Solving infinite games on trees with back-edges

Aniruddh Gandhi[1] **Bakhadyr Khoussainov**[2] **Jiamou Liu**[3]

[1] Department of Computer Science, University of Auckland
Auckland, New Zealand.
Email: agan014@aucklanduni.ac.nz
[2] Department of Computer Science, University of Auckland
Auckland, New Zealand.
Email: bmk@cs.auckland.ac.nz
[3] School of Computing and Mathematical Sciences, Auckland University of Technology,
Auckland, New Zealand
Email: jiamou.liu@aut.ac.nz

Abstract

We study the computational complexity of solving the following problem: Given a game $\mathcal{G}$ played on a finite directed graph G, output all nodes in G from which a specific player wins the game $\mathcal{G}$. We provide algorithms for solving the above problem when the games have Büchi and parity winning conditions and the graph G is a tree with back-edges. The running time of the algorithm for Büchi games is $O(\min\{r\cdot m, \ell+m\})$ where m is the number of edges, ℓ is the sum of the distances from the root to all leaves and the parameter r is bounded by the height of the tree. The algorithm for parity has a running time of $O(\ell+m)$.

1 Introduction

In the last 10-20 years, there has been an extensive study of infinite games played on directed graphs. These games are natural models for reactive systems (9), concurrent and communication networks (12), and have close interactions with model checking, verification problems, automata and logic (3, 8, 11, 14). In this paper, we study turned-based games where both players have perfect information and each move is deterministic. Each such game involves a finite directed graph whose nodes are partitioned into two sets V_0 and V_1. There are two players: Player 0 and Player 1. They play the game by moving a token that is initially placed on some starting node u of the graph. The two players take turns to move the token along the edges of the graph while respecting the edge directions. If the token is placed on a node $u \in V_\sigma$, $\sigma \in \{0,1\}$, then Player σ chooses an outgoing edge (u,v) and places the token on node v. The players play the game indefinitely and thus producing a possibly infinite walk in the graph.

The goal of Player 0 is to produce a walk that satisfies the winning condition, while the goal of Player 1 is the opposite. If Player σ wins regardless of the moves by the other player, we say that the starting node u is a winning position of Player σ. An algorithm *solves* the game if it detects all the winning positions of Player 0. Formally, the *winning region problem* is defined as follows:

Given a game $\mathcal{G}$ played on a finite graph G, output all nodes in G from which Player 0 has a strategy to win the game $\mathcal{G}$.

This paper studies the algorithms for solving the winning region problem for Büchi and parity games played on trees with back-edges. Given a directed graph G, a *Büchi game* played on G specifies a set of target nodes T in G, and Player 0 wins the game from a node u if the player has a strategy to visit nodes in T infinitely often starting from u. It is well-known that the winning region problem for Büchi games can be solved in polynomial time (7, Ch.2). A *parity game* on G associates a priority $\rho(u) \in \mathbb{N}$ with every node u. Player 0 wins the game from a node u if the player has a strategy such that the minimum priority amongst all the nodes visited infinitely often in any play starting from u is even. Though intensely studied, polynomial time algorithms for solving the winning region problem for parity games remain unknown. Parity games are known to be in $\mathsf{NP}\cap\mathsf{Co}-\mathsf{NP}$ but not known to be in P (7, Ch.6).

While Büchi games are solved in polynomial time for arbitrary graphs, it is still unclear whether the known algorithms are optimal for a given class of games. This paper concentrates on algorithms for solving the winning region problem for games with Büchi winning conditions played on trees with back-edges. In this paper we will demonstrate that, even for this severely restricted subclass of infinite games, the analysis for the winning region problem can still be non-trivial and that the winning region may be rich in structure. We then apply our analysis for the case of Büchi games to solve the winning region problem for games with the parity winning condition played on trees with back-edges.

The classical algorithm for solving Büchi games uses iterations: at iteration i, the algorithm computes the set of nodes U_i from which Player 0 has a strategy to visit the target set i times (see Section 2 for a detailed description of the classical algorithm). It can be shown that the set of winning positions for Player 0 is $\bigcap_{i\in\mathbb{N}} U_i$ (7). Each iteration of the algorithm takes time $O(n+m)$, where m and n are the number of edges and nodes in $\mathcal{G}$ respectively. The algorithm performs at most n iterations and hence the running time of the algorithm is $O(n\cdot(n+m))$.

The classical algorithm has a seemingly repetitive nature as a node may be processed several times. Hence, it makes sense to carry out a more detailed analysis of Büchi games and see if the classical algorithm can be improved. For instance, the paper (5) investigates the class of graphs with constant out-degrees and shows that solving Büchi games played on such graphs can be done in $O(n^2/\log n)$ time. For graphs with unbounded out-degrees, the paper (4) presents an algorithm that runs in time $O(n \cdot m \cdot \log \delta(n)/\log n)$ where $\delta(n)$ is the out-degree of the game graph. These investigations suggest the idea of designing more efficient algorithms in specified classes of graphs such as trees with back-edges.

Trees with back-edges are widely used and studied in computer science. The paper (6) studies counterexamples in model checking whose transition diagrams are trees with back-edges. Furthermore, as discussed in (1), they form a natural class of directed graphs that has directed tree-width 1 and unbounded entanglement. Also, consider the trees generated by depth-first search. If the original graph has only tree-edges and back-edges but no cross-edges, then the algorithms described in this paper can be used. Another use of trees with back-edges is in μ-calculus where the syntax graph of a μ-calculus formula is a tree with back-edges (2). As pointed out in (2) a finite Kripke structure can be viewed as a tree with back-edges by performing a partial unraveling of the structure.

In our analysis, we use the notion of *snares* to classify the winning nodes of Player 0 as follows. Intuitively, a snare of rank 0 is a subtree from which Player 0 has a strategy to stay in the subtree forever and win the game. A snare of rank i, $i > 0$, is a subtree from which Player 1 may choose between two options: (a) staying in the subtree forever and losing the game, or (b) going to an $(i-1)$-snare. We show that the collection of all snares corresponds exactly to winning nodes of Player 0. In particular, we present an efficient algorithm that solves a Büchi game played on trees with back-edges. The algorithm runs in time $O(\min\{r \cdot m, \ell + m\})$ where r is the largest rank of a snare (whose value is at most the height of the tree) and ℓ is the external path length, i.e., sum of the distances from the root and all leaves, in the underlying tree.

We then give an algorithm for solving parity games played on trees with back-edges by reduction to Büchi games on trees with back-edges and prove the following theorem: any parity game played on trees with back-edges can be solved in time $O(\ell + m)$. J. Obdržálek in his work (13) (Chapter 3) outlines a proof that parity games played on trees with back-edges are solvable in polynomial time. The work does not provide a detailed analysis of the algorithm but rather concentrates on attacking the problem for the class of all parity games. The algorithm detects whether Player 0 wins the game from the root of the tree and it is claimed that this can be done in time $O(m)$ (where m is the number of edges in the graph).

We would like to point out that the running time of any algorithm for solving the winning region problem is heavily dependent on the data structures and underlying model of computation. In particular, under the reasonable assumption that trees with back-edges are encoded as binary strings it can be shown that $O(m \cdot log(m))$ bits are necessary to encode a tree with back-edges with $O(m)$ edges. This can be seen as follows: for a tree $\mathcal{T}$, we determine the main branch by starting from the root and always choosing the next node v such that v has the most number of nodes below it. Now consider a tree $\mathcal{T}$ with $O(m)$ edges such that the main branch has $m/2$ nodes and the first $m/4$ nodes of the main branch have exactly one offbranching leaf. Below the first $(m/4)^{th}$ nodes of the main branch, there is a full binary tree with $m/2$ nodes and $m/4$ leaves. If we now consider the class of trees with back-edges that can be obtained from $\mathcal{T}$ by adding exactly one back-edge per leaf, it can be seen that there are $O((m/4)^{(m/4)})$ trees with back-edges in this class. Hence we need at least $O(m \cdot log(m))$ bits to encode a tree with back-edges with $O(m)$ edges. This fact shows that any algorithm to solve the winning region problem for games on trees with back-edges must have a running time of al least $O(m \cdot log(m))$. Hence it is not clear how the time bound of $O(m)$ of (13) can be achieved when trees with back-edges are encoded using binary strings.

Notwithstanding the above observations, the algorithm of (13) can be modified to run in $O(h \cdot m)$ (where h is the height of the underlying tree). In this paper we give an alternative algorithm for solving parity games on trees with back-edges based on our analysis for Büchi games played on trees with back-edges which has a running time of $O(\ell + m)$. Note that since ℓ maybe much smaller as compared to $h \cdot m$, our algorithm performs better than the time bound of $O(h \cdot m)$ in many cases. Importantly, we clarify the data structures and model of computation used (see Section 3) and hence analyze the problem in greater detail.

