Chapter 2

Finite Two-Player Games with Perfect Information

2.1 Determinacy

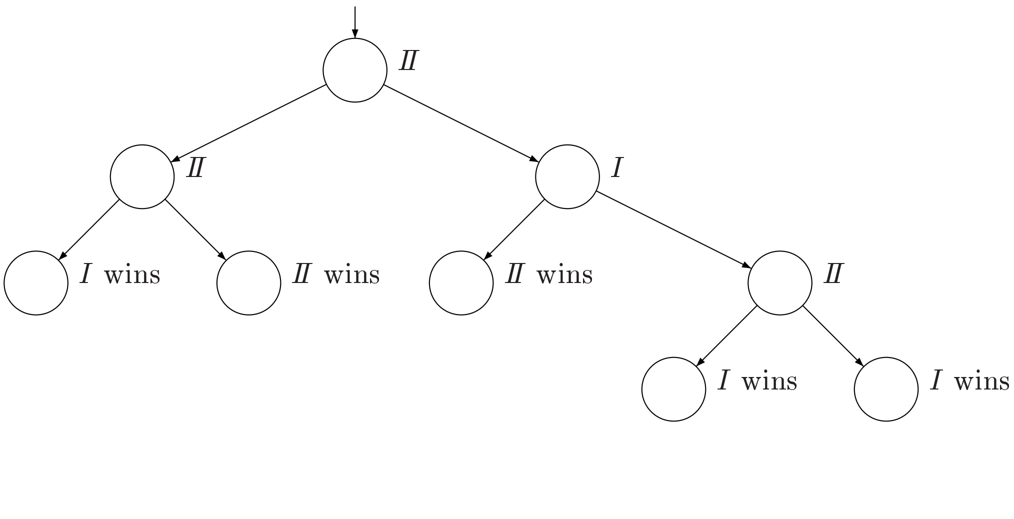
We concentrate on games with

- two players, since we want to avoid the many problems generated by possible coalitions of players in multiplayer games (when there are at least three players).;
- perfect information, for the reason that these games are easier to analyze. And also, and in many situations, games with imperfect information can be translated in terms of more complicated rules for games with perfect information.
- Zero sum, which means, what a player wins is what the other loses. Hence the outcomes are just win or lose.

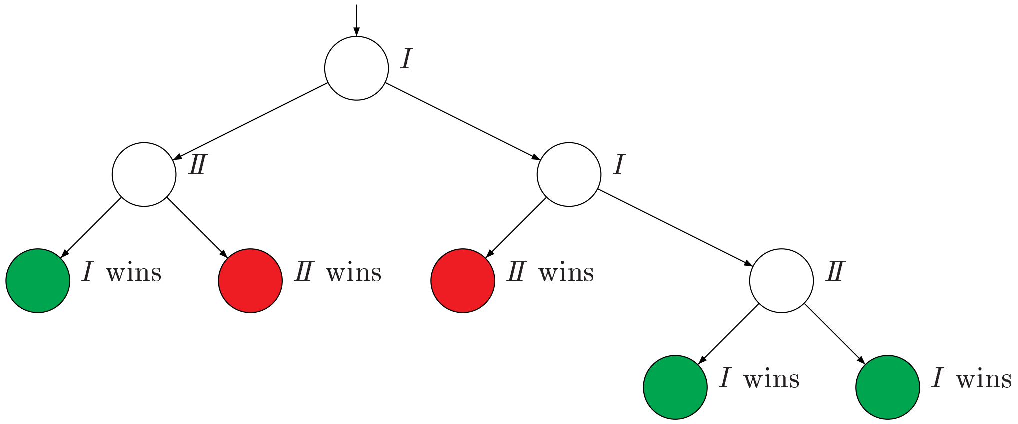
Let $\mathcal{T}$ be the tree that stands for the *extensive form* of two-player game with *perfect information*. By definition, this game is *finite* if its tree is *wellfounded* which means $\mathcal{T}$ does not have an infinite branch. In other words, any path through a branch of $\mathcal{T}$ eventually reaches an end node (called a *leaf*).

Theorem 7 *[Zermelo]*
All finite two-player games with perfect information are determined.

Proof of theorem 7: Given a tree $\mathcal{T}$ that stands for the extensive form of that two-player game, we describe an algorithm for determining the player having the winning strategy at any node.



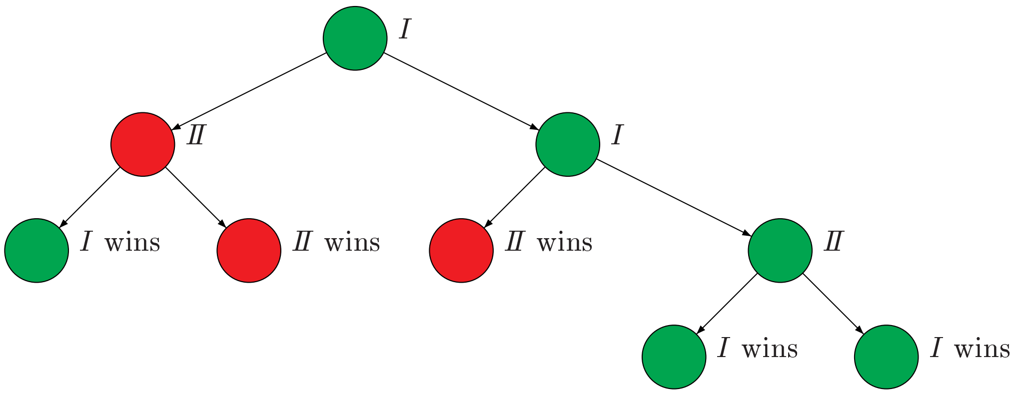
(1) colour all leaves (end nodes) *red* that are wins for player *II*. Colour all the other leaves *green*.



