos}((r(p) = (in, i) \wedge s[i] = true) \vee (r(p) = (\neg in, i) \wedge s[i] = false) \to (p, w) \in Moves)$.
  In other words, w can be reached from any input vertex from the original circuit, which evaluates to `true`.
- $Pos = V_{cir} \cup \{w\}$ is the set of positions.
  Vertices with $\wedge$ symbol belong to the opponent, all others belong to the first player:
  $Pos_\forall = \{p \mid r(p) = \wedge\}, Pos_\exists = \{p \mid p \notin Pos_\forall\}$
- $Moves$ is the set of edges, i.e. moves on the graph.
  $Moves = \{(v, u) \mid (u, v) \in E_{cir}\} \cup \{(p, w) \mid (r(p) = (in, i) \wedge s[i] = true) \vee (r(p) = (\neg in, i) \wedge s[i] = false)\}$
- The starting position of the game is the top vertex of the circuit (the one to be evaluated in circuit value problem).

The first player can reach w in this game if and only if the original circuits evaluates to 1. For each $\vee$ node, the first player can choose which compound of the alternative they want to be true. For each $\wedge$ node, we let the second player choose any compound of the conjunction. Once the game reaches a node corresponding to an input in the original circuit (which of course happens in finite time since this is a DAG), the first player wins if and only if this input evaluates to `true`.