Since the worst case performance of our algorithm for Büchi games is the same as that of the classical algorithm, we carried out experiments to compare the actual performance of the two algorithms. The experiments can be broadly divided into two categories: 1) comparing the average running times for games with a small n (number of nodes) and 2) for large n, we compare the running times of the two algorithms on games whose underlying trees belong to different classes of randomly generated trees. We found that our algorithm has significantly better performance than the classical algorithm in both these categories. Our algorithm outperforms the classical algorithm by an order of magnitude for even small values of n. The performance gap becomes even clearer for large values of n, where our algorithm again has a significantly better running time than the classical algorithm in all the classes of randomly generated trees used. In fact our algorithm has a linear growth in running time as compared to the quadratic growth in running time for the classical algorithm for all the classes of randomly generated games.

The rest of the paper is organized as follows. Section 2 describes the known algorithm for solving Büchi games. Section 3 lays out the basic framework and proves a normal form lemma (Lemma 2) for games played on trees with back-edges. Section 4 introduces the notion of snares and describes the algorithm that uses snares to solve Büchi games played on trees with back edges. In Section 5 we apply the algorithm to parity games played on trees with back-edges. Finally we present experimental results to support our claims in Section 6.

2 Games Played on Finite Directed Graphs

For background on games played on graphs, see e.g. (7). A *game* is a tuple $\mathcal{G} = (V_0, V_1, E, \mathrm{Win})$ where $G = (V_0 \cup V_1, E)$ forms a finite directed graph (called the underlying graph of $\mathcal{G}$), $V_0 \cap V_1 = \emptyset$ and the set $\mathrm{Win} \subseteq (V_0 \cup V_1)^\omega$. Nodes in the set V_0 are said to be *0-nodes* and nodes in the set V_1 are said to be *1-nodes*. We use V to denote $V_0 \cup V_1$ and $E(u)$ to denote the set $\{v \mid (u, v) \in E\}$. The game is played by Player 0 and Player 1 in rounds. Initially, a token is placed on some *initial node* $v \in V$. In each round, if the token is placed on a node $u \in V_\sigma$, where $\sigma \in \{0,1\}$, then Player σ selects a node $u' \in E(u)$ and moves the token from u to u'. The play continues indefinitely unless the token reaches a node u where $E(u) = \emptyset$. Thus, a *play* starting from u is a (possibly infinite) sequence of nodes $\pi = v_0 v_1 \dots$ such that $v_0 = u$ and for every $i \geq 0$, $v_{i+1} \in E(v_i)$. We use $\text{Plays}(G)$ to denote the set of all plays starting from any node in V. The *winning condition* of $\mathcal{G}$, denoted by Win, is a subset of $\text{Plays}(G)$ and Player 0 *wins* a play $\pi \in \text{Plays}(G)$ if $\pi \in \text{Win}$ and Player 1 wins π otherwise. We use $\text{Occ}(\pi)$ to denote the set of nodes that appear in π and $\text{Inf}(\pi)$ to denote the set of nodes that appear infinitely often in π.

A *reachability game* is a game $\mathcal{G} = (V_0, V_1, E, W_{\mathsf{reach}})$. The winning condition W_{reach} is determined by a set of *target nodes* $T \subseteq V$ such that

$$W_{\mathsf{reach}} = \{\pi \in \text{Plays}(G) \mid \text{Occ}(\pi) \cap T \neq \emptyset\}.$$

Hence, for convenience, we denote the reachability $\mathcal{G}$ by (V_0, V_1, E, T).

A *Büchi game* is a game $\mathcal{G} = (V_0, V_1, E, W_{\mathsf{buchi}})$. As in the case of reachability games, we specify a set of target nodes $T \subseteq V$. Then the Büchi winning condition can be expressed as follows:

$$W_{\mathsf{buchi}} = \{\pi \in \text{Plays}(G) \mid \text{Inf}(\pi) \cap T \neq \emptyset\}.$$

For convenience, we also denote a Büchi game by (V_0, V_1, E, T). It will be clear from the context whether reachability or Büchi games are considered.

A *parity game* is a game $\mathcal{G} = (V_0, V_1, E, W_{\mathsf{parity}})$ along with a *priority function* $\rho : V \to \mathbb{N}$. The winning condition is expressed as follows:

$$W_{\mathsf{parity}} = \{\pi \in \text{Plays}(G) \mid \min\{\rho(v) \mid v \in \text{Inf}(\pi)\} \text{ is even}\}.$$

We use the tuple (V_0, V_1, E, ρ) to denote a parity game.

When playing a game, the players use *strategies* to determine the next move from the previous moves. Formally, a *strategy* for Player σ (or a *σ-strategy*), where $\sigma \in \{0,1\}$, is a function $f_\sigma : V^* V_\sigma \to V$ such that if $f_\sigma(v_1 v_2 \dots v_i) = w$ then $(v_i, w) \in E$ (here $V^* V_\sigma$ denotes the set of all finite paths in the graph G with the last node in V_σ). A play $\pi = v_0 v_1 \dots$ is *consistent* with f_σ if $v_{i+1} = f_\sigma(v_0 v_1 \dots v_i)$ whenever $v_i \in V_\sigma$ ($i \geq 0$). A strategy f_σ is *winning* for Player σ on v if Player σ wins all plays starting from v consistent with f_σ. If Player σ has a winning strategy on u, we say Player σ *wins* the game on u, or u is a *winning position* for Player σ. The *σ-winning region*, denoted by W_σ, is the set of all winning positions for Player σ. Note that $W_0 \cap W_1 = \emptyset$. A strategy f_σ for Player σ is called *memoryless* if $f_\sigma(v_1 v_2 \dots v_i) = f_\sigma(v_i)$ for all $(v_1 v_2 \dots v_i) \in V^* V_\sigma$.

For the sake of simplicity, we assume $E(u) \neq \emptyset$ for all $u \in V$ in any game $\mathcal{G}$. This can be achieved by performing the following whenever $E(u) = \emptyset$: we add two extra vertices u_1, u_2 such that $E(u) = u_1$, $E(u_1) = u_2$ and $E(u_2) = u_1$. In the case of reachability and Büchi games we declare $u_1, u_2 \notin T$ and for parity games we declare u_1, u_2 to have odd priorities. Note that this does not change the winning region of Player 0 for any of the winning conditions described earlier. Hence we may assume that $\text{Plays}(G) \subseteq V^\omega$.

For the rest of this paper, we will assume that our underlying model of computation is a *random access machine* (RAM) and that manipulating registers/pointers of logarithmic lengths takes constant time.

A game *enjoys determinacy* if $W_0 \cup W_1 = V$. The determinacy result in the next theorem is a special case of the well-known Borel determinacy theorem which states all Borel games enjoy determinacy.

Theorem 1. (10)(7) *Reachability, Büchi and parity games enjoy memoryless determinacy. Moreover, computing W_0 and W_1 takes time $O(m+n)$ for a reachability game and $O(n \cdot (m+n))$ for a Büchi game, where m, n are respectively the number of edges and nodes in the underlying graph.*

By the above theorem, we are justified in restricting ourselves to memoryless strategies for such games and in this paper we shall only consider memoryless strategies. By *solving a game*, we mean to provide an algorithm that takes as input a game $\mathcal{G}$, and outputs all nodes in W_0. We briefly describe the classical algorithms that solve reachability and Büchi games.

We first provide the algorithm for solving reachability games. Suppose $\mathcal{G}$ is a reachability game. For $Y \subseteq V$, let

$$\begin{aligned}\text{Pre}(Y) = &\{v \in V_0 \mid \exists u : (v,u) \in E \ \wedge \ u \in Y\} \\ &\cup \ \{v \in V_1 \mid \forall u : (v,u) \in E \to u \in Y\}.\end{aligned}$$

The algorithm computes a sequence of sets $T_0, T_1, T_2 \dots$ where $T_0 = T$, and for $i > 0$, $T_i = \text{Pre}(T_{i-1}) \cup T_{i-1}$. Since the graph is finite, we have $T_s = T_{s+1}$ for some $s \in \mathbb{N}$. A node v is a winning position for Player 0 if and only if $v \in T_s$. This algorithm can be implemented to run in $O(m+n)$ by performing a reverse breadth first search (starting from T) to compute the out-degrees of the nodes followed by a second reverse breadth first search to compute the T_i's exploiting the out-degrees computed previously. The reader is directed to (7, Ch.2) for details. We refer to this algorithm as the *reach algorithm*. Let G be the underlying graph of the game. For any set $X \subseteq V$, we use $\mathsf{Reach}_\sigma(X, G)$ to denote the σ-winning region for the reachability game (V_0, V_1, E, X). In other words, from any node in $\mathsf{Reach}_\sigma(X, G)$, Player σ has a strategy that forces any play starting from this node to visit X. Hence in the above algorithm we have $T_s = \mathsf{Reach}_0(T, G)$.

