A Formal Proof of the Termination of Zielonka's Algorithm for Solving Parity Games

Remco Abraham
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
r.abraham@student.utwente.nl

ABSTRACT

It is currently unknown whether parity games can be solved in polynomial time. Still, it is known that many problems related to model-checking and synthesis can be reduced to parity games. Algorithms that solve parity games are therefore of great value.
One of the earliest algorithms invented for this purpose is the algorithm proposed by Zielonka. Various pen and paper proofs of its correctness have been provided, but although parity games play an important role in many machine-generated and machine verified proofs, Zielonka's algorithm has no machine-checked proof itself. As a starting point for the formal proof of Zielonka's algorithm, this research will provide a formal proof for the termination of the algorithm using the formal theorem prover Isabelle/HOL.

Keywords

Zielonka, Parity Games, Formal Proof, Termination

1. INTRODUCTION

Parity games have shown their importance in several fields like model-checking [3] and controller synthesis [1] since many problems in those fields can be reduced to parity games. Many algorithms have been created that can solve parity games, although none of them can solve parity games in polynomial time [11]. One of the earliest algorithms for solving parity games was invented by Zielonka [13]. Still, Zielonka's algorithm is one of the major algorithms for solving parity games since it is relatively simple yet turns out to be quick in practise [5]. It has also been optimized multiple times [12, 7].

Although correctness and termination have been proven in the past [13, 4], it was never done using a formal approach. In this text, we define an *informal proof* as a proof written in a natural language, and a *formal proof* as a proof written in an artificial language that can be mechanically verified for correctness. The main advantages of a formal proof over an informal proof are that a formal proof is much more reliable and that it is easier to verify by a peer-reviewer [6]. Moreover, a formal proof for Zielonka's algorithm is especially useful because of its possible usage in model-checking. Any parity game solving algorithm is essentially a model-checker and Zielonka's algorithm turns out to be the best parity game solving algorithm in practice [5]. Providing a formal proof for Zielonka's algorithm using a formal theorem prover would, therefore, be a great step towards model-checking in that theorem prover. It could eventually enable formal model-checking of any algorithm and is, therefore, a great addition to the field of software verification.

31st Twente Student Conference on IT July 5th, 2019, Enschede, The Netherlands.

Contributions. In this paper, a fundamental part of the formal proof for Zielonka's algorithm is given. Namely, we describe a formal proof for the termination of Zielonka's algorithm. The full formal proof is given in Isabelle/HOL [8] and available on `https://doi.org/10.5281/zenodo.3252437`. The proof is constructed for an optimized version of Zielonka's algorithm as proposed by Lui et al. [7]. Afterwards, the proof is extended to also include an optimization proposed by Verver [12] and implemented by van Dijk [11]. In conclusion, the research contributes the following:

- A formal proof that Zielonka's algorithm as proposed by Lui et al. [7] terminates.
- A formal proof that Zielonka's algorithm as implemented by van Dijk [11] terminates.

2. BACKGROUND

In this section, the necessary background knowledge for this research is given. The first section will explain the definition of a parity game and the second section will provide the full algorithm and explain it on a conceptual level.

2.1 Parity Games

For the definition of a parity game, we adhere to the mathematical notation used by Obdrzálek [9], but we use a slightly different definition to also consider finite plays. A parity game $\mathcal{G} = (V, E, V_0, \lambda)$ consists of a directed graph $G = (V, E)$ where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. Furthermore, it includes a so-called parity function $\lambda : V \mapsto \mathbb{N}$, which maps the vertices of the graph to some priority number.

A parity game is always played with two players, frequently referred to as Even and Odd. We will use P_0 and P_1. Every node is either "owned" by P_0 or by P_1. Let V_0 (V_1) denote the set of nodes owned by P_0 (P_1). Then $V = V_0 \cup V_1$. For the definition of a Parity Game, knowing V_0 is sufficient since $V_1 = V \setminus V_0$.

The game starts at some initial vertex v. If $v \in V_0$ then P_0 starts, otherwise (so when $v \in V_1$) P_1 starts. A move is made by the player by moving a token along an edge

$(v, w) \in E$ from v to some vertex w. After a move is made, we check again whether the current vertex (w) is owned by P_0 or P_1 to determine the player that makes the next move. The game will end when there is no outgoing edge from the current vertex (a finite play) or the game will go on forever in the case no such vertex is ever reached (an infinite play).

In the case of a finite play, the winner is determined by the parity of the priority given by the parity function λ of the final vertex v_f. The winner of the game then is $P_{\lambda(v_f) \mod 2}$.

More interesting is the case of an infinite play. In that case the token is moved along an infinite path $\pi = \pi_1\pi_2...$ where $\forall i > 0, (\pi_i, \pi_{i+1}) \in E$. Not all vertices that occur in the path need to occur in the path infinitely often. Let $Inf(\pi) = \{v \in V \mid v \text{ occurs infinitely often in } \pi\}$. Then the winner of the game is the player with the parity of the largest priority in $Inf(\pi)$, which can be written as $P_{\max\{\lambda(v)|v \in Inf(\pi)\} \mod 2}$.