(2) Repeat the following procedure: if all children of node c have been coloured already, do one of the following:

(a) if player *II* is to move at that node, and at least one child is red: colour c *red*, if all children are *green*, colour c *green*.

(b) If player I is to move, and at least one child is *green* : colour c *green*, if all children are *red*, colour c *red*.



It is easy to see that this procedure eventually gives the colour *green* to all nodes where I has a winning strategy. Indeed, player I has a w.s. in a node iff one the three following situations occur:

- it is a leaf which is winning for I,
- it is I's turn to play and there is a move available to a node where I has a winning strategy,
- it is *II*'s turn, but all *II*'s moves lead to nodes where I has a winning strategy.

$\dashv$ 7

2.1.1 Computational Complexity

Once we know that necessarily one of the two players of a finite game with perfect information has a winning strategy, the first question that comes to mind is: how complicated is it to find out which player has a winning strategy ?
If we carefully look at the above proof, we see that in this tree-colouring algorithm, every node is visited

(1) once for the initial coloring and

(2) once for to find the color for the parent node.

Thus, this colouring procedure visits each node precisely twice. This shows that the problem of determining which player has a w.s. in a finite game can be solved in *linear time* of the size of its tree.

However, many problems stated in terms of game that we encounter do not come with their extensive form. Usually, the whole tree $\mathcal{T}$ is not given. Think about Chess for instance. What is given is a certain configuration, who is to play next (white or black) and the rules of the game which say which succession of configuration is legal, and finally the condition for the ending of the game and which player wins (if any). If we think of the extensive form of Chess, we see that this tree contains many different copies of the same configuration, and that holds for lots of them (simply because there is many ways to reach a certain configuration from the initial one). Assume we agree on the convention that a player who reaches a configuration that had already been visited in the run loses the game. With this convention board games like Chess are all finite games (Chess really *is* finite), hence they are determined which means one of the two player has a *non losing* strategy: that is a strategy so to force a win or a draw (the ending of a game in which neither of two players wins, sometimes also called a "tie).
The problem of finding who has a w.s. in such game is *linear time* in the size of the game tree, but what about in the size of the initial configuration ?

(1) Generalizations of Chess to n*n (instead of 8*8) boards are EXPTIME-complete.

(2) Checkers and Draughts. In this game players move pieces diagonally forward one square at a time on alternating squares of an n*n board, removing the other players' pieces by jumping diagonally over them. Object is to leave opponent with no move by blocking or jumping all pieces. The 8*8 version is called checkers, but on larger boards it is called draughts. Draughts are PSPACE-complete.

2.2 Evalutation Games for First Order Predicate Logic

Given a model $\mathcal{M}$ of First Order Predicate Logic and a formula ϕ with no free variable, from the same logic and with all predicate having an interpratation in $\mathcal{M}$. The evaluation game $\mathbb{E}v\ (\mathcal{M}\ ,\ \phi)$ is an example of a finite, hence determined game, due to Jaakko Hintikka:

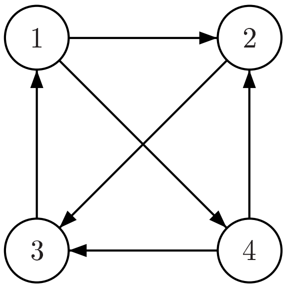
(1) Two players called *Verifier* and *Falsifier*. *Verifier* (**V**) claims that $\mathcal{M} \models \phi$ holds, and *Falsifier* (**F**) on the contrary claims it is false. Their moves consist in chosing formulae with the following rules:

(2)

if ϕ is of the form:	Who's turn	play goes on with
Atomic formula	no one	Stop
$\varphi \vee \psi$	**V** chooses φ ψ	**V**'s choice
$\varphi \wedge \psi$	**F** chooses φ ψ	**F**'s choice
$\exists x \varphi(x)$	**V** chooses $a \in M$	$\varphi(a)$
$\forall x \varphi(x)$	**F** chooses $a \in M$	$\varphi(a)$
$\neg\varphi$	role switch between **F** and **V**	φ

(3) The winning conditions occur when the play stops, i.e. when the underlying formula $\phi' = R(a_1, \ldots, a_k)$ is atomic. **V** wins if $\mathcal{M} \models \phi'$,and **F** wins otherwise.

Example 8 Let $\mathcal{M} = (\{1, 2, 3, 4\}, E)$ where E is a binary relation defined by



A possible run in the game $\mathbb{E}v$ $(\mathcal{M}\ ,\ \forall x \forall y\, (E(x, y)\ \wedge\ \exists z(E(x, z)\ \wedge\ E(z, y))))$:

Player	move	play goes on with
F	picks 2	$\forall y\, (E(2, y)\ \wedge\ \exists z(E(2, z)\ \wedge\ E(z, y)))$
F	picks 4	$E(2, 4)\ \wedge\ \exists z(E(2, z)\ \wedge\ E(z, 4))$
V	chooses	$\exists z(E(2, z)\ \wedge\ E(z, 4))$
V	picks 1	$E(2, 1)\ \wedge\ E(1, 4)$
F	chooses	$E(1, 4)$
	STOP	**V** wins