We now present the classical algorithm for solving Büchi games. Suppose $\mathcal{G}$ is a Büchi game. Compute the sequences of sets $T_0, T_1, \dots$, $R_0, R_1, \dots$ and $U_0, U_1, \dots$ as follows: Let $T_0 = T$. Suppose T_i is defined for $i \geq 0$. Set $R_i = \mathsf{Reach}_0(T_i, G)$ and $U_i = V \setminus R_i$. Set $T_{i+1} = T_i \setminus \mathsf{Reach}_1(U_i, G)$. Hence we have $T_0 \supseteq T_1 \supseteq T_2 \supseteq \dots$. The process terminates when we have $T_s = T_{s+1}$ for some $s \in \mathbb{N}$. A node v is a winning position for Player 0 if and only if $v \in \mathsf{Reach}_0(T_s, G)$. The algorithm takes time $O(n \cdot (m+n))$. See (7, Ch.2) for details.

3 Trees with back-edges

We consider rooted directed trees where all edges are directed away from the root. All terminologies on trees are standard. The ancestor relation on a tree $\mathcal{T}$ is denoted by $\leq_{\mathcal{T}}$ and the root is its least element. For $u \leq_{\mathcal{T}} v$, let $\text{Path}[u,v] = \{x \mid u \leq_{\mathcal{T}} x \leq_{\mathcal{T}} v\}$. The *level* $\mathsf{lev}(u)$ of a node $u \in V$ is the length of the unique path from the root to u. The *height* h of the tree is $\max\{\mathsf{lev}(v) \mid v \in V\}$. The *external path length* ℓ is $\sum\{\mathsf{lev}(v) \mid v \text{ is a leaf in } \mathcal{T}\}$.

Definition 1. *A directed graph $G = (V, E)$ is a* tree with back-edges *if its edge relation E can be partitioned into two sets E^{T} and E^{B} where $\mathcal{T} = (V, E^{\mathsf{T}})$ is a rooted directed tree and all edges in E^{B} are of the form (v, u) where $u <_{\mathcal{T}} v$. Edges in E^{B} are called* back-edges*. We denote a tree with back-edges by $(V, E^{\mathsf{T}} \cup E^{\mathsf{B}})$. We refer to leaves of $\mathcal{T}$ as* leaves of G*. A Büchi game is* played on a tree with back-edges *if its underlying graph is a tree with back-edges.*

Let $G = (V, E^{\mathsf{T}} \cup E^{\mathsf{B}})$ be a tree with back-edges. A *subtree* of G is a subgraph of G that is also a tree. In particular, all the edges of a subtree are from E^{T} and the root of the subtree is not necessarily the root of G. A *subtree with back-edges* consists of a subtree and all induced back-edges on the subtree. We use $\mathfrak{B}$ to denote the class of all Büchi games played on trees with back-edges. A game $\mathcal{G} \in \mathfrak{B}$ is denoted by the tuple $(V_0, V_1, E^{\mathsf{T}}, E^{\mathsf{B}}, T)$ where $T \subseteq V$ are target nodes. Recall that we may assume without loss of generality that for any node $u \in V$, we have $E^{\mathsf{T}}(u) \cup E^{\mathsf{B}}(u) \neq \emptyset$.

In the rest of the paper we will present our algorithm for solving the winning region problem for games in $\mathfrak{B}$. Our claim on the running time of the algorithm depends on the following assumptions on the data structures and the underlying model of computation. A node u in a game $\mathcal{G} \in \mathfrak{B}$ is stored as the tuple

$$(p(u), \mathrm{tar}(u), \mathrm{pos}(u), \mathrm{Ch}(u), \mathrm{InBk}(u), \mathrm{OutBk}(u)),$$

where $p(u)$ is a pointer to the parent of u, $\mathrm{tar}(u)$ is **true** if and only if $u \in T$, $\mathrm{pos}(u) = \sigma$ if and only if $u \in V_\sigma$, $\mathrm{Ch}(u)$ is a list of children of u, $\mathrm{InBk}(u)$ is a list of incoming back-edges into u, $\mathrm{OutBk}(u)$ is a list of outgoing back-edges from u. Note that as pointed out in section 2, our underlying model of computation is a random access machine and manipulating registers (of logarithmic lengths) takes constant time. Hence, accessing $p(u)$, $\mathrm{tar}(u)$, $\mathrm{pos}(u)$ as well as the first elements of $\mathrm{Ch}(u)$, $\mathrm{InBk}(u)$ and $\mathrm{OutBk}(u)$ takes constant time. In the following we define a canonical form for all games $\mathcal{G} \in \mathfrak{B}$.

Definition 2. *A Büchi game $\mathcal{G} \in \mathfrak{B}$ is* reduced *if the following conditions hold:*

- *For all $(u, v) \in E^{\mathsf{B}}$, u is a leaf in the underlying tree with back-edges $(V, E^{\mathsf{T}} \cup E^{\mathsf{B}})$.*
- *All target nodes are leaves.*
- *Each leaf in $(V, E^{\mathsf{T}} \cup E^{\mathsf{B}})$ has exactly one outgoing back-edge.*

The following lemma is easy to see.

Lemma 1. *Suppose $\mathcal{G}$ is a reduced Büchi game and π is a play in $\mathcal{G}$. Let R be the set of leaves that are visited infinitely often by π. Then we have*

$$\mathrm{Inf}(\pi) = \bigcup \{Path[v, u] \mid (u, v) \in E^{\mathsf{B}}, u \in R\}.$$

The next lemma reduces solving games in the class $\mathfrak{B}$ to solving games that are reduced.

Lemma 2. *Given a Büchi game $\mathcal{G} = (V_0, V_1, E_1^{\mathsf{T}}, E_1^{\mathsf{B}}, T) \in \mathfrak{B}$, there exists a reduced game $\mathrm{Rd}(\mathcal{G}) = (U_0, U_1, E_2^{\mathsf{T}}, E_2^{\mathsf{B}}, S)$ such that*

- *$V \subseteq U$ and $|U| \leq |V| + |E_1^{\mathsf{B}}|$.*
- *A node $v \in V$ is winning for Player 0 in $\mathcal{G}$ if and only if v is winning for Player 0 in $\mathrm{Rd}(\mathcal{G})$.*
- *$\mathrm{Rd}(\mathcal{G})$ is constructed from $\mathcal{G}$ in time $O(|E_1^{\mathsf{T}} \cup E_1^{\mathsf{B}}|)$.*

Proof. The game $\mathrm{Rd}(\mathcal{G})$ is constructed from $\mathcal{G}$ as follows :

1. For each back-edge $(u, v) \in E_1^{\mathsf{B}}$, add a new leaf $\alpha(u, v)$ and subdivide the edge (u, v) into $(u, \alpha(u, v))$ and $(\alpha(u, v), v)$.
2. $S = \{\alpha(u, v) \mid \mathrm{Path}[v, u] \cap T \neq \emptyset\}$.

See Fig. 1 for an example. It is easy to see that $\mathrm{Rd}(\mathcal{G})$ is a reduced game and that $V \subseteq U$ and $|U| \leq |V| + |E_1^{\mathsf{B}}|$. We now need to prove that a node $v \in V$ is winning for Player 0 in $\mathcal{G}$ if and only if v is winning for Player 0 in $\mathrm{Rd}(\mathcal{G})$. The proof for this is quite technical and is omitted due to space restrictions.

The *target level* of a node $u \in V$ is the number of target nodes that occur on the unique path from the root to u. To construct $\mathrm{Rd}(\mathcal{G})$ from $\mathcal{G}$, we first compute the levels and target levels for all nodes in $V_0 \cup V_1$. This can be done by a preorder traversal of the tree starting from the root. We use $k(u)$ and $\ell(u)$ to denote resp. the target level and level of u. The value of $k(u)$ and $\ell(u)$ are set to 0 when u is the root. We increment the ℓ-value by 1 as the tree traversal visits a node of a higher level. If a target node u is visited, we increase the k-value by 1; see Algorithm 1. The algorithm is executed with parameters $(r, 0)$ where r is the root of $(V, E_1^{\mathsf{T}} \cup E_1^{\mathsf{B}})$.