Finding a solution to a parity game entails finding for every possible starting vertex, which player eventually wins the game, and optionally also finding the strategy (which edge to choose to pass the token along) that the players should adhere to for every vertex. The set of vertices $W_a \subseteq V$ is called a winning region for player a if player a wins for every starting vertex in W_a. The next section explains the algorithm proposed by Zielonka [13] for finding these winning regions.

2.2 Zielonka's algorithm

Zielonka's original algorithm [13] has frequently been optimized [7, 12]. Therefore, this research will use the optimized version of Zielonka's algorithm as implemented by van Dijk [11] to provide the final formal proof. In this section, if a is a player, we denote the other player by $\overline{a}$. Before providing the full algorithm, a few preliminaries will be discussed.

2.2.1 Attractor

An attractor for player a on the graph G of game $\mathcal{G}$ for the set of vertices U (which we call the attractor's base) is the set of vertices for which player a can force the token into one of the vertices of U. We denote this attractor by $Attr_a^{\mathcal{G}}(U)$. It can be computed iteratively by starting with U and adding the vertices in V_a that have an edge to some vertex in U and adding the vertices in $V_{\overline{a}}$ that only have edges to vertices in U. More formally:

$$U_0 = U$$
$$U_{i+1} = U_i \cup \{v \in V_a | \exists (v, w) \in E : w \in U_i\} \cup \{v \in V_{\bar{a}} | \forall (v, w) \in E : w \in U_i\}$$
$$Attr_a^{\mathcal{G}}(U) = \bigcup_{i=0}^{\infty} U_i$$

At some point, since the graph is finite, $Attr_a^G(U)$ converges, which gives us the attractor on $\mathcal{G}$ of the vertices U for player a.

2.2.2 Subgame

A subgame of a parity game $\mathcal{G}$, is a parity game with some of the vertices of the graph of $\mathcal{G}$ removed. A more formal definition is given below.

DEFINITION 1. *Let $\mathcal{G}$ be a Parity Game and A be a set of vertices, then $\mathcal{G} \setminus A$ is the subgame of $\mathcal{G}$ reduced by A, with $V_{\mathcal{G}\setminus A} = V_{\mathcal{G}} \setminus A$ and $E_{\mathcal{G}\setminus A} = E_{\mathcal{G}} \setminus (A \times A)$*

2.2.3 Description of the algorithm

This section describes the algorithm as implemented by van Dijk [11]. The algorithm can be found in Algorithm 1. The algorithm works by removing the currently winning region A from the game and then recursively computing the winning regions W_0 and W_1 for that subgame. It now needs to be determined whether A should be part of the winning region of a. To determine this, we compute the attractor of $\overline{a}$ of $W_{\overline{a}}$ in $\mathcal{G}$ as $W'_{\overline{a}}$. If it turns out that this attractor is equal to $W_{\overline{a}}$, then we know that $\overline{a}$ could not attract any vertices from A to its winning region $W_{\overline{a}}$. Thus, a can force the token from A into W_a instead. Therefore A is added to the winning region of a.

In the case that $\overline{a}$ does manage to attract vertices from A to its winning region, then $\overline{a}$ can force the token from $W'_{\overline{a}}$ into its winning region on the subgame without $W'_{\overline{a}}$, because it is not possible for a to escape to A; A is already an attractor for a. Therefore $W'_{\overline{a}}$ can be added to the winning region of a.

For more details about the algorithm, see Zielonka's paper [13], where the algorithm is proposed as a constructive proof to prove the existence of solutions of parity games. Further elaboration on the optimizations can be found in Verver's thesis [12] and the paper of Lui et al.[7].

Algorithm 1 Zielonka's algorithm with optimizations

```
1: function ZIELONKA(G)
2:     if G = ∅ then
3:         return ∅, ∅
4:     end if
5:     a ← max_p(G) mod 2
6:     A ← ATTR(G, a)
7:     W_0, W_1 ← ZIELONKA(G \ A)
8:     W'_ā ← Attr^G_ā(W_ā)
9:     if W'_ā = W_ā then
10:        W_a ← W_a ∪ A
11:    else
12:        W_0, W_1 ← ZIELONKA(G \ W'_ā)
13:        W_ā ← W_ā ∪ W'_ā
14:    end if
15:    return W_0, W_1
16: end function
17: function ATTR(G, a)
18:    A ← ∅
19:    while max_p(G \ A) mod 2 = a ∧ A ≠ V_G do
20:        A ← A ∪ Attr_a^{G\A}(p_set(max_p(G \ A), G \ A))
21:    end while
22:    return A
23: end function
```

2.2.4 ATTR function

The $ATTR$ function used in Algorithm 1 implements the optimization found by Verver [12]. It makes use of the two helper functions *max_p* and *p_set* defined as follows:

$$max\_p(\mathcal{G}) = \max\{\lambda(v) | v \in V_{\mathcal{G}}\}$$

$$p\_set(p, \mathcal{G}) = \{v | v \in V_{\mathcal{G}}, \lambda(v) = p\}$$

$max\_p(\mathcal{G})$ computes the maximal priority in the given game $\mathcal{G}$ and $p\_set(p, \mathcal{G})$ gives the set of vertices of priority p in $\mathcal{G}$.