These games are clearly determined, independently of the size of the model, since every run has a lenght bounded by the size of the initial formula. Therefore, they are determined. In our example, *Falsifier* has a winning strategy. Moreover, the following holds:

Theorem 9

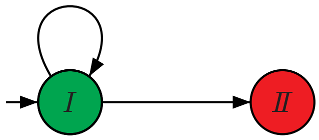
$$\mathcal{M} \models \phi \iff \mathbf{V} \textit{ has a w.s. in } \mathbb{E}v\ (\mathcal{M}\ ,\ \phi)\,.$$

Proof of theorem 9: The proof goes on by induction on formulas, showing simultaneously that if $\mathcal{M} \models \phi$ then **V** has a w.s. in $\mathbb{E}v$ $(\mathcal{M}, \phi)$, and if $\mathcal{M} \models \neg\phi$ then **F** has a w.s. in $\mathbb{E}v$ $(\mathcal{M}, \phi)$.

$\dashv$ 9

2.2.1 memory and strategies

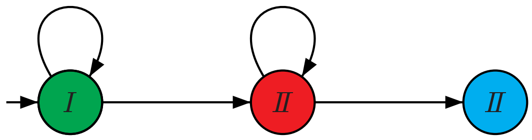
Consider the following game with only two configurations: Red and Green. Player I starts and has two choices: either Green or Red, if I chooses Green it is I's turn again, on the contrary, chosing Red, leads to an end.



The winning conditions are I wins iff the configuration *Green* is encountered more than once during the run **and** *Red* is finally reached. Clearly, I has a w.s., but I has no memoryless winning strategy since such a strategy would be a mapping $f : \{Red, Green\} \longmapsto \{Red, Green\}$, but no such is winning for I: if it sends *Green* to *Green* then this strategy never reaches *Red*, if it sends *Green* to *Red*, it never encounters *Green* more than once.

In other words, a w.s. in this easy game must keep some memory of what happened in the past of the run.

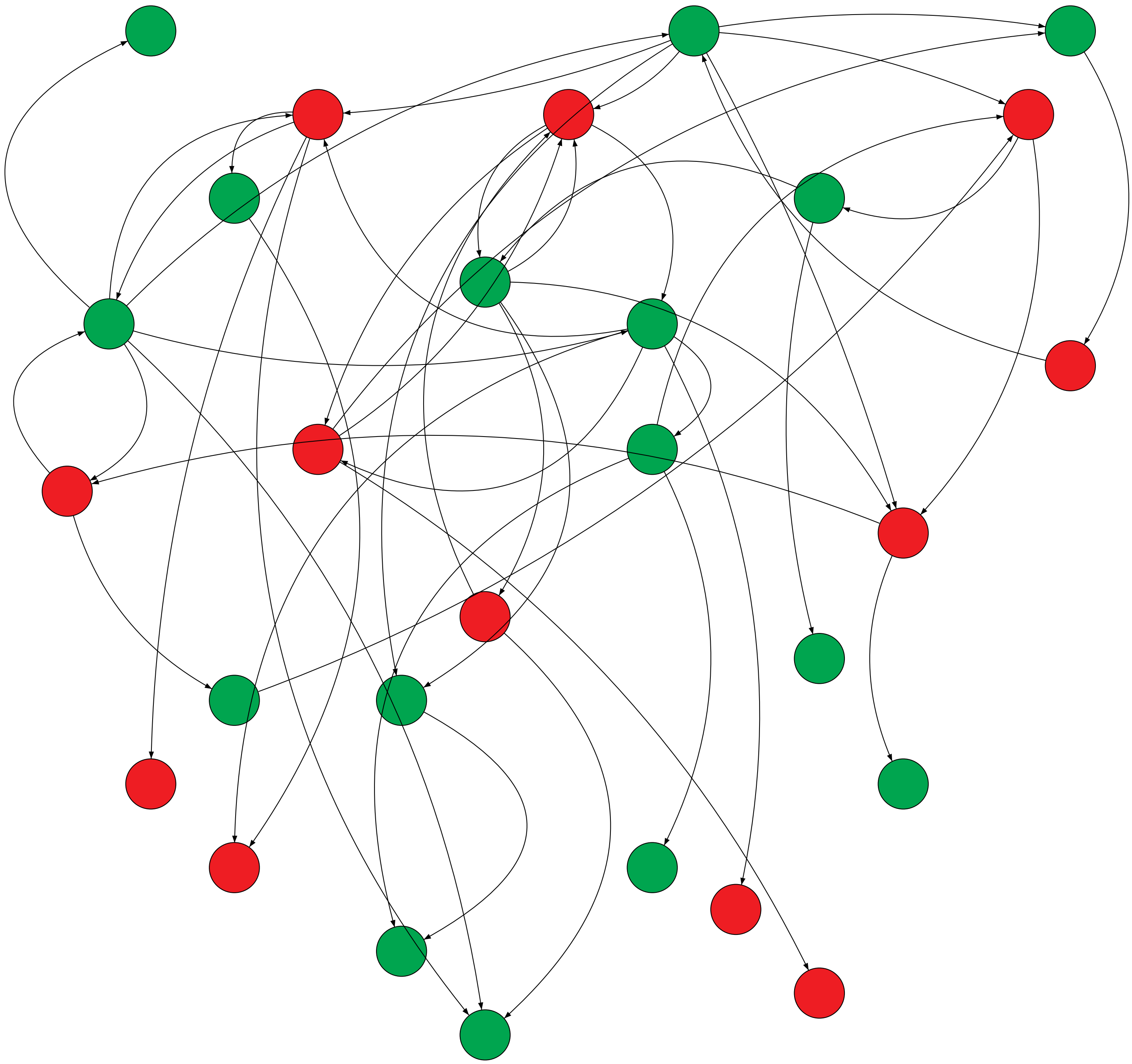
How much memory is needed to win a game is one major issue in game theory. Consider the following one with three configurations: *Red*, *Green* and *Blue*. Player I starts and has two choices: either Green or Red, if I chooses Green it is I's turn again, on the contrary, chosing Red, leads to an end.



