Solving Parity Games by a Reduction to SAT

Keijo Heljanko [a] Misa Keinänen [b] Martin Lange [c]
Ilkka Niemelä [a]

[a] *Laboratory for Theoretical Computer Science, Helsinki University of Technology, Finland*

[b] *Space Systems Finland Ltd., Finland*

[c] *Department of Computer Science, University of Munich, Germany*

Abstract

This paper presents a reduction from the problem of solving parity games to the satisfiability problem in propositional logic (SAT). The reduction is done in two stages, first into difference logic, i.e. SAT combined with the theory of integer differences, an instance of the SAT modulo theories (SMT) framework. In the second stage the integer variables and constraints of the difference logic encoding are replaced with a set of Boolean variables and constraints on them, giving rise to a pure SAT encoding of the problem. The reduction uses Jurdziński's characterisation of winning strategies via progress measures. The reduction is motivated by the success of SAT solvers in symbolic verification, bounded model checking in particular. The paper reports on prototype implementations of the reductions and presents some experimental results.

Key words: Parity games, propositional satisfiability, difference logic

1 Introduction

Solving a parity game is an intrinsic and interesting problem in theoretical computer science. It is closely related to the problem of solving other games like mean pay-off, discounted pay-off or stochastic games [1–3]. It is also equivalent

Email addresses: Keijo.Heljanko@tkk.fi (Keijo Heljanko), Misa.Keinanen@ssf.fi (Misa Keinänen), Martin.Lange@ifi.lmu.de (Martin Lange), Ilkka.Niemela@tkk.fi (Ilkka Niemelä).

to the model checking problem for the modal μ-calculus [4]. Since the modal μ-calculus subsumes most of the commonly used temporal logics, solving parity games has direct applications in automatic program verification.

Solving a parity game is also one of the few inhabitants of the complexity class NP $\cap$ co-NP [5] not known to be in PTIME. However, the problem is in UP $\cap$ co-UP [2] and many people do conjecture that it is in PTIME. Many algorithms for solving parity games have been invented so far although none of them provably runs in deterministic polynomial time.

Recursive methods like Zielonka's algorithm [6], etc. solve a game with at most p different priorities by referring several times to games with strictly less than p many different priorities. Consequently, their running time is exponential in the number of priorities in the game.

Strategy improvement as done by Jurdziński and Vöge's algorithm [7] based on Puri's [1] – and similar to Hoffman and Karp's [8] as well as Ludwig's [9] algorithms for stochastic games – uses the fact that strategies can be partially ordered with a winning strategy being maximal w.r.t. this order. According to [7,10] this performs very well for some families of parity games but it is not known whether a polynomial number of iteration steps always suffices to find a winning strategy.

A randomised and subexponential algorithm is due to Björklund, Sandberg, and Vorobyov [11].

Every model checker for the full μ-calculus is in principle also an algorithm for solving parity games. Several of the former have emerged beginning with tableau-like methods by Stirling or Cleaveland [12,13], automata-theoretic ones by Emerson and Jutla [4], equation solvers by Cleaveland et al. [14] and Mader [15], and symbolic model checking procedures by Clarke et al. [16].

An algorithm with a good asymptotic complexity is Jurdziński's *small progress measures* procedure [17]. It is exponential in the number of odd priorities occurring in the game, i.e. in the half of the maximal priority. A similar asymptotic bound is achieved by Seidl's fixpoint iteration [18]. Recently, Jurdziński, Paterson and Zwick have found the first deterministic and subexponential ($n^{O(\sqrt{n})}$, where n is the number of nodes in the game) algorithm for solving parity games [19].

Opposing common undergraduate syllabi, polynomial time is not a synonym for efficiency. The famous SAT problem is NP-complete [20] and, hence, widely not believed to admit polynomial time algorithms. However, there are many SAT solvers that are astonishingly efficient in practice for many instances of the problem, zChaff [21] for example. Such solvers are used successfully, for example, in bounded model checking [22].

Inspired by this we present a different approach to solving parity games: by a reduction to SAT. The reduction is done in two stages, first into difference logic [23], i.e. SAT combined with the theory of integer differences, an instance of the SAT modulo theories (SMT) framework. This gives rise to a solving method for parity games through efficient difference logic solvers such as DPLL(T) [23]. In the second stage the integer variables and constraints of the difference logic encoding are replaced with a set of Boolean variables and constraints on them, giving rise to a pure SAT encoding of the problem. This enables a much wider range of solvers to be used than the SMT framework.

Theoretically this approach is not too exciting since it is clear that such a reduction must exists. Furthermore, SAT is believed to be harder than solving parity games. Again, clever heuristics and advanced search space pruning techniques implemented in current SAT solvers can make up for this and result in an algorithm that is efficient in practice.