Note that the definition of $ATTR$ in Algorithm 1 does not entirely match the algorithm in van Dijk's paper. Line 19 of Algorithm 1 differs from van Dijk's paper in that the

condition $A \neq V_{\mathcal{G}}$ has been added. During construction of the formal proof of the termination of the $ATTR$ function, it was discovered that the original definition in van Dijk's paper did not terminate. The correction made here is based on the description of the $ATTR$ function by Verver [12], which does not contain this error.

For the initial proof without Verver's optimization, line 6 of Algorithm 1 will be replaced by:

$$A \leftarrow Attr_a^{\mathcal{G}}(p\_set(max\_p(\mathcal{G}), \mathcal{G} \setminus A))$$

We will refer to this simplified version of the algorithm as *ZIELONKA_SIMP*.

3. METHODOLOGY

The formal theorem prover Isabelle/HOL [8] is used to provide a formal proof of the termination of Zielonka's algorithm. While there are many other formal theorem provers, Isabelle/HOL was the most natural choice because Dittman has already formalized parity games in Isabelle/HOL. This formalization of parity games can be found in the Archive of Formal Proofs [2]. We build upon this formalization in order to give a formal proof of the termination of Zielonka's algorithm. A first attempt at formalizing Zielonka's algorithm with this approach has already been made by Thijssen [10]. We extend upon his formalization.

Informal proofs of the termination of Zielonka's algorithm [12, 9, 4] inspire the formal proof. The informal proofs are studied in detail to identify small lemmas that can be proven in Isabelle/HOL. Eventually, these lemmas are combined to form the proof of the termination of Zielonka's non-optimized algorithm. Afterwards, the proof is extended to include the optimization by Verver as discussed in subsection 2.2.

4. RESULTS

This section describes the formal proof constructed during this research. First, an informal proof of the termination of the algorithm will be presented. Then an outline of the formal proof is presented. Finally, we provide an in-depth explanation of the formal proof.

4.1 Informal Proof

Although, it is natural to prove termination of the algorithm using induction on the number of vertices of the game's graph (see Friedmann [4] for an example), we prove termination by induction on the recursive calls of the function instead. By constructing the proof using the number of vertices in the recursive calls, we map the arguments of the recursive calls to the natural numbers. Since we know that if $\forall i\ x_i \in \mathbb{N}$, there is no infinite sequence $x_0, x_1, x_2 ...$ such that $x_n < x_{n+1}$, and since the algorithm's non-recursive steps trivially terminate, proving that the number of vertices is strictly decreasing in each recursive call is sufficient to prove termination of the algorithm. Termination proofs using this argument have great support in Isabelle and it is therefore only natural to also provide the informal proof using the same technique.

First, the proof for the termination of *ZIELONKA_SIMP* is provided. Then the termination proof of *ZIELONKA* is derived from the termination proof of *ZIELONKA_SIMP*.

THEOREM 1. *Let $\mathcal{G}$ be a Parity Game, then ZIELONKA_SIMP($\mathcal{G}$) terminates*

PROOF. We prove termination by showing that in each recursive call, the number of vertices in the game's graph is strictly decreasing.

Let $a = max\_p(\mathcal{G}) \bmod 2$ and let $A = Attr_a^{\mathcal{G}}(p\_set(max\_p(\mathcal{G}), \mathcal{G} \setminus A))$. Obviously $A \subseteq V_{\mathcal{G}}$. We also have that $A \neq \emptyset$ since there is always a vertex with maximal priority. We then have $|V_{\mathcal{G}\setminus A}| < |V_{\mathcal{G}}|$, the number of vertices in the subgame $\mathcal{G}$ reduced by A is strictly smaller than the number of vertices in $\mathcal{G}$.

Now let $W_0, W_1 = ZIELONKA\_SIMP(\mathcal{G} \setminus A)$ and $W_{\overline{a}'} = Attr_{\overline{a}}^{\mathcal{G}}(W_{\overline{a}})$. We make a case distinction on the condition of line 9 of Algorithm 1. Suppose $W_{\overline{a}}^{'} = W_{\overline{a}}$, then the algorithm obviously terminates so we focus on the case where $W_{\overline{a}}^{'} \neq W_{\overline{a}}$. We again need to show that the number of vertices in the recursive call is strictly decreasing. We do that by showing that $W_{\overline{a}}^{'} \neq \emptyset$ and that there is at least one vertex that is both in $V_{\mathcal{G}}$ and in $W_{\overline{a}}^{'}$. We prove both conditions by contradiction.

Suppose $W_{\overline{a}} = \emptyset$, then obviously $W_{\overline{a}}^{'} = \emptyset$ but then $W_{\overline{a}} = W_{\overline{a}}^{'}$ which contradicts our previous supposition.

Next suppose $\nexists v(v \in V_{\mathcal{G}} \wedge v \in W_{\overline{a}}^{'})$. Then obviously the attractor of $W_{\overline{a}}^{'}$ for $\overline{a}$ on $\mathcal{G}$ is just $W_{\overline{a}}^{'}$, thus again $W_{\overline{a}} = W_{\overline{a}}^{'}$, contradicting our supposition.

Now we have, similar to the first recursive call, $|V_{\mathcal{G}\setminus W_{\overline{a}}^{'}}| < |V_{\mathcal{G}}|$. Since we have now shown that both recursive calls strictly decrease in the number of vertices and there is no strictly decreasing infinite sequence of natural numbers, we know this cannot continue indefinitely, thus we have shown termination of the algorithm. □

Next, we prove that $ATTR$ terminates, that $ATTR(\mathcal{G}, a) \subseteq V_{\mathcal{G}}$ and that $ATTR(\mathcal{G}, a) \neq \emptyset$ to show that *ZIELONKA* terminates.