The winning conditions are I wins iff *Blue* is never reached **or** configuration *Red* is encountered less often than *Green*. Clearly, II has a w.s. in this game, but he certainly does not have a memoryless winning strategy, neither has he a w.s. with bounded memory. Any winning strategy "needs to know" how many times I went through *Green*. Any winning strategy highly depends on the history of the play.

2.3 Games over Finite Graphs

We consider $\mathbb{G}(\mathcal{G}\ ,\ u)$, games played over finite directed graphs of the form $\mathcal{G} = (V, V_I, E)$ where $V_I \subseteq V$ is the set of vertices where player I moves, and $V_{II} = V \setminus V_I$, the coresponding one for player II. Given $u \in V$, we write uE to denote the set of successor of u in $\mathcal{G}$. A run starts at configuration u. It is a (possibly infinite) path in the graph. i.e. a sequence $u_0, u_1, \ldots$ such that $u_0 = u$ and for all integer k $(u_k, u_{k+1}) \in E$. At configuration u_k, I chooses u_{k+1} if $u_k \in V_I$, otherwise II does it. Next figure gives an example of such an arena. We colored the set V_I in green and the set V_{II} in red.



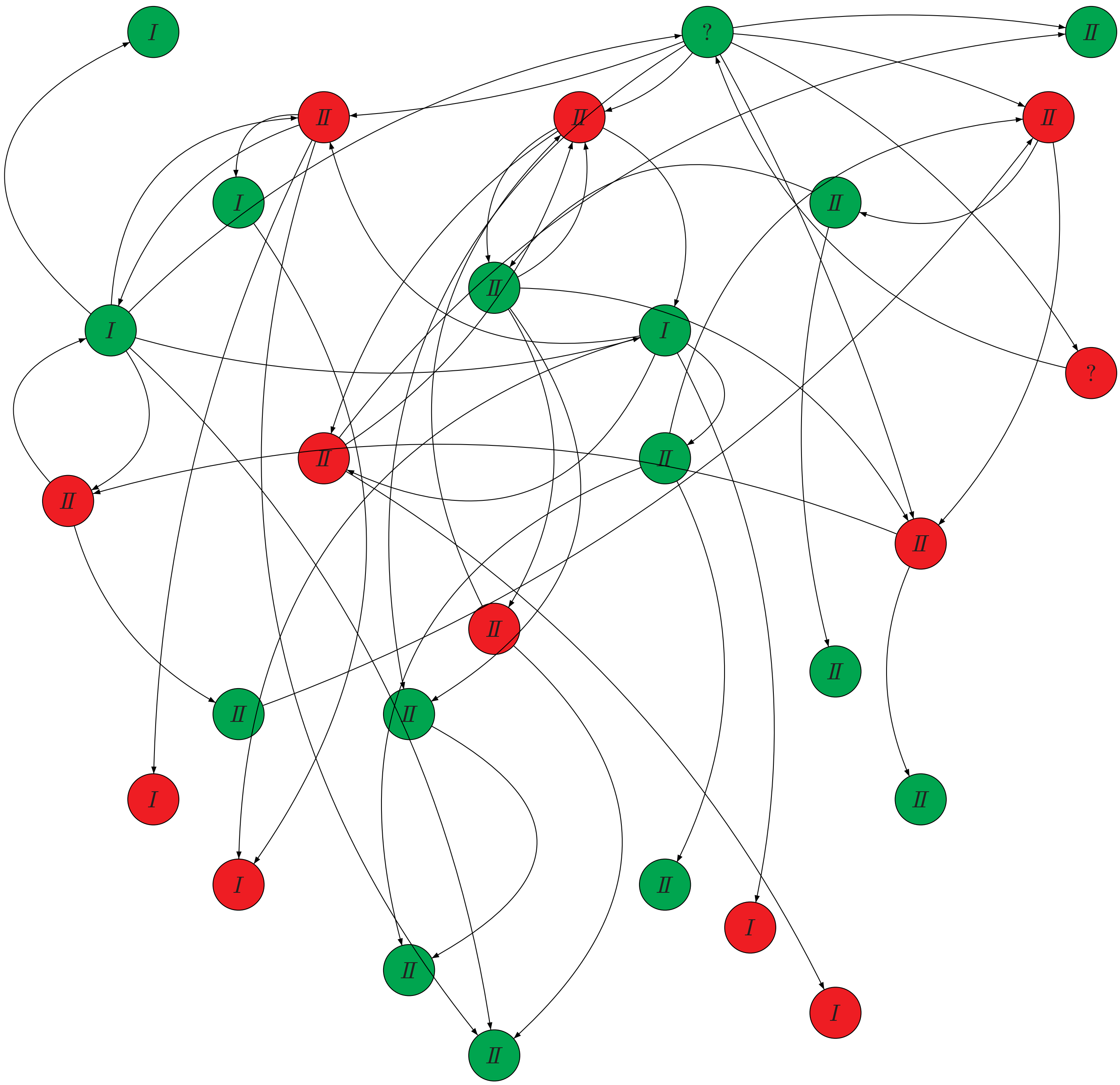
The winning conditions are: a player wins if the run ends in a configuration u_n where the other player should move but cannot since $u_n E = \emptyset$. If the run is infinite, for the moment, we use the convention that none of the players wins nor loses.
We consider the problem of determining who (and where) has a winning strategy in a finite game graphs:

$$GAME = \{(\mathcal{G}, u) : \textit{ player I has a w.s. in } \mathbb{G}(\mathcal{G}\ ,\ u)\}.$$