Algorithm 1 AssignLabel(u, i).

```
if tar(u) then k(u) ← i + 1
else k(u) ← i end if
for v ∈ Ch(u) do
    ℓ(v) ← ℓ(u) + 1
    Run AssignLabel(v, k(u)).
end for
```

The algorithm then copies the nodes and edges in the tree (V, E_1^{T}) to $\mathrm{Rd}(\mathcal{G})$. When a back-edge (u, v) is detected, the algorithm creates a new node $\alpha(u, v)$, and connects u (resp. $\alpha(u, v)$) with $\alpha(u, v)$ (resp. v). Finally, the node $\alpha(u, v)$ is set as a target in S if u and v have different target levels. Algorithm 1 runs in time $O(|V|)$ because the algorithm visits each node in the tree exactly once. The construction of $\mathrm{Rd}(\mathcal{G})$ takes $O(|E_1^{\mathsf{T}} \cup E_1^{\mathsf{B}}|)$ time because each edge (tree edge or back-edge) in $\mathcal{G}$ is visited exactly once. □

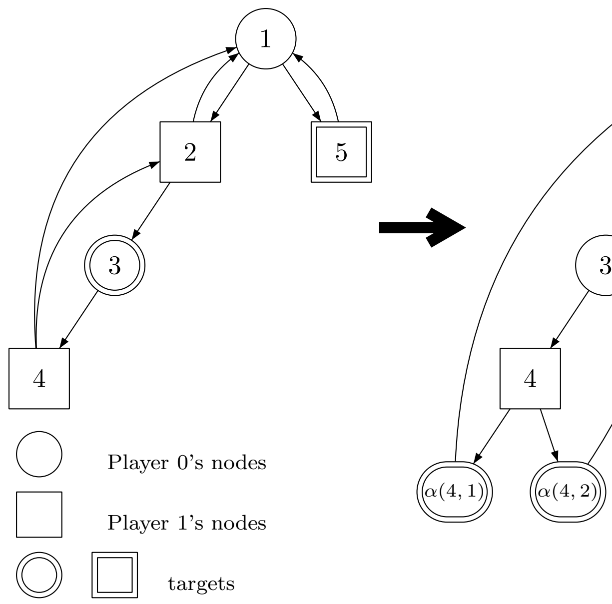
4 Solving Büchi games played on trees with back-edges

Our goal is to describe an algorithm that solves a Büchi game played on trees with back-edges. By Lemma 2, it suffices to describe an algorithm that solves reduced Büchi games.

4.1 Snares

Let $\mathcal{G} = (V_0, V_1, E^{\mathsf{T}}, E^{\mathsf{B}}, T)$ be a reduced Büchi game. Let u be a leaf in the tree $\mathcal{T} = (V, E^{\mathsf{T}})$. Since $\mathcal{G}$ is reduced, we abuse the notation by writing $E^{\mathsf{B}}(u)$ for the unique node v such that $(u, v) \in E^{\mathsf{B}}$. We will often identify a subset $S \subseteq V$ with the subgraph of $G = (V_0 \cup V_1, E^{\mathsf{T}} \cup E^{\mathsf{B}})$ induced by S. Recall

Figure 1: Example of a Büchi game played on a tree with back edges and the equivalent reduced game.



that W_0 denotes the 0-winning region of $\mathcal{G}$. We now give a refined analysis of the set W_0 by introducing the notion of snares. Essentially, we will construct a sequence $S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots$ of subsets of nodes in V that contain all nodes in W_0.

Definition 3. *For a subset $S \subseteq V$, a* snare strategy *in S is a strategy for Player 0 such that all plays consistent with the strategy starting from a node in S stay in S forever. A* 0-snare *is a subtree S of the tree (V, E^{T}) such that all leaves in S are targets and Player 0 has a snare strategy in S.*

Note that by definition, Player 0 wins the Büchi game $\mathcal{G}$ from any nodes in a 0-snare. On the other hand, there can be winning positions of Player 0 that do not belong to any 0-snares. To capture the entire winning region W_0 of Player 0, we inductively define the notion of i-snares for all $i \in \mathbb{N}$. Let S_0 be the set $\{u \mid u \text{ belongs to a 0-snare in } \mathcal{G}\}$, and let $T_0 = T$. For $i > 0$, define the set

$$T_i = \{x \mid E^{\mathsf{B}}(x) \in S_{i-1}\},$$

where the sets $S_1, S_2, \ldots$ are defined inductively as follows.

Definition 4. *For $i > 0$, an* i-snare *is a subtree S of the tree (V, E^{T}) such that*

- *All leaves of S are in $T \cup T_i$.*
- *From any node $v \in S$, Player 0 has a snare strategy in $S \cup S_{i-1}$.*

We let S_i denote the set of all nodes that are in an i-snare.

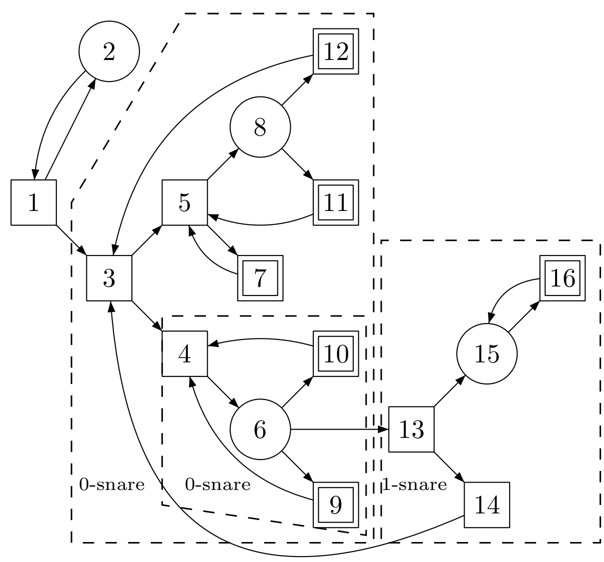
Note that the sequence of nodes $S_0, S_1, \ldots$ satisfies that

$$S_0 \subseteq S_1 \subseteq S_2 \subseteq \cdots$$

The *snare rank* of the node $u \in V$ is $\min\{i \mid u \in S_i\}$. The *snare rank* of the Büchi game $\mathcal{G}$ is the maximum snare rank of the nodes in $\mathcal{G}$. From now on we always use r to denote the snare rank of $\mathcal{G}$. Note that the definition requires that $\mathcal{G}$ is a reduced game. When $\mathcal{G}$ is not reduced, the snare rank of $\mathcal{G}$ is defined on the game $\mathsf{Rd}(\mathcal{G})$. We use the term *snare* to refer to an i-snare for some $i \in \{0, \ldots, r\}$.

As an example, consider the Büchi game shown in Fig. 2. The 0-snares and 1-snare are shown in the figure. Note that the subtree with back-edges rooted at 5 and containing $7, 8, 11, 12$ is *not* a 0-snare. Node 13 is the root of a 1-snare. Note that from 13, Player 1 has two options: 1) move to 15 and lose the game since 16 is a target node *or* 2) move to 14 and lose the game since the play must move to the 0-snare rooted at 3.

Figure 2: Example of a Büchi game with snares shown.



Proposition 1. *The snare rank r of $\mathcal{G}$ is bounded by the height h of the tree (V, E^{T}).*

Proof. Let H_r be a snare of rank r with root u_r. It is clear from the definition of an i-snare that there must be a leaf $x \in H_r$ such that there is a back-edge from x to some node $v \in S_{r-1}$. Also note that all nodes in Path$[u_r, x]$ belong to H_r and therefore $v <_{\mathcal{T}} u_r$.

Let u_{r-1} be the root of the snare H_{r-1} which contains v ($u_{r-1} \leq_{\mathcal{T}} v <_{\mathcal{T}} u_r$). Since $u_{r-1} <_{\mathcal{T}} u_r$, we have $\mathsf{lev}(u_{r-1}) < \mathsf{lev}(u_r)$. We can apply a similar argument to the snare H_{r-1} to find a snare H_{r-2} which has a root $u_{r-2} <_{\mathcal{T}} u_{r-1}$ ($\mathsf{lev}(u_{r-2}) < \mathsf{lev}(u_{r-1})$). In this manner we find a sequence of nodes $u_r, u_{r-1}, u_{r-2}, ..$ such that each u_i is the root of an i-snare and for each i we have $\mathsf{lev}(u_{i-1}) < \mathsf{lev}(u_i)$. Since the tree (V, E^{T}) has height h, the value of r is bounded by h. ☐

The following lemmas reduce the problem of solving a reduced Büchi game to computing snares.

Lemma 3. *If $u_0 \in S_i$ for some $i \in \{0, \dots, r\}$, then $u_0 \in W_0$.*

Proof. We prove the lemma by induction on i. Suppose u_0 belongs to an i-snare S for some $i > 0$. Let f be a snare strategy in $S \cup S_{i-1}$. Then a play starting from u_0 that is consistent with f either stays in S forever or goes to an $(i-1)$-snare. If all plays starting from u_0 consistent with f reach S_{i-1}, then $u_0 \in W_0$ by the inductive assumption.