LEMMA 1. *Let $\mathcal{G}$ be a Parity Game and let a be a player, then $ATTR(\mathcal{G}, a)$ terminates*

PROOF. In each iteration of the while loop, there must be at least one vertex v of maximal priority in $G \setminus A$. Only if $G \setminus A = \emptyset$ this is not true, but that would violate the while condition. Since v is not in A, at the end of the iteration, v will be added to A. Thus it is guaranteed that A grows in each iteration. Combined with the fact that $ATTR(\mathcal{G}, a)$ terminates if $A = V_{\mathcal{G}}$, it can be concluded that $ATTR(\mathcal{G}, a)$ terminates. □

THEOREM 2. *Let $\mathcal{G}$ be a Parity Game, then ZIELONKA($\mathcal{G}$) terminates.*

PROOF. Since $ATTR(\mathcal{G}, a)$ is just a repeated application of *Attr*, it is trivial that $ATTR(\mathcal{G}, a) \subseteq V_{\mathcal{G}}$. It is also trivial to show that $ATTR(\mathcal{G}, a) \neq \emptyset$: Since $a = max\_p(\mathcal{G}) \bmod 2$, $ATTR(\mathcal{G}, a)$ will do at least one iteration. In the proof of Lemma 1, it was shown that each iteration increases the size of A. Thus $ATTR(\mathcal{G}, a) \neq \emptyset$. Using these facts in combination with Lemma 1, we again have $|V_{\mathcal{G}\setminus A}| < |V_{\mathcal{G}}|$. The rest of the proof is the same as for Theorem 1. □

4.2 Formal Proof

This section describes the formal proof of Zielonka's algorithm. First, the global outline of the proof is explained. Subsequent sections then provide a more in depth explanation of the formal proof. The formal proof itself can be found on `https://doi.org/10.5281/zenodo.3252437`.

4.2.1 Global outline

Since $ZIELONKA\_SIMP$ and $ZIELONKA$ are identical except for line 6, the formal proof is not given explicitly for either of these algorithms. Instead, a formal proof

is given which uses a generalized version of line 6. In fact, anything can be assigned to A in line 6 as long as it meets the following requirements:

1. $A \subseteq V_{\mathcal{G}}$
2. $A \neq \emptyset$

Note that these requirements do not guarantee the correctness of the algorithm, they only guarantee termination. If the proof is to be extended to a correctness proof, additional requirements are most certainly necessary.

Additionally, since it is only possible to prove termination under the condition that the input is valid (i.e. a parity game with a finite amount of vertices is provided), the algorithm used in the proof is modified slightly to check its own inputs.

Second, it is proven that both $Attr_a^{\mathcal{G}}(p\_set(max\_p(\mathcal{G}), \mathcal{G} \setminus A))$ and $ATTR(\mathcal{G}, a)$ meet the two requirements and can thus be used as instances of the generalized termination proof. Again, for $ATTR$, a modified algorithm is used to allow for an unconditional termination proof.

Finally, the original version of the algorithm is formalized and it is shown that the modified version discussed above is identical to the original version under the assumption that valid input is received. Since it is shown that both versions are identical under normal conditions, and the modified version is proven to terminate, the original version must also terminate (given valid inputs), concluding our proof.

4.2.2 Utilities

To simplify the termination proof, some definitions and abbreviations have been made. This section explains their purpose, as they occur frequently in the formal proof.

Basic parity game definitions.

As the basis for the proof, the formalization of parity games from the Archive of Formal Proofs [2] was used. The following definitions (and their appurtenant theorems) are used:

`ParityGame` defines a parity game.

`Player` defines a player in a parity game, which is either even or odd.

`ParityGame.subgame` gives a subgame of a parity game, given a subset of the game's vertices.

`ParityGame.attractor` gives an attractor of a parity game, given a subset of the game's vertices and a player.

The details of these definitions can be found in the Archive of Formal Proofs [2].

Minimal priority.

`min_prio` is a definition used to determine the minimal priority in a game. Although in literature, the parity of the maximal priority is normally used, the `ParityGame` definition actually uses minimal priority to refer to the maximal priority. The confusion arises because the literature has no general consensus on whether the maximal priority is 0 or whether the maximal priority is the highest priority number in the game. In either way, it should still be referred to as the maximal priority. Usage of the term "minimal priority" is therefore technically wrong. However, to avoid confusion to the reader of the proof, it was decided to use the same terminology as used in the formalization of Parity Games as a trade-off between coherence and terminological correctness.

Subgames.

In order to conveniently create a subgame of `G` without some vertices, the definition `removeVerts` was created. It uses the subgame definition of the Parity Game formalization to create a subgame. It is frequently used in the proof through its infix notation `-⇩G`.

Vertices of minimal priority.

The definition `min_prios` calculates the set of vertices that correspond to the minimal priority of the game. It reflects $p\_set(max\_p(\mathcal{G} \setminus A), \mathcal{G} \setminus A))$ of Algorithm 1.

Empty game.