Obviously, this GAME can be solved in polynomial time: let W_i^n be the set of all vertices v such that player i has a strategy to win the game $\mathbb{G}(\mathcal{G}\ ,\ v)$ in at most n moves. W_i^n is inductively defined buy

- $W_i^0 = \{v \in V_{\bar{\imath}} :\ vE = \emptyset\}$; where $\bar{\imath}$ stands for I if $i = I\!I$ and $\bar{\imath}$ stands fo $I\!I$ if $i = I$.
- $W_i^{n+1} = \{v \in V_i :\ vE \cap W_i^n \neq \emptyset\} \cup \{v \in V_{\bar{\imath}} :\ vE \subseteq W_i^n\} \cup W_i^n$.

This sequence $(W_i^n)_{n\in\mathbb{N}}$ is increasing for $\subseteq$, the first fixpoint $W_i^n = W_i^{n+1}$ gives the desired set of all nodes from where player i has a winning stategy. See next figure where the winning set of each player for the previous directed graph is indicated. Notice that two nodes do not allow a player to have a winning strategy.



But we have a much better result:

Theorem 10 *Winning sets of finite games over finite graphs can be computed in linear time.*

Proof of theorem 10: The algorithm below computes for each vertice u, which player, if any, has a w.s. starting at node u. This computation uses three arrays:

(1) $win[v]$: contains either I, II or ? depending on which player (I or II) we know for sure wins the game starting at v, and ? if we still don't know.

(2) $pred[v]$ is the list of the predecessors of v.

(3) $nb[v]$ is the number of the successors of v for which we still don't know ($win[v] :=?$).

The procedure `game`(V_I, V_{II}, E) which realizes the desired computation relies on the procedure `propagate`(v, i) which is called each time we find out that player i has a w.s. from node v. `propagate`(v, i) records this fact and checks if we are now able to determine which player has a w.s. at the predecessors of v. This may happen in one of the following cases

(1) The predecessor u belongs to player i, then this player can force a win by moving from u to v.

(2) (a) The predecessor u belongs to $\bar{\imath}$,
  (b) $win[u] =?$ (so that we still don't know which player has a w.s. from u),
  (c) $nb[u] = 0$ (so that the winning player has already been determined for all successors of u).

  Then in that case, $win[w] = i$ for all the successors of u (if a successor w' would be a winning position for $\bar{\imath}$, then the algorithm would have already set $win[u]$ to $\bar{\imath}$.

(1) **procedure** `game`(V_I, V_{II}, E)

```
    forall v ∈ V_I ∪ V_II do                    //* (1) initialization *//
        win[v] := ⊥
        pred[v] := ∅
        nb[v] := 0
    end

    forall (u, v) ∈ E do                        //* (2) calculate pred and nb *//
        pred[v] := pred[v] ∪ {u}
        nb[u] := nb[u] + 1
    end

    forall v ∈ V_I do                           //* (3) calculate win *//
        if nb[v] = 0 then
            propagate(v, II)
    forall v ∈ V_II do
        if nb[v] = 0 then
            propagate(v, I)
    return win
end
```

(2) **procedure** `propagate`(v, i)

```
if win[v] = ⊥ then                                   //* (4) mark v as winning for i *//
    win[v] := i
    forall u ∈ pred[v] do            //* (5) propagate the change to all predecessors *//
        nb[u] := nb[u] − 1
        if u ∈ V_i or nb[u] = 0 then
            propagate(u, i)
    end
end
```

The correctness of the value assigned to $win[u]$ is proved by induction on the number n of moves in which the coresponding player can ensure that he wins.

- the case $n = 0$ is obvious, since u is a leaf, hence $win[u] =?$ and $nb[u] = 0$. The procedure `propagate`(v, i) is called for the player who is not in charge of u, which result in $win[u] = i$.
- If player i can ensure a win from u in at most $n + 1$ moves (but not less than that), then
  - either u is a node for i, in which case there exists v a successor of u such that i has a w.s. from u; and (5) makes sure $win[u]$ is set to i
  - either u is a node for $\bar{\imath}$, in which case i has a w.s. from every successor of u, and when $nb[u] = 0$ is reached, (4) makes sure $win[u]$ is set to i.

The complexity of the algorithm is based on the fact the procedure `propagate`(v, i) is called only once for each node v. Moreover, the loop (5) is executed at most $\sum_{v \in V} |pred[v]| = |E|$ times. Therefore this algoritm runs in time $\mathcal{O}(|V| + |E|)$.