Developing a *computationally attractive* reduction to SAT is often a non-trivial and challenging task. For example, even though it was known that a polynomial size reduction from parity games to SAT exists, no tight upper bound on the size of the reduction was reported in the literature. All known SAT checkers use algorithms whose worst case running time is exponential in the number of variables. Moreover, if the encoding is of substantial size, this can confuse the search heuristics and cause significant computational overhead in search space pruning. Hence, a computationally interesting reduction should introduce variables scantly and be of low polynomial size. This often means that the problem in question needs to be studied quite deeply to understand the essential properties of the solutions.

For the reduction from the parity game problem to SAT what we want is a formula of propositional logic that is satisfiable iff the existential player has a winning strategy in the parity game. It is rather straightforward to write a compact encoding whose models capture all possible strategies. However, it is more challenging to develop a concise set of conditions expressing the fact that a given strategy is winning. The novelty here is that we put forward a set of local constraints which express this property.

The reduction is based on a comment by Emerson where he explains inclusion of the model checking problem for the modal μ-calculus in NP. He essentially writes "Guess a rank for each μ-subformula at each state in a transition system. Show that the lexicographic order on the tuples through the transition system is well-founded" [24].

Following the idea about ranks consequently with the aim of a local characterisation of winning strategies we define the notion of a μ-annotation – effectively and unintentionally re-inventing Jurdziński's progress measures [17]. These are

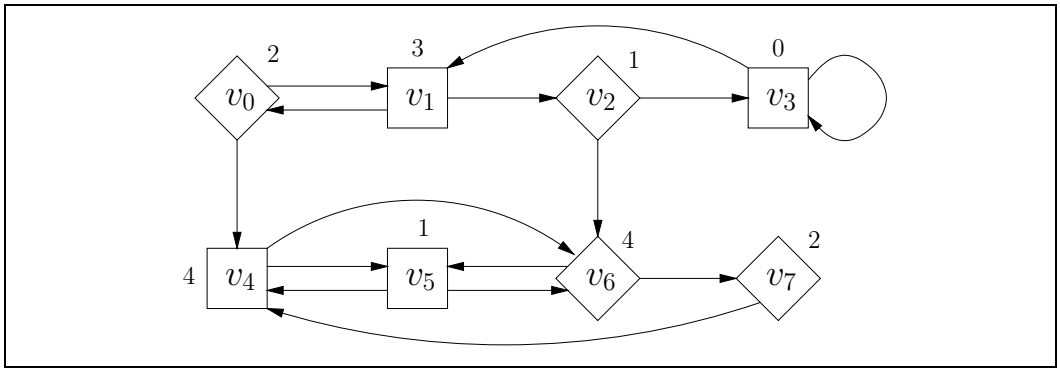

Fig. 1. An example of a parity game.

data structures consisting of local constraints which together ensure the global property that for all cycles of the parity game conforming to a certain strategy the parity of the least occurring priority in the cycle is even. Jurdziński's algorithm sets these data structures to an initial value and updates them iteratively. Our reduction leaves it entirely to the SAT solver to find their final values. Since this is not in any way an iterative procedure, we call these data structures μ-annotations in order to stress their static nature. We hereby attribute the theory of μ-annotations to Jurdziński explicitly. Nevertheless, we provide correctness proofs that differ slightly from Jurdziński's work [17] by reflecting this static nature.

The rest of the paper is organised as follows. Section 2 recalls the definitions of parity games. Since we are interested in solving them practically we restrict ourselves to finite parity games only. Section 3 contains the aforementioned theory regarding μ-annotations, resp. progress measures. The fact that we consider finite parity games only is essential for this part. Sections 5 and 6 present the reductions to difference logic and pure SAT, respectively. Section 7 presents experimental results of these translations, and Section 8 discusses further work.

2 Parity Games

A *parity game* is a tuple $G = (V, E, v_0, \Omega)$ where (V, E) is a finite, non-empty, directed graph and V is partitioned into two sets $V_\exists$ and $V_\forall$, $v_0 \in V$ is the *starting node*, and $\Omega : V \to \mathbb{N}$ is a *priority function*. G is assumed to be *total*, i.e. for every $v \in V$ there is a $w \in V$ with $(v, w) \in E$.

Example 1 An example of a parity game with eight nodes is depicted in Fig. 1. The three nodes making up $V_\exists$ are drawn using diamond shapes, the five nodes of $V_\forall$ are drawn using boxes. Here the node names are given inside the nodes, and the priority $\Omega(v)$ of a node v is shown beside the node v itself. The starting node is v_0 in the upper left corner.

We will use the following abbreviations: $V_p = \{v \in V \mid \Omega(v) = p\}$ for any p, and $V_{odd} = \bigcup_{