An abbreviation `isEmpty` was made to indicate that a game is empty (meaning that there are no vertices). This improves the readability of the proof. This abbreviation is automatically substituted by its definition during parsing to avoid the need to manually unfold it.

Mapping priorities to players.

A priority trivially maps to a player through its parity. The definition `player` converts a priority to a `Player` through this mapping.

Player corresponding to minimal priority.

The abbreviation `minP` corresponds to line 5 of Algorithm 1. It determines the player corresponding to the minimal priority of the game.

Winning set.

`getW` selects the winning set corresponding to the given player out of the result of a call to Zielonka's algorithm. It matches W_a in Algorithm 1.

4.2.3 Definition

Zielonka's algorithm has been formalized as can be seen in Formalization 1. It starts with the definition of the `gen_attr` locale. Locales are classes with certain properties. They allow us to provide a generalized proof by specifying that a fixed variable has certain properties, without knowing the exact value beforehand. In this case, the `gen_attr` locale fixes an identically named variable `gen_attr` with the properties `non_empty` and `in_v`. These properties match the properties discussed in subsubsection 4.2.1. `non_empty` and `in_v` assume that `G` is a parity game and that the game has a finite amount of vertices. Additionally, `non_empty` assumes that `p` is the player with the minimal priority and that the parity game is not empty.

The formalization of Zielonka's algorithm is given within the context of the `gen_attr` locale. The formal definition matches Algorithm 1, with a few exceptions. First of all, notice how `gen_attr` is used on line 9 of Formalization 1. This is what generalizes the definition to allow its usage for both versions of Zielonka's algorithm. Furthermore, on line 8 we see that the formal definition has a stronger condition than Algorithm 1. This is the adaption to make the termination proof more simple. Later on, we prove that this adaption does not affect the result of the algorithm, nor does it affect termination (under normal circumstances).

Formalization 1 Formalization of the generalized Zielonka's algorithm with the strong condition

```
1: locale gen_attr =
2:   fixes gen_attr :: "'a ParityGame ⇒ Player ⇒ 'a set"
3:   assumes
4:    non_empty: "ParityGame G ⟹ p = minP G ⟹ ¬ isEmpty G ∧ finite V⇘G⇙ ⟹ gen_attr G
   p ≠ {}"
5:    and in_v: "ParityGame G ⟹ finite V⇘G⇙ ⟹ gen_attr G x ⊆ V⇘G⇙"
6: begin
7: function Zielonka :: "'a ParityGame ⇒ 'a set × 'a set" where
8:   "Zielonka G = (if (isEmpty G ∨ infinite V⇘G⇙ ∨ ¬ ParityGame G) then ({},{})
9:       else let p = minP G; A = gen_attr G p; Z = (Zielonka (G -⇩G A)) in
10:     (if (attr_stable G p** Z )
11:        then (add_to_result p A Z )
12:        else let att = ParityGame.attractor G (p**) (getW Z p**) in
13:         (add_to_result (p**) att (Zielonka (G -⇩G att)))))"
14:   by pat_completeness auto
```

Some definitions in Formalization 1 are still left unexplained. First of all, `attr_stable` is used to represent line 8 and 9 of Algorithm 1. It calculates the attractor of the winning set and checks whether the result is the same as the original winning set and is defined as follows:

```
1: definition attr_stable :: "'a ParityGame
   ⇒ Player ⇒ 'a set × 'a set ⇒ bool"
2: where "attr_stable G p Z ≡ (Pari-
   tyGame.attractor G p (getW Z p)) = getW
   Z p"
```

`add_to_result` is a definition used in Formalization 1 that aids in the implementation of line 10 and 13 of Algorithm 1. In the formalization, we need to explicitly state that the other winning set remains unchanged. This is done by taking the first and second element out of the pair manually (using `fst` and `snd` respectively), and then reconstructing the pair while only modifying one part of it. The definition is given below.

```
1: definition add_to_result :: "Player ⇒
   'a set ⇒ 'a set × 'a set ⇒ 'a set × 'a
   set" where
2:  "add_to_result p a Z = (if p = Even then
   (((fst Z)∪a),snd Z) else (fst Z,((snd
   Z)∪a)) )"
```

4.2.4 Generalized termination proof

Since the condition on line 8 is stronger than in Algorithm 1, it is possible to prove that Formalization 1 terminates for all possible inputs, independent on whether the inputs are actually valid. This simplifies the proof because it allows for an unconditional proof of termination, formally written as **`termination`** `Zielonka`. The rest of this section will explain this termination proof.

The termination proof starts by applying the relation tactic using a measure function of the number of vertices in `G`. This tactic implements a termination proof based on a well-founded relation, which is what we implicitly used in our informal proof (see subsection 4.1) as well. The measure function we provide is simply a function that returns the number of vertices in the game. The tactic creates three goals. We need to show that our measure function is a well-founded relation and we need to show for both recursive calls, that the measure function decreases. We obviously have that our measure function is a well-founded relation, so we continue with the other two goals. Through the whole proof, we assume that the condition on line 8 of Formalization 1 is false, since the case where the condition is true trivially terminates.