$\dashv$ 10

2.4 Horn Formula

A (propositional) *Horn Formula* is a formula $\phi = \bigwedge_i \bigvee_j X_{i_j}$ in Conjunctive Normal Form where every disjunction $\bigvee_j X_{i_j}$ contains at most one positive literal. Therefore, Horn Formulas can be written as conjunctions of implications:

$$\neg X_{i_0} \vee \neg X_{i_1} \vee \ldots \vee \neg X_{i_k} \vee X_{i_{k+1}} \;\equiv\; (X_{i_0} \wedge X_{i_1} \wedge \ldots \wedge X_{i_k}) \longrightarrow X_{i_{k+1}}$$

$$\neg X_{i_0} \vee \neg X_{i_1} \vee \ldots \vee \neg X_{i_k} \;\equiv\; (X_{i_0} \wedge X_{i_1} \wedge \ldots \wedge X_{i_k}) \longrightarrow \bot$$

We consider the problem, given a Horn Formula ϕ, of determining whether there is a truth assignment for each variable occuring in ϕ, so that ϕ is true.

$$SAT\text{-}HORN = \{\phi \in \text{Horn Formula} : \ \phi \text{ is satisfiable}\}$$

It is well known that $SAT\text{-}HORN$ is P-time complete, (even P-time complete via logspace reduction) and solvable in linear time.
In fact, $GAME$ and $SAT\text{-}HORN$ are equivalent under log-lin reductions (reductions computable with logarithmic space and linear time). These reductions are so simple that one can see $GAME$ and $SAT\text{-}HORN$ as almost the *same* problem.

Theorem 11

$$GAME \equiv_{log-lin} SAT\text{-}HORN.$$

Proof of theorem 11:

$GAME \leq_{log-lin} SAT\text{-}HORN$ Given a finite game graph $\mathcal{G} = (V, V_I, E)$, we construct a Horn Formula $\phi_{\mathcal{G}}$ such that the minimal model of $\phi_{\mathcal{G}}$ is precisely the set W_I of all nodes where I has a winning strategy. In other words, v is a node where I has a w.s. iff the formula $\phi_{\mathcal{G}} \wedge (v \longrightarrow \bot)$ is unsatisfiable. This formula $\phi_{\mathcal{G}}$ is defined as the conjunction of

- all formulas $v \longrightarrow u$ for all edge $(u, v) \in E$ with $u \in V_I$,
- all formulas $(v_1 \wedge \ldots \wedge v_k) \longrightarrow u$ for all node $u \in V_{II}$ and $uE = \{v_1, \ldots, v_k\}$.

$SAT\text{-}HORN \leq_{log-lin} GAME$ Given a propositional Horn Formula $\phi(X_1, \ldots, X_k) = \bigwedge_{i \in I} C_i$, with Horn clauses C_i of the form $(X_{i_1} \wedge \ldots \wedge X_{i_m}) \longrightarrow X_{i_0}$; we define a finite game graph $\mathcal{G}_\phi$:

- the positions of player I are the propositional variables $X_1, \ldots, X_k$ occuring in ϕ,
- the positions of player II are the clauses C_i.
- Player I can move from a variable X to any clause C_i with head variable X,
- player II can move from a clause C_i to any variable occuring in the body of C_i.

Formally:

- $\mathcal{G}_\phi = (V, V_I, E)$
- $V_I = \{X_1, \ldots, X_k\}$
- $V \setminus V_I = V_{II} = \{C_i : i \in I\}$
- $E = \{(X, C) \in V_I \times V_{II} : X = head(C)\} \cup \{(C, X) \in V_{II} \times V_I : X \in body(C)\}$

Player I has a w.s. for $\mathcal{G}_\phi$ iff $\phi \models X$.

$\dashv$ 11

2.5 Alternating Turing Machines

See `http://www-mgi.informatik.rwth-aachen.de/Teaching/KT-WS01/index.html` *chapter 5: Alternating Machines* for the basic definitions. We essentially emphasize on the game theoretical approach of Alternating Turing Machine, and the following result:

$$\begin{array}{ccccccccc}
\text{LOGSPACE} & \subseteq & \text{PTIME} & \subseteq & \text{PSPACE} & \subseteq & \text{EXPTIME} & \subseteq & \text{EXPSPACE} \\
 & & \| & & \| & & \| & & \| \\
 & & \text{ALOGSPACE} & \subseteq & \text{APTIME} & \subseteq & \text{APSPACE} & \subseteq & \text{AEXPTIME}
\end{array}$$

2.6 The Game of Geography

The name of this games goes back to elementary school. It is a two player game where players take turns. Player I starts the game by naming a fixed city, say *Aachen*, then player II must find a city whose name starts with the last letter of the city previously named (e.g. New-York). Then I must find a city whose first letter is a k and so on... But no city can be reused throughout the game.
This childish game can be reformulated in terms of graph. We have a directed graph whose nodes are all the cities on earth, and such that there is an edge from a city u to a city v if the last letter of u is the firs letter of v. A play consists in a path through the graph that starts at the starting node i and ends at a node, say u, when there is no more node v such that there is an edge from u to v and v has not been visited yet.
This game can also be generalized to any given directed graph, and even to any given directed game graph $\mathcal{G} = (V, V_I, E)$. The game $\mathbb{G}eo\,(\mathcal{G}\ ,\ i)$ consists in:

(1) two players (I and II), starting at a fixed node i,

(2) at node $u \in V_I$ (respectively $u \in V_{II} = V \smallsetminus V_I$), player I (respectively player II) chooses a node $v \in uE$ that was never previously chosen by any of the players,

(3) the first player who gets trapped loses.