Suppose $\pi = u_0, u_1, ..$ is a play consistent with f that does not reach S_{i-1}. Since the game $\mathcal{G}$ is reduced, the play π eventually reaches a leaf node. Let u_{j_0} be the first leaf node visited by π. Then $u_{j_0} \in T_i \cup T$. If $E^{\mathsf{B}}(u_{j_0}) \in S_{i-1}$, then $u_{j_0+1} \in S_{i-1}$ which is impossible by assumption. Hence, by definition, u_{j_0} is a target node. Now applying the same argument to the play starting from u_{j_0+1}, we obtain u_{j_1} which is the second leaf node visited by π. Continuing this argument, we obtain a sequence of target nodes $u_{j_0}, u_{j_1}, u_{j_2}, \dots$ of nodes in π where $j_0 < j_1 < j_2 < \cdots$. Hence π is a winning play for Player 0. This concludes the proof that $u_0 \in W_0$. ☐

Now our goal is to show that every node in W_0 belongs to some snare. We need the following definition:

Definition 5. *Let f be a winning strategy for Player 0 on a node u. We define the* tree induced by f on u *as a subtree $\mathcal{T}_u^f = (V_u^f, \leq_T)$ of the underlying tree (V, E^{T}) such that the root of $\mathcal{T}_u^f$ is u and for every node $w >_T u$, $w \in V_u^f$ whenever*

- *$w = f(x)$ for some $x \in V_0 \cap V_u^f$, or*
- *$w \in E^{\mathsf{T}}(x)$ for some $x \in V_1 \cap V_u^f$.*

The edge relation $E_{u,f}$ is E^{T} restricted to V_u^f.

Lemma 4. *If $u \in W_0$ then $u \in S_i$ for some $i \in \{0, \dots, r\}$.*

Proof. Suppose u does not belong to any snare. Assume for the sake of contradiction that $u \in W_0$. Let f be the winning strategy for Player 0 starting from u. Consider the tree $\mathcal{T}_u^f$ induced by f on u. If for all leaves v in $\mathcal{T}_u^f$ we have $E^{\mathsf{B}}(v) \in S_i$ for some $i \in \mathbb{N}$, then by definition $\mathcal{T}_u^f$ is a $(j+1)$-snare where $j = \max\{i \mid v \text{ is a leaf in } \mathcal{T}_u^f \text{ and } E^{\mathsf{B}}(v) \text{ has snare rank } i\}$. This is in contradiction with the assumption. Therefore, let L be the non-empty set containing all leaves v of $\mathcal{T}_u^f$ such that $E^{\mathsf{B}}(v)$ does not belong to any snare. We define a node u_1 as follows.

- If all nodes in L are targets, then there is some $v \in L$ with $E^{\mathsf{B}}(v) <_{\mathcal{T}} u$ as otherwise $\mathcal{T}_u^f$ is a 0-snare. In this case, let $u_1 = E^{\mathsf{B}}(v)$.
- Suppose a node $v \in L$ is not a target. If $E^{\mathsf{B}}(v) \geq_{\mathcal{T}} u$, then the path $u, v, E^{\mathsf{B}}(v), v, E^{\mathsf{B}}(v), \cdots$ defines a play consistent with f but is winning for Player 1. Thus $E^{\mathsf{B}}(v) <_{\mathcal{T}} u$ and we let $u_1 = E^{\mathsf{B}}(v)$.

Note that in both cases, $u_1 <_{\mathcal{T}} u$ and u_1 does not belong to any snare. Also since f is a winning strategy for Player 0 , u_1 is a winning position for Player 0. Applying the same argument as above, we obtain the sequence $u >_{\mathcal{T}} u_1 >_{\mathcal{T}} u_2 >_{\mathcal{T}} \cdots$ such that no node in this sequence is in a snare. The sequence is finite and let u_k be the last node in it. Any play consistent with f starting at u_k stays in the induced tree $\mathcal{T}_{u_k}^f$ forever. Therefore if all leaves in $\mathcal{T}_{u_k}^f$ are targets, u_k belongs to a 0-snare which is impossible. Hence let y be the leaf in $\mathcal{T}_{u_k}^f$ that is not a target. Then the sequence $u_k, y, E^{\mathsf{B}}(y), y, E^{\mathsf{B}}(y), \cdots$ defines a winning play that is consistent with f and is winning for Player 1. This is in contradiction with the fact that f is a winning strategy for Player 0. ☐

Combining Lemma 3 and Lemma 4, we have the following:

Theorem 2. *For any reduced Büchi game $\mathcal{G}$, the winning region W_0 of Player 0 coincide with the set S_r, where r is the snare rank of $\mathcal{G}$.*

4.2 Finding Snares

Our goal is to present an algorithm that computes all snares in the reduced game $\mathcal{G}$. Let $\mathcal{T} = (V, E^{\mathsf{T}})$. Recall from Section 2 that for any $X \subseteq V$, Player 0 has a strategy to force any play into X from $\mathsf{Reach}_0(X, \mathcal{T})$ by using only tree-edges. For simplicity, we denote $\mathsf{Reach}_0(X, \mathcal{T})$ by $\mathsf{Reach}(X)$. Note that if $v \in S_0$ then $v \in \mathit{Reach}(T)$.

For every node $v \in \mathsf{Reach}(T)$, we define a value $\mathsf{b}_0(v) \in \mathbb{N}$ inductively as follows:

- If v is a leaf, let $\mathsf{b}_0(v) = \mathsf{lev}(E^{\mathsf{B}}(v))$. Notice that $v \in T$.
- If v is an internal node and $v \in V_0$, let $\mathsf{b}_0(v) = \max\{\mathsf{b}_0(u) \mid u \in E^{\mathsf{T}}(v) \cap \mathsf{Reach}(T)\}$.
- If v is an internal node and $v \in V_1$, let $\mathsf{b}_0(v) = \min\{\mathsf{b}_0(u) \mid u \in E^{\mathsf{T}}(v)\}$.

Intuitively, the value $b_0(v)$ represents the level in the tree the game may arrives in if the play starts from v and goes through one back-edge, when the players adopt the following strategy:

- Player 0 would like to stay as close to the leaves as possible.
- Player 1 would like to stay as close to the root as possible.

For any node $v \in V$, and a strategy f for Player 0, let

$$\mathsf{b}_0(f, v) = \min\{\mathsf{lev}(E^{\mathsf{B}}(w)) \mid w \text{ is a leaf in the tree } \mathcal{T}_v^f\}.$$

Intuitively, $\mathsf{b}_0(f, v)$ is the largest number k with the following property: the first leaf w that appears in any play starting from v and consistent with f has $\mathsf{lev}(E^{\mathsf{B}}(w)) \geq k$. In other words if Player 0 adopts the strategy f starting from v, then $b_0(f, v)$ is the closest level to the root that Player 1 can guarantee to move to after following exactly one back-edge. Note that when v is itself a leaf, $\mathsf{b}_0(f, v) = \mathsf{lev}(E^{\mathsf{B}}(v))$ for any strategy f. For any $X \subseteq V$ and $v \in \mathsf{Reach}(X)$, let $\mathcal{S}_{\mathsf{Reach}}(X, v)$ denote the set of all 0-strategies that force any play into X from v. The next lemma relates $b_0(v)$ with $b_0(f, v)$ for all $f \in \mathcal{S}_{\mathsf{Reach}}(T, v)$.

Lemma 5. *For every $v \in \mathsf{Reach}(T)$, $\mathsf{b}_0(v) = \max\{\mathsf{b}_0(f, v) \mid f \in \mathcal{S}_{\mathsf{Reach}}(T, v)\}$.*

The proof of the above lemma is by induction on the level of v.

For every $u \in \mathsf{Reach}(T)$, let

$$S_0(u) = \{v \geq_{\mathcal{T}} u \mid \forall w \in \text{Path}[u, v] : \mathsf{b}_0(w) \geq \mathsf{lev}(u)\}.$$

The following lemma provides a way to check if a node belongs to a 0-snare.

Lemma 6. *For every $v \in \mathsf{Reach}(T)$, v belongs to a 0-snare if and only if $v \in S_0(u)$ for some $u \leq_{\mathcal{T}} v$ such that $\mathsf{b}_0(u) \geq \mathsf{lev}(u)$.*

Proof. Suppose v belongs to a 0-snare S that is rooted at some node u. Let f be the snare strategy in S. Since all leaves of S are targets, for all $w \in \text{Path}[u, v]$, $f \in \mathcal{S}_{\mathsf{Reach}}(T, w)$, and by definition of a snare strategy, all plays starting from w that are consistent with f will stay in S forever. In particular, we have $\mathsf{b}_0(f, u) \geq \mathsf{lev}(u)$. By Lemma 5, $\mathsf{b}_0(u) \geq \mathsf{b}_0(f, u) \geq \mathsf{lev}(u)$. Furthermore, for all nodes $w \in \text{Path}[u, v]$, $\mathsf{b}_0(w) \geq \mathsf{b}_0(f, w) \geq \mathsf{lev}(u)$. Hence, $v \in S_0(u)$.