The first recursive call is `Zielonka (G -⇩G A)`. We prove that the number of vertices in `G -⇩G A` is smaller than in `G` by showing that `A` is not empty and `A` is a subset of the vertices of `G`. Luckily, since `A` is our generic attractor, we can use its assumptions to conclude that `A` is not empty and that `A` is a subset of the vertices of `G`. The only step left is to show that these assumptions also imply that the cardinality of the set of vertices of `G -⇩G A` is smaller than that of `G`. This is trivially true and was shown using the lemma `removeVerts_dec_cardinality`, which is a simple lemma saying that if we remove vertices from the game, the cardinality of the game's vertex set decreases.

The last recursive call (`Zielonka (G -⇩G att)`) is less trivial. We use the same approach as before, showing that `att` is both non-empty and a subset of the vertices of `G`. Since `att` is an attractor, we only need to show that the base of the attractor has these properties. In the proof, we use `W` to indicate this attractor base.

To start, it is proven that `W` is not empty. We suppose that `W` is empty and find a contradiction. We know that if `W` is empty, its attractor must also be empty. But we also know that `W` is not equal to its attractor since the recursive call is done where the condition on line 10 of Formalization 1 is false, thus contradicting that `W` is empty.

Next, instead of proving that `W` is a subset of the vertices of `G`, we prove the weaker but sufficient statement that there is at least one vertex in `W` that is also in `G`. We again construct a proof by contradiction. If we suppose that no such vertex exists, then we know that the attractor of `W` must equal `W`, again contradicting the condition on line 10 of Formalization 1.

Finally, we show again that both properties together imply that `G -⇩G att` is smaller than `G` using the same lemma as before. Now since both recursive calls strictly decrease the number of vertices, we have proven that `Zielonka` terminates.

4.2.5 *ZIELONKA_SIMP* terminates

To show that *ZIELONKA_SIMP* terminates, it is necessary to show that $Attr_a^{\mathcal{G}}(p\_set(max\_p(\mathcal{G}), \mathcal{G} \setminus A))$ is a generic attractor. This is formally written as **`interpretation`** `classical: gen_attr "λ G p. ParityGame.attractor G p (min_prios G)"` We do so by proving the two assumptions of a genertic attractor, namely that it is non-empty and that it is a subset of the vertices of the game.

That the attractor is non-empty is shown using the lemma `attractor_nonempty_for_min_prio_player` which

is a lemma that uses the fact that there is always a vertex of minimal priority in a non-empty game and that an attractor always contains its base to show that the attractor of all vertices of minimal priority cannot be empty. The attractor is also a subset of the vertices of the game since the base (the vertices with minimal priority in the game) is a subset of the game, which concludes the proof. We can now use the rule `classical.Zielonka.simps` to get the specific definition of *ZIELONKA_SIMP*. This rule has no domain assumptions, thus it is guaranteed that all inputs terminate.

4.2.6 *ZIELONKA terminates*

To show that *ZIELONKA* terminates, we use the same approach as for *ZIELONKA_SIMP*. Before we can show that *ATTR* is a generic attractor, we need to define it formally. The formal definition can be found in Formalization 2. Contrary to the definition of *ATTR* in Algorithm 1, the formalization of *ATTR* is tail-recursive. This is necessary since Isabelle/HOL has no support for loops. Also, this definition deviates from Algorithm 1 in the condition. Like for `Zielonka`, this stronger condition was added to allow for an unconditional termination proof.

ATTR terminates.

Since it is not immediately clear that this definition terminates, a termination proof for it has to be constructed before we can prove that `attr` is a generic attractor. This proof uses a similar well-founded relation as used in the generalized termination proof of `Zielonka`. Formally, the statement **`interpretation`** `verver: gen_attr attr` is what is proven.

Since in each recursive step, `A` increases in size, we prove that the number of vertices in `V⇘G↙ - A` strictly decreases. To prove this, we use the fact that `A` is a strict subset of `attrStep G p A`. This is true since the attractor of the game without `A` for the set of vertices with minimal priority is not empty. We can now conclude that the number of vertices in `V⇘G↙ - (attrStep G p A)` is strictly smaller than in `V⇘G↙ - A`, thus `attr` terminates.

ATTR is a generic attractor.

Since the proof that *ATTR* is a generic attractor is quite complex, it was decided to split the proof in two lemmas, each corresponding to one of the properties of a generic attractor. The lemmas can be found in Formalization 3. Both proofs use induction on the recursive calls of `attr_rec`, and are split in two cases, where the cases are that the condition in the definition of `attr_rec` is true or false.

The non-emptiness proof's induction hypothesis is that the result of the recursive call is non-empty. Since the case where the condition is true, `attr_rec` simply returns the result of the recursive call, this case is simply true because of the induction hypothesis. For the case that the condition is false, `attr_rec` returns `A` so it needs to be proven that `A ≠ {}`. We do so through a proof by contradiction. If we assume `A` to be empty, then we know, by definition of `p`, that `p` is the player of minimal priority in the game `G -⇓G A`. We also have, since we assume the game to not be empty, that the vertices of the game are not a subset of `A`. Combining these facts with our assumptions contradicts the fact that the condition is false. Thus `A` must not be empty. Since we have now shown that both possible paths in the definition of `attr_rec` results in a non-empty outcome, we can conclude that `attr_rec` (and therefore also `attr`) gives a non-empty result.