Clearly this game is finite (there are at most $|V|$ possible moves), hence it is determined. As for GAME we are interested in the following computational problem:

$$GEOGRAPHY = \{(\mathcal{G}, i) :\ \textit{player } I \textit{ has a w.s. in } \mathbb{G}eo\,(\mathcal{G}\ ,\ i)\}.$$

Although this problem is not that far from GAME. its computational complexity is somehow very different.

Theorem 12 *GEOGRAPHY is* **PSPACE**-*complete*

Proof of theorem 12:

GEOGRAPHY $\in$ **PSPACE**: the following algorithm for Alternating Turing Machine works in time $\mathcal{O}\left((|V_I| + |V_{II}| + |E|)^2\right)$.

input: $\mathcal{G} = (V_I, V_{II}, E)$, $u := i$
while $(V_I \cup V_{II}) \neq \emptyset$, **do**
  if $u \in V_I$, **then**
    if $uE = \emptyset$, **reject**,
    else (*Existential step*)
      Guess $v \in uE$,
      $u := v$, $V_I := V_I \smallsetminus \{v\}$, $E := E \smallsetminus \{(u, w) :\ w \in V\}$
  if $u \in V_{II}$, **then**
    if $uE = \emptyset$, **accept**,
    else (*Universal step*)
      Choose $v \in uE$,
      $u := v$, $V_{II} := V_{II} \smallsetminus \{v\}$, $E := E \smallsetminus \{(u, w) :\ w \in V\}$

GEOGRAPHY is **PSPACE**-hard: we reduce *QSAT* to *GEOGRAPHY*. Assume we are given a quantified Boolean expression

$$\phi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \psi$$

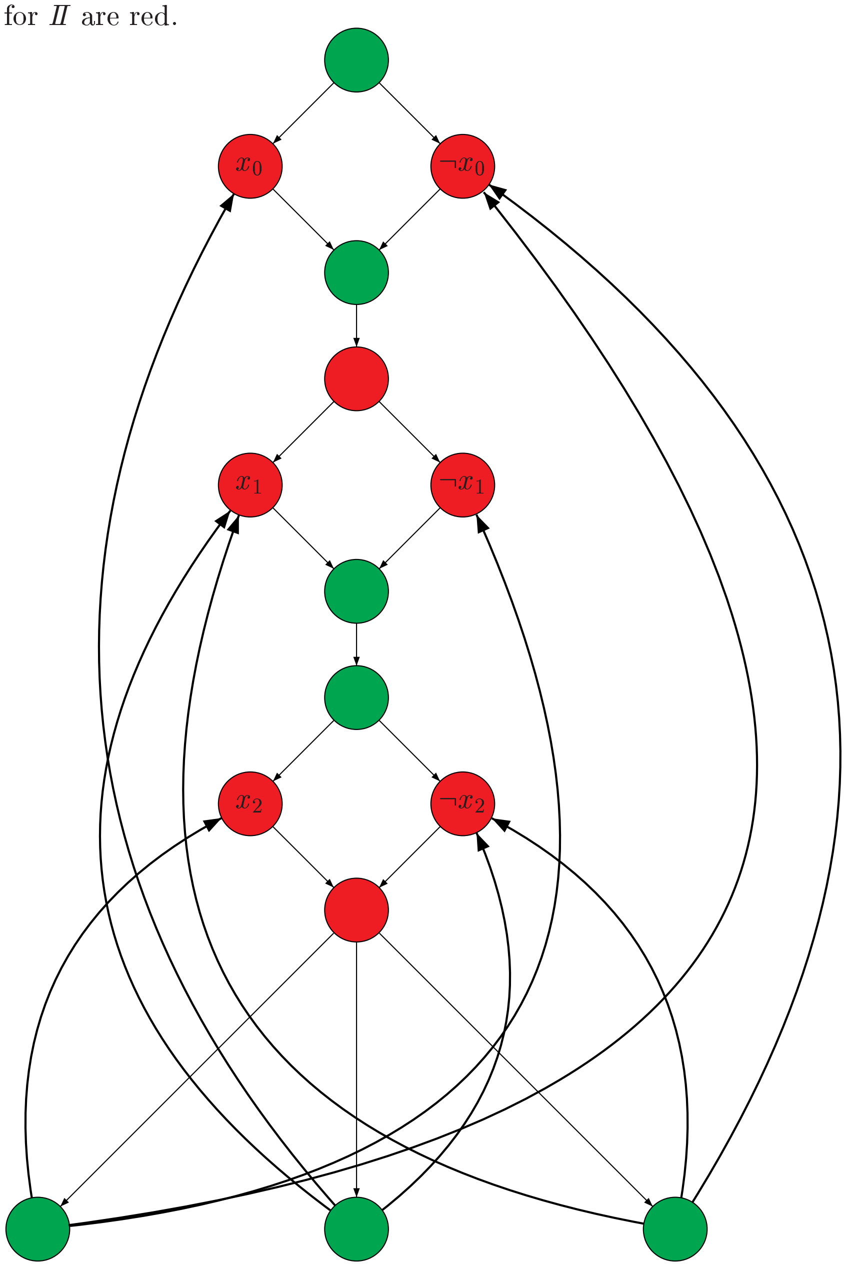
where the Q_i are quantifiers ($\exists$ or $\forall$), ψ is a formula from propositional logic in conjunctive normal form with variables among $x_1, \dots, x_n$:

$$\psi = \bigwedge_{i \in I} C_I, \quad \text{each } C_i \text{ being of the form } C_i = \bigvee_{j \in J_i} \bar{x_j} \text{ with } \bar{x_j} = x_j \text{ or } \neg x_j$$