Conversely, let $u \in \mathsf{Reach}(T)$ be such that $\mathsf{b}_0(u) \geq \mathsf{lev}(u)$. We prove that the set $S_0(u)$ forms a 0-snare. It is clear that all leaves in $S_0(u)$ are targets. By Lemma 5, for every node $v \in S_0(u)$, there is a strategy f_v such that any leave w in the tree $\mathcal{T}_v^{f_v}$ is a target and $\mathsf{lev}(E^{\mathsf{B}}(w)) \geq \mathsf{b}_0(u)$. Therefore we define a strategy for Player 0 such that $g(v) = f_v(v)$ for all node $v \in V_0 \cap S_0(u)$. Now let π be a play starting from some $v \in S_0(u)$ and consistent with g. Whenever π reaches a leaf w, we have $\mathsf{lev}(E^{\mathsf{B}}(w)) \geq \mathsf{lev}(u)$ and thus $E^{\mathsf{B}}(w) \in S_0(u)$. Hence any play starting from $S_0(u)$ and consistent with g will stay in $S_0(u)$ forever. This means $S_0(u)$ is a 0-snare. ☐

Lemma 6 gives us a way to check if a node belongs to a 0-snare. In particular, the following equality holds:

$$S_0 = \bigcup \Big\{ S_0(u) \mid u \in \mathsf{Reach}(T), \mathsf{b}_0(u) \geq \mathsf{lev}(u) \Big\}.$$

We now apply our reasoning above to i-snares where $i > 0$. For $i > 0$, recall that $T_i = \{w \mid E^{\mathsf{B}}(w) \in S_{i-1}\}$. Note that a node belongs to an i-snare only if it belongs to $\mathsf{Reach}(T \cup T_i)$. Recall that h is the height of the tree $\mathcal{T}$. We inductively define a function $\mathsf{b}_i : \mathsf{Reach}(T \cup T_i) \to \{0, \dots, h\}$ in the same way as b_0 with the following difference: If $v \in T_i$, let $\mathsf{b}_i(v) = h$.

For any node $v \in V$ and a strategy f, let $\mathsf{b}_i(f, v) = h$ if all leaves in the tree $\mathcal{T}_u^f$ belong to T_i and let $\mathsf{b}_i(f, v) = \mathsf{b}_0(f, v)$ otherwise. In other words, $\mathsf{b}_i(f, v)$ is the largest number $k \in \{0, \dots, h\}$ with the following property: the first leaf w that appears in any play starting from v and consistent with f has either $E^{\mathsf{B}}(w) \in S_{i-1}$ or $\mathsf{lev}(E^{\mathsf{B}}(u)) \geq k$. In the same way as the proof of Lemma 5, we can prove the following lemma:

Lemma 7. *For every $v \in \mathsf{Reach}(T \cup T_i)$, $\mathsf{b}_i(v) = \max\{\mathsf{b}_i(f, v) \mid f \in \mathcal{S}_{\mathsf{Reach}}(T \cup T_i, v)\}$.*

For every $u \in \mathsf{Reach}(T \cup T_i)$, let

$$S_i(u) = \{v \geq_{\mathcal{T}} u \mid \forall w \in \text{Path}[u, v] : \mathsf{b}_i(w) \geq \mathsf{lev}(u)\}.$$

We can prove the next lemma similarly as proving Lemma 6 with every appearance of Lemma 5 replaced by Lemma 7.

Lemma 8. *For any node $v \in \mathsf{Reach}(T_i \cup T)$, v belongs to an i-snare if and only if $v \in S_i(u)$ for some $u \leq v$ such that $\mathsf{b}_i(u) \geq \mathsf{lev}(u)$.*

Hence, we obtain the following equality for every $i \in \{0, \dots, r\}$.

$$S_i = \bigcup \Big\{ S_i(u) \mid u \in \mathsf{Reach}(T \cup T_i), \mathsf{b}_i(u) \geq \mathsf{lev}(u) \Big\}. \tag{1}$$

4.3 An algorithm for solving Büchi games on trees with back-edges

Recall that for a tree $\mathcal{T}$, the *external path length* ℓ is $\sum\{\mathsf{lev}(v) \mid v \text{ is a leaf in } \mathcal{T}\}$. For any game $\mathcal{G}$ played on trees with back-edges (not necessarily reduced), by the *height* and *external path length* of $\mathcal{G}$ we respectively mean the height and external path length of the underlying tree of $\mathcal{G}$.

Theorem 3. *There exists an algorithm that solves any Büchi game $\mathcal{G}$ played on trees with back-edges in time $O(\min\{r \cdot m, \ell + m\})$ where r is the snare rank, m is the number of edges and ℓ is the external path length of $\mathcal{G}$.*

Proof. By Lemma 2, we first compute in time $O(m)$ the reduced game $\text{Rd}(\mathcal{G})$. The rest of the algorithm works on $\text{Rd}(\mathcal{G})$, which we simply write as $\mathcal{G}$. For $i \geq 0$, assume S_{i-1} has been computed. By Lemma 8 and (1), Algorithm 2 computes the set S_i. By Lemma 3

Algorithm 2 $\mathsf{FindSnare}[i](\mathcal{G})$. (Outline)

Compute the set $T_i = \{w \mid E^{\mathsf{B}}(w) \in S_{i-1}\}$.
Compute $\mathsf{Reach}(T \cup T_i)$.
For all $u \in \mathsf{Reach}(T_i \cup T)$ do:
  Compute $\mathsf{b}_i(u)$
  If $\mathsf{b}_i(u) \geq \mathsf{lev}(u)$, compute $S_i(u)$ and add $S_i(u)$ to S_i.

and Lemma 4, we obtain that $W_0 = S_r$. Hence, to compute the entire winning region W_0, it suffices to run $\mathsf{FindSnare}[0](\mathcal{G})$, $\mathsf{Findsnare}[1](\mathcal{G})$, …, in order. The algorithm terminates after running $\mathsf{FindSnare}[r{+}1](\mathcal{G})$ where $S_r = S_{r+1}$ (and thus r is the snare rank).

In $\mathsf{FindSnare}[i](\mathcal{G})$, $i \in \{0, \dots, r+1\}$, we compute $\mathsf{b}_i(u)$ for all $u \in \mathsf{Reach}(T_i \cup T)$ in order: we only compute $\mathsf{b}_i(u)$ when $\mathsf{b}_i(v)$ for all $v \in E^{\mathsf{T}}(u) \cap \mathsf{Reach}(T_i \cup T)$ have been computed. When $\mathsf{b}_i(u) \geq \mathsf{lev}(u)$, we apply a depth-first search on the subtree rooted at u to compute the set $S_i(u)$. After $S_i(u)$ has been computed, we contract all the nodes in $S_i(u)$ into a meta-node $M_{S_i(u)}$ and redirect edges as follows: any edge (u, v) where $u \in S_i(u)$ and $v \notin S_i(u)$ is substituted by an edge $(M_{S_i(u)}, v)$ and conversely any edge (v, u) where $v \notin S_i(u)$ and $u \in S_i(u)$ is substituted by $(v, M_{S_i(u)})$. Hence, each edge in $\mathcal{G}$ is visited a fixed number of times and the $\mathsf{FindSnare}[i](\mathcal{G})$ algorithm takes time $O(m)$.

To further reduce the running time of the algorithm, we maintain a variable $\mathsf{b}(u)$ for every node u throughout the entire algorithm. When $\mathsf{FindSnare}[i](\mathcal{G})$ is executed, $\mathsf{b}(u)$ will store the value of $\mathsf{b}_i(u)$. During the first iteration (when $\mathsf{FindSnare}[0]$ is performed), $\mathsf{b}(u) = \mathsf{b}_0(u)$ for all $u \in \mathsf{Reach}(T)$ and undefined for all other nodes. In the subsequent iterations, we do the following to compute $\mathsf{b}_i(u)$ for $i > 0$ and $u \in \mathsf{Reach}(T \cup T_i)$:

1. If u is a leaf in T_i, set $\mathsf{b}(u) = h$. If u is a leaf in T, the value of $b(u)$ remains unchanged.
2. Then "propagate" the value of $\mathsf{b}(u)$ to ancestors of u as follows: let v be the parent of u. If $v \in V_1$ and $b(v) > b(u)$, then set $b(v) = b(u)$. If $v \in V_0$ and $b(v) < b(u)$, then set $b(v) = b(u)$. This process continues until we reach a node $w <_{\mathcal{T}} u$ such that $b(w)$ does not need to be updated or w is the root.

Hence, at any iteration of the algorithm, we only change the value of $\mathsf{b}(v)$ when $\mathsf{b}(u)$ is changed for some leaf $u \geq_{\mathcal{T}} v$. Also, for any leaf u, if the value of $\mathsf{b}(u)$ is set to h, it is never changed again. Therefore, the number of times we visit a node $v \in V$ is at most the number of leaves in the subtree rooted at v. This means that the algorithm runs in time $O(\ell + m)$ (since the external path length of $\mathrm{Rd}(\mathcal{G})$ is at most $\ell + m$). By the arguments above, we conclude that the algorithm runs in time $O(\min\{r \cdot m, \ell + m\})$. $\square$

5 Solving parity games played on trees with back-edges

We now apply Theorem 3 to obtain an algorithm for solving parity games on trees with back-edges. Recall the definition of parity games from Section 2: In a *parity game* $\mathcal{G}$, each node $u \in V$ is associated with a priority $\rho(u) \in \mathbb{N}$ and Player 0 wins $\pi = v_0 v_1 \cdots$ if and only if $\min\{\rho(v) \mid v \in \mathrm{Inf}(\pi)\}$ is even. Also recall that we may assume that $E(u) \neq \emptyset$ for any $u \in V$. The following lemma reduces the problem of solving parity games played on trees with back-edges to solving reduced Büchi games.