To prove that `attr_rec` is a subset of the game's vertices, we assume that the lemma holds for the recursive call. Since we assume that `A` is a subset of the game's vertices, we only need to consider the case where the condition in `attr_rec` is true. The induction hypothesis can prove this case if we prove that the assumptions of the induction hypothesis hold. Since `G` does not change in the recursive call, it is obviously still a parity game. We therefore only need to show that `attrStep G p A` is a subset of the vertices of the game. In the proof, this is done using the lemma `attrStep_in_V`, which is a lemma that shows that if `A` is such a subset, then `attrStep G p A` is as well, which is trivial considering the definition of `attrStep`.

Now that we have proven both lemmas, all that rests to do is to (trivially) combine them. This formally proves the following statement: **`interpretation`** `verver: gen_attr attr`. We can now use the rule `verver.Zielonka.simps` to get the specific definition of *ZIELONKA*. This rule has no domain assumptions, thus it is guaranteed that all inputs terminate.

4.2.7 *Strong condition is unneccesary*

Although the results of the previous section are already useful on their own, it is better if we eliminate the strong condition from the definition so that the formalization of *ZIELONKA* actually matches Algorithm 1. To achieve this, we give a new formalization `Zielonka_real` and show that it equals `Zielonka` under the assumption that the condition holds. Similarly, we give a new definition `attr_rec_real` that is equal to `attr_rec` if the condition holds. The definitions for `Zielonka_real` and `attr_rec_real` are the same as Formalization 1 and Formalization 2 respectively, except that `infinite V⇘G↙ ∨ ¬ ParityGame G` and `¬ (isEmpty G ∨ infinite V⇘G↙ ∨ ¬ A ⊆ V⇘G↙ ∨ ¬ ParityGame G)` have been removed from the respective conditions.

Under the assumption that the removed conditions hold, we can now show that `Zielonka_real` equals `Zielonla` and that `attr_rec_real` equals `attr_rec`. Although it may appear that it makes no difference if we remove the condition from the algorithm yet still assume it holds, there is actually a difference. If we leave the condition build into the algorithm, then the condition is re-evaluated in every recursive call. If, however, we just assume the condition to hold, we assume this only for the root call. So we need to prove that if we assume the condition to hold for the root call, it must also hold in its direct recursive calls. Furthermore, the proof is slightly more complex due to the way Isabelle handles partially terminating functions.

Strong condition in `Zielonka` *is unneccesary.*

The lemma that shows that the strong condition for `Zielonka` is unneccesary can be found in Formalization 4. It is proven using induction on the recursive calls of `Zielonka`. The induction hypothesis is that `Zielonka_real` terminates (using `Zielonka_real_dom`) and that it equals `Zielonka`. We distinguish the cases where the game is empty and where it is not. If the game is empty, then the lemma trivially holds so we continue with the assumption that the game is non-empty. In that case, we use the induction hypothesis to show that each recursive call of `Zielonka_real` terminates and that it is equal to `Zielonka`. For the first recursive call (`Zielonka_real (G -⇓G A)`), we can easily show that the assumptions of the lemma also imply the assumptions of the induction hypothesis. After all, the

Formalization 2 Formalization of $ATTR$ using strong condition

```
definition "attrStep G p A ≡ A ∪ ParityGame.attractor (G -⇩G A) p (min_prios (G -⇩G A))"
function attr_rec :: "'a ParityGame ⇒ Player ⇒ 'a set ⇒ 'a set" where
"attr_rec G p A = (if ¬ (V⇘G⇙ ⊆ A) ∧ ParityGame.winning_priority p (min_prio (G -⇩G A))
∧ ¬ (isEmpty G ∨ infinite V⇘G⇙ ∨ ¬ A ⊆ V⇘G⇙ ∨ ¬ ParityGame G) then attr_rec G p (attrStep
G p A)
else A)" by pat_completeness auto
abbreviation attr :: "'a ParityGame ⇒ Player ⇒ 'a set"
  where "attr G p ≡ attr_rec G p {}"
```

Formalization 3 Lemmas used to prove that $ATTR$ is a generic attractor

```
lemma attr_rec_nonempty:
  assumes "ParityGame G" "¬ isEmpty G ∧ finite V⇘G⇙" "p = (player (min_prio G mod 2))"
  shows "attr_rec G p A ≠ "
lemma attr_rec_in_v:
  assumes "ParityGame G" "A ⊆ V⇘G⇙"
  shows "attr_rec G p A ⊆ V⇘G⇙"
```

Formalization 4 Lemma that shows that the strong condition is unnecessary for `Zielonka`

```
lemma Zielonka_real_equals_Zielonka:
  assumes "¬ (infinite V⇘G⇙ ∨ ¬ ParityGame
G)"
  shows "Zielonka_real G = Zielonka G"
```

Formalization 5 Lemma that shows that the strong condition is unnecessary for `attr_rec`

```
lemma attr_rec_real_equals_attr_rec:
  assumes "¬ isEmpty G" "finite V⇘G⇙" "A
⊆ V⇘G⇙" "ParityGame G"
  shows "attr_rec_real G p A = attr_rec
G p A"
```

subgame of a finite parity game will remain a finite parity game. We can, therefore, state that `Zielonka_real (G -⇩G A)` terminates and that it equals `Zielonka (G -⇩G A)`. The same reasoning can also be applied to the next recursive call (`Zielonka_real (G -⇩G att)`) to show that it terminates and that it equals `Zielonka (G -⇩G att)`