We build a game graph $\mathcal{G}_\phi$ with an input node i such that

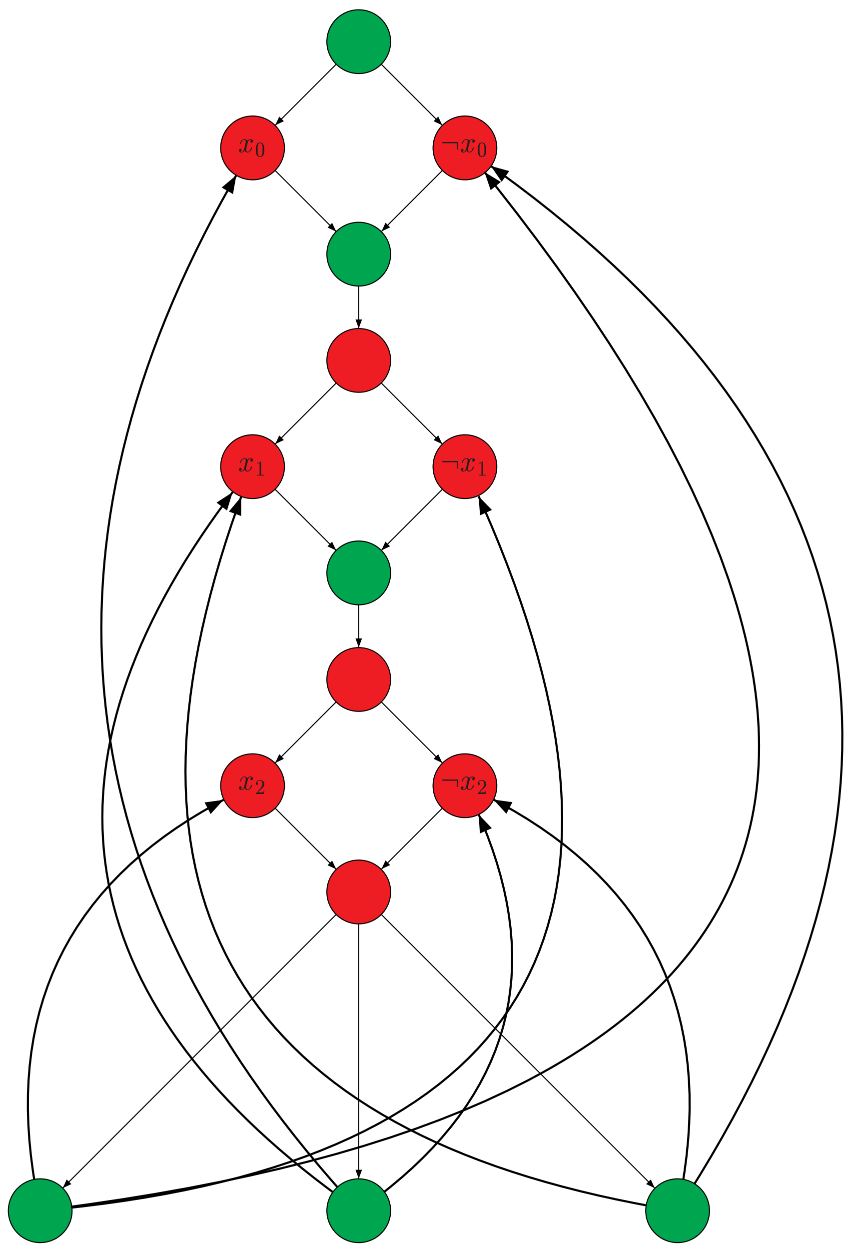
$$I \text{ has a w.s. in } \mathbb{G}eo\,(\mathcal{G}_\phi\ ,\ i) \iff \phi \text{ is satisfiable.}$$

The construction is explained by the following two examples where nodes for player I are green and nodes for II are red.



$\exists x_0 \ \ \forall x_1 \ \ \exists x_2 \quad ((\neg x_0 \vee \neg x_1 \vee x_2) \wedge (x_0 \vee x_1 \vee \neg x_2) \wedge (\neg x_0 \vee x_1 \vee \neg x_2))$

player I ($\exists$) has a w.s. strategy



$\exists x_0 \ \ \forall x_1 \ \ \forall x_2 \quad ((\neg x_0 \vee \neg x_1 \vee x_2) \wedge (x_0 \vee x_1 \vee \neg x_2) \wedge (\neg x_0 \vee x_1 \vee \neg x_2))$

player II ($\forall$) has a w.s. strategy

The generalization to any expression from QBF is straightforward. Each variable is replaced by a diamond like choice subgraph, forcing player I to choose between x_i and $\neg x_i$ if

$Q_i = \exists$, and forcing player *II* between these two nodes otherwise. one should have in mind that the idea is that a player chooses the side $\neg x_i$ if he decides x_i to be *true*, and the side x_i if he decides x_i to be *false*. This way, for each variable x_i, a choice between x_i and $\neg x_i$ is made by the right player. One should also notice that the move from a "literal" node is a move for player *II* that necessarily reaches the bottom node of the diamond like shape; a node that cannot be crossed twice. After the choice of literal is completed, player *II* is forced to pick up one of the clauses and, for this specific clause, player *I* finally must point out a literal. At this point, either there is no *true* literal in this clause and *I* loses, or there is some literal available, in which case next move (which is a move for *II*) is forbidden.

$\dashv$ 12