Lemma 9. *Given a parity game* $\mathcal{G} = (V_0, V_1, E_1^{\mathsf{T}}, E_1^{\mathsf{B}}, \rho)$ *played on trees with back-edges, there is a reduced Büchi game* $\mathcal{H} = (U_0, U_1, E_2^{\mathsf{T}}, E_2^{\mathsf{B}}, T)$ *such that*

- $V \subseteq U$ *and* $|U| \leq |V| + |E_1^{\mathsf{B}}|$.
- *A node* $u \in V$ *is winning for Player 0 in* $\mathcal{G}$ *if and only if* u *is winning for Player 0 in* $\mathcal{H}$.
- $\mathcal{H}$ *is constructed in time* $O(\ell + m)$ *where* ℓ *is the external path length of* $\mathcal{G}$.

Proof. To define the sets $U_0, U_1, E_2^{\mathsf{T}}$ and E_2^{B}, we use the same construction as in the proof of Lemma 2. The target set T in the game $\mathcal{H}$ is defined as

$$T = \{\alpha(u,v) \mid (u,v) \in E_1^{\mathsf{B}}, \min\{\rho(x) \mid x \in \mathrm{Path}[v,u]\} \text{ is even}\}.$$

We prove the following claim.

Claim. A node $u \in V$ is winning for Player 0 in $\mathcal{G}$ if and only if u is winning for Player 0 in $\mathcal{H}$.

Fix $u \in V$. Suppose u is a winning position of Player 0 in $\mathcal{G}$. Let f be the winning strategy for Player 0 at u. By Theorem 1 we know that f is a memoryless strategy. Define the strategy g in the same way as in the proof of Lemma 2. Any play π starting from u and consistent with g in $\mathcal{H}$ defines a play π' starting from u and consistent with f in $\mathcal{G}$ such that $\mathrm{Inf}(\pi') = \mathrm{Inf}(\pi) \cap V$. Since f is a winning strategy for Player 0, $\min\{\rho(x) \mid x \in \mathrm{Inf}(\pi')\}$ is even. Let $e = \min\{\rho(x) \mid x \in \mathrm{Inf}(\pi)\}$. There must be a back edge $(x,y) \in E_1^{\mathsf{B}}$ that is visited infinitely often by π' and $\min\{\rho(z) \mid z \in \mathrm{Path}[y,x]\} = e$. By definition, the node $\alpha(x,y) \in T$ and appears in π infinitely often. Hence π is winning for Player 0.

Conversely, suppose u is winning position of Player 0 in $\mathcal{H}$. Then u belongs to a snare by Theorem 2. Let i be the snare rank of u and let S be the i-snare containing u. Let f be a snare strategy for Player 0 in $S \cup S_{i-1}$ (recall that S_{i-1} denotes all nodes in $\mathcal{H}$ that are in an $(i-1)$-snare). Define the strategy $g : V_0 \to V$ in the same way as in the proof of Lemma 2. Let π be a play consistent with g starting from u in $\mathcal{G}$. Our goal is to prove that π is a winning play for Player 0 in $\mathcal{G}$. Suppose for the sake of contradiction that π is winning for Player 1.

Note that π corresponds to a play π' consistent with f such that $\mathrm{Inf}(\pi) = \mathrm{Inf}(\pi') \cap V$. By definition of an i-snare, each leaf in the snare S is either a target or has a back edge that goes to S_{i-1}. Assume π' never visits S_{i-1}. In this case, all leaves visited by π' are targets. Let R be the set of leaves visited by π', then by Lemma 1, $\mathrm{Inf}(\pi') = \bigcup\{\mathrm{Path}[x, \alpha(y,x)] \mid \alpha(y,x) \in R\}$. Therefore

$$\mathrm{Inf}(\pi) = \mathrm{Inf}(\pi') \cap V = \bigcup\{\mathrm{Path}[x,y] \mid \alpha(y,x) \in R\}.$$

Since $R \subseteq T$, for all $\alpha(y,x) \in R$, $\min\{\rho(z) \mid z \in \mathrm{Path}[x,y]\}$ is even. Hence $\min\{\rho(z) \mid z \in \mathrm{Inf}(\pi)\}$ is also even and π is winning for Player 0. This is in contradiction with the assumption that π is winning for Player 1.

Hence π' (and π) must visit a node $u_1 <_{\mathcal{H}} u$ that belongs to an $(i-1)$-snare. Now apply the same argument on u_1. Continuing this process, we obtain a sequence of nodes $u = u_0 >_{\mathcal{H}} u_1 >_{\mathcal{H}} u_2 >_{\mathcal{H}} \cdots$ such that for all $j \in \mathbb{N}$, u_{j+1} has snare rank strictly smaller than the snare rank of u_j. Contradiction.

Hence the claim is proved.

We use the depth-first search (DFS) algorithm on the underlying tree $\mathcal{T}$ of $\mathcal{G}$. Consider a path P in $\mathcal{T}$ from the root to a leaf. We represent the length of P by $|P|$. We use the algorithm of (17) to preprocess P in time $O(|P|)$ such that for any $u, v \in P$ ($u <_{\mathcal{T}} v$), we may find the value of $\min\{\rho(x) \mid x \in \mathrm{Path}[u,v]\}$ in constant time. Then it is clear that preprocessing every such path in $\mathcal{T}$ takes $O(\ell)$ time (where ℓ is the external path length of $\mathcal{G}$) and subsequently finding $\min\{\rho(x) \mid x \in \mathrm{Path}[u,v]\}$ for any nodes $u <_{\mathcal{T}} v$ takes constant time.

Hence for every back edge $(u,v) \in E_1^{\mathsf{B}}$, $\min\{\rho(x) \mid x \in \mathrm{Path}[v,u]\}$ maybe found in constant time after the above preprocessing has been completed. Therefore the algorithm constructs the reduced Büchi game $\mathcal{H}$ as follows:

1. Preprocess every path P of $\mathcal{T}$ from the root to a leaf using the algorithm of (17).
2. For each back edge $(u,v) \in E_1^{\mathsf{B}}$, create a new leaf $\alpha(u,v)$ by subdividing (u,v). Set $\alpha(u,v)$ as a target if and only if $\min\{\rho(x) \mid x \in \mathrm{Path}[v,u]\}$ is even (note that this takes constant time now).

The above procedure takes time $O(\ell + m)$. $\square$

By Lemma 9 and Theorem 3 we obtain the following theorem.

Theorem 4. *Any parity game $\mathcal{G}$ played on trees with back-edges can be solved in time $O(\ell + m)$ where ℓ is the external path length of $\mathcal{G}$ and m is the number of edges in $\mathcal{G}$.*

6 Experimental results

In order to compare the performance of our algorithm with that of the classical algorithm, we implemented both the algorithms using the *Sage* mathematics software system (18). All the experiments were performed on an *Intel Core 2 Duo processor (*2.4 *GHz)* with a *L2 cache* of 4 MB and *RAM* of 3 GB.

The experiments can be broadly divided into two categories:

1. Average case running time comparison: We systematically enumerate all games on trees with back-edges with order $n \in \mathbb{N}$ nodes. For each game, we compare the running times of the classical algorithm and our algorithm.

2. Running time comparison with random sampling: We considered three classes of rooted trees: (a) rooted trees with unbounded outdegree (denoted by **RANUD**) , (b) rooted binary trees (denoted by **RANBT**) and (c) rooted trees where all internal nodes have only a single child node (denoted by **RANDL**) We consider the class **RANDL** since it is the simplest class of rooted trees.

   Whenever we refer to a random Büchi game $\mathcal{G}$ from one of these classes of trees, we mean that the underlying tree of G is a randomly generated tree from that class. Since for a given $n \in \mathbb{N}$, there is a unique tree of order n from **RANDL**, we describe how we generate random samples of order n from the classes **RANUD** and **RANBT**:

   - **RANUD** : We first construct a random free tree T_{free} of order n by generating a sequence of $(n-2)$ random integers chosen independently and uniformly from $\{0, 1, \ldots, n-1\}$ and then applying a reverse Prüfer transformation to this sequence (15). We then randomly select a root from the nodes of T_{free}, hence obtaining rooted tree T.
   - **RANBT**: We construct a random rooted binary tree by the algorithm of (16).