Under the assumption that the game is non-empty, `Zielonka_real` could potentially terminate at two locations. Either `attr_stable G p** Z` is true and termination happens in the "then" part or it is false and termination happens in the "else" part. In the first case, there are no recursive calls anymore. `add_to_result` trivially terminates so we use the domain introduction rule of `Zielonka_real` to show that `Zielonka_real` terminates. Since `Zielonka_real (G -⇩G A)` is called prior to this, we need the fact that `Zielonka_real (G -⇩G A)` terminates under the induction hypothesis. In the other case, we additionally need that `Zielonka_real (G -⇩G att)` also terminates. Again, the domain introduction rule allows us to prove that also in this case `Zielonka_real` terminates. Since all possible paths of `Zielonka_real` terminate, we know that `Zielonka_real` terminates.

Using the fact that `Zielonka_real` terminates and that the recursive calls of `Zielonka_real` can be substituted by `Zielonka`, we can conclude that our lemma holds.

Strong condition in `attr_rec` *is unnecessary.*

Since `attr_rec` is a tail-recursive function, we can define `attr_rec_real` using Isabelle's **`partial_function`** with tail-recursion. This omits the need for a termination proof. Termination is implicitly proven when proving equality with `attr_rec`. We therefore only need to show its equality with `attr_rec` using the lemma given in Formalization 5. We do so using induction on `attr_rec`, assuming that the lemma holds for the recursive call. Since we assume the lemma to hold for the recursive call, we can show that the recursive call is equivalent to a call to `attr_rec`. All that needs to be done is to show that the assumptions of the induction hypothesis hold when the recursive call is reached. Since `G` does not change in the recursive calls, this is trivial using the assumptions of the lemma. The only assumption that cannot be directly concluded from the assumptions of the lemma is that `attrStep G p A ⊆ V⇘G⇙` but this is also trivially proven using the other assumptions. Combining the fact that the recursive call can be substituted by a call to `attr_rec` with the fact that the condition in `attr_rec_real` is equivalent to the condition in `attr_rec` under the current assumptions, we can conclude that the lemma holds.

Finally, we show that `attr_rec_real` can be interpreted as a generic attractor (**`interpretation verver_real: gen_attr attr_real`**). This is trivial as the assumptions used in the previous lemma to show equality with `attr_rec` are also in the assumptions of the generic attractor. We can then use `verver_real.Zielonka_real.psimps` in combination with the rule `verver_real.Zielonka_real_equals_Zielonka` to remove the domain predicate from the rule under the right assumptions, giving us a terminating definition of Zielonka's algorithm.

5. CONCLUSION

Zielonka's algorithm is used in practice in model-checking because of its simplicity and great performance. Enabling model-checking in a formal theorem prover would allow for complex proofs for any algorithm in that formal theorem prover. To reach this goal, the correctness of Zielonka's algorithm first needs to be proven formally. An essential step towards formally proving the correctness of Zielonka's algorithm is proving termination of the algorithm. In this paper, we have provided a formal proof of the termination of the algorithm. The next step is to prove its correctness. Afterwards, the correctness rule can be used for model-

checking by converting the model-checking problem to a parity game.

6. ACKNOWLEDGEMENT

I would like to thank Sebastiaan Joosten for his extensive guidance and feedback throughout this research and for his great technical support for Isabelle. Furthermore, I would like to thank the reviewers of the Software technology and Formal methods track for their useful comments and suggestions.

7. REFERENCES

[1] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7 – 34, 2003. Logic and Complexity in Computer Science.

[2] C. Dittmann. Positional determinacy of parity games. *Archive of Formal Proofs*, Nov. 2015. `http://isa-afp.org/entries/Parity_Game.html`, Formal proof development.

[3] E. Emerson, C. S. Jutla, and A. Sistla. On model checking for the μ-calculus and its fragments. *Theoretical Computer Science*, 258(1):491 – 522, 2001.

[4] O. Friedmann. Recursive algorithm for parity games requires exponential time. *RAIRO-Theoretical Informatics and Applications*, 45(4):449–457, 2011.

[5] O. Friedmann and M. Lange. Solving parity games in practice. In *International Symposium on Automated Technology for Verification and Analysis*, pages 182–196. Springer, 2009.

[6] J. Harrison. Formal proof - theory and practice. *Notices of the AMS*, 55(11):1395–1406, 2008.

[7] Y. Liu, Z. Duan, and C. Tian. An Improved Recursive Algorithm for Parity Games. In *2014 Theoretical Aspects of Software Engineering Conference*, pages 154–161, Sep. 2014.

[8] T. Nipkow, M. Wenzel, and L. C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.

[9] J. Obdrzalek. *Algorithmic analysis of parity games*. PhD thesis, University of Edinburgh, 2006.

[10] P. Thijssen. The begin of the Zielonka proof in Isabelle. Capita Selecta thesis, University of Twente, 2019.

[11] T. van Dijk. Oink: An Implementation and Evaluation of Modern Parity Game Solvers. In D. Beyer and M. Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 291–308, Cham, 2018. Springer International Publishing.

[12] M. Verver. Practical improvements to parity game solving. Master's thesis, University of Twente, 2013.

[13] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1):135 – 183, 1998.