   Given a random rooted tree T from **RANUD** or **RANBT**, we generate a random reduced Büchi game $\mathcal{G}$ from T as follows:

   - For each leaf v of T, we make a random choice of an ancestor u of v and declare a back-edge from v to u. In this manner we obtain a random tree with back-edges T_b.
   - From the nodes of T_b, we randomly select nodes of Player 0 and target nodes to obtain a random Büchi game $\mathcal{G}$ on trees with back-edges. Note that $\mathcal{G}$ is reduced.

   Given a rooted tree T from **RANDL**, for every node v of T we randomly choose an ancestor u of v and add a back-edge from v to u. Then we randomly choose nodes of Player 0 and target nodes as before to obtain a random game $\mathcal{G}$. We then use the procedure described in Lemma 2 to convert $\mathcal{G}$ to a reduced game.

   In our experiment, we generated random Büchi games from the three classes described above and then compared the running times of the classical algorithm and our algorithm.

Results: *Average case running time comparison:* Figure 3 (bottom right) shows the average running time of the classical algorithm and our algorithm for games of size $n \in \{5, 6, \ldots, 13\}$. It is clear that our algorithm performs better than the classical algorithm in all cases. Moreover as n increases, the difference in average case running time between the two algorithms increases. This is particularly evident for $n = 13$, where our algorithm performs an order of magnitude better than the classical algorithm.

The picture becomes even clearer when we analyze the results of the experiments with random sampling. In order to enable a more convenient comparison between the two algorithms we have scaled down the running times of the classical algorithm by a factor of 10^2 for all experiments with random sampling. The sizes of the random games are in the range $n \in \{100, \ldots, 10000\}$.

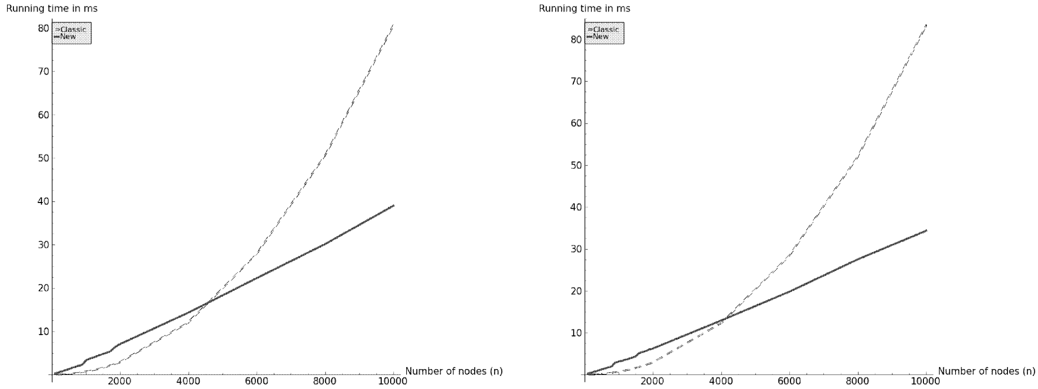
Random games from **RANUD** : Figure 3 (top left) shows the graph comparing the running time of the classical algorithm versus our algorithm for random Büchi games from the class **RANUD**. The sample sizes are as follows: 10^6 random games for $n \in \{100, \ldots, 2000\}$, 10^5 random games for $n \in \{4000, \ldots, 6000\}$, $5 \cdot 10^4$ games for $n = 8000$ and $4 \cdot 10^4$ games for $n = 10000$.

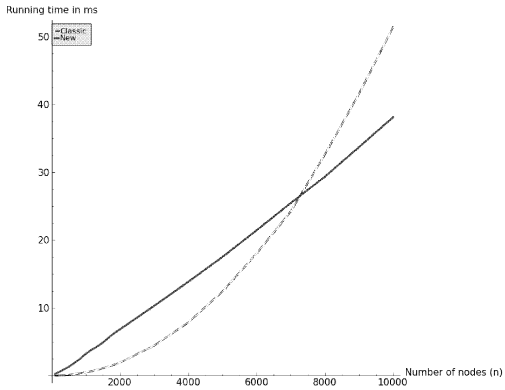
As the graph shows, the classical algorithm exhibits quadratic growth in running time whereas our algorithm has linear growth in running time (with respect to n). This is better than the worst case bound of $O(min\{r \cdot m, l\})$ mentioned in Theorem 3.

Random games from **RANBT**: Figure 3 (top right) shows the graph comparing the running time of the classical algorithm versus our algorithm for random Büchi games from the class **RANBT**. The sample sizes are as follows: 10^5 games for $n \in \{100, \ldots, 2000\}$ and 10^4 games for $n \in \{4000, 10000\}$. Again the classical algorithm shows a quadratic growth in running time as compared to our algorithm which has a linear growth in running time (w.r.t. n) which is better than the worst case bound of $O(min\{r \cdot m, \ell\})$.

Random games from **RANDL**: Figure 3 (bottom left) shows the graph comparing the running time of the classical algorithm versus our algorithm for random Büchi games from the class **RANDL**. The sample sizes are identical to the **RANBT** case. Here also, our algorithm shows a linear growth as compared to the quadratic growth shown by the classical algorithm.

Hence the experiments demonstrate that our algorithm outperforms the classical algorithm in all the three classes. In particular, for $n > 4000$, our algorithm performs two orders of magnitude better than the classical algorithm for all three classes of Büchi games. The experiments clearly show that, in practice, not only our algorithm is much more efficient asymptotically but it also performs better for games with small number of nodes on the set of trees with back-edges.





No. of nodes	Average running time in ms	
	Classical	New
5	0.032809	0.028340
6	0.037290	0.030867
7	0.041399	0.033830
8	0.045444	0.036673
9	0.050211	0.039603
10	0.054622	0.042404
11	0.059377	0.045168
12	0.097281	0.057533
13	0.111520	0.061552

Figure 3: The top left graph shows the comparison of algorithms for **RANUD**, the top right graph for **RANBT** and the bottom left for **RANDL** (the dashed lines represent the classical algorithm). The bottom right table compares the average running times of the two algorithms. Note that the running time of the classical algorithm has been scaled down by 10^2 in the graphs.

References

[1] Berwanger, D., Grädel, E.: Entanglement - a measure for the complexity of directed graphs with applications to logic and games. In: Proceedings of LPAR'04, pp. 209-223, 2004.

[2] Berwanger, D., Grädel, E., Lenzi, G.: The variable hierarchy of the μ calculus is strict, In: Theory of Computing Systems 40, no. 4, 2007, 437-466.

[3] Chatterjee, K., Henzinger, T., Jurdzinski, M.: Mean-Payoff Parity Games. In: Proc. of LICS'05, pp.178–187, 2005.

[4] Chatterjee, K., Henzinger, T., Piterman, N.: Algorithms for Büchi games. In: Proc. of the 3rd Workshop of Games in Design and Verification (GDV'06), 2006.

[5] Chatterjee, K., Jurdziński, M., Henzinger, T.: Simple stochastic parity games. In: Proc. of CSL'03. LNCS 2803:100-113. Springer, 2003.

[6] Clarke, E., Lu, Y., Veith, H., Jha, S.: Tree-Like counterexamples in model checking. In: Proc. of LICS'02, pp.19-29, IEEE Computer Society, 2002.

[7] Grädel, E., Thomas, W., Wilke, T. (Eds.): Automata, Logics, and Infinite Games: A Guide to Current Research. LNCS 2500. Springer, 2002.

[8] Immerman, N.: Number of quantifiers is better than number of tape cells. In: Journal of Computer and System Sciences, 22, 1981, 384–406.

[9] Mang, F.: Games in Open Systems Verification and Synthesis, PhD thesis, University of California at Berkeley, 2002.

[10] Martin, D.: Borel determinacy, Ann. Math. 102, No. 2(Sep., 1975), pp. 363-371.

[11] Murawski, A.: Reachability games and game semantics: Comparing nondeterministic programs. In: Proc. of LICS'08, pp. 353–363, 2008.

[12] Nerode, A., Yakhnis, A., Yakhnis, V.: Concurrent programs as strategies in games, in: Logic from Computer Science (Y. Moschovakis, ed.), Springer, 1992.

[13] Obdržálek, J.: Algorithmic Analysis of Parity Games, PhD thesis, Univ. of Edinburgh, 2006.

[14] Thomas, W.: Infinite games and verification. In: Proc. of the International Conference on Computer Aided Verification (CAV'02), LNCS 2404:58-64, 2002.

[15] Cho, M., Kim, D., Seo S., and Shin, H.: Colored Prüfer Codes for $k-$Edge Colored Trees, Electron. J. Combin. **11** no. 1, #N10, 2004.

[16] Arnold, D.B., and Sleep, M.R.: Uniform random generation of balanced parenthesis strings, ACM Trans. Program. Lang. Syst. **2**. 1980, 122-128.

[17] Fischer, J., and Heun, V.: A New Succinct Representation of RMQ-Information and Improvements in the Enhanced Suffix Array, LNCS 4614:459-470, 2007.

[18] Sage open source mathematical software, `http://www.sagemath.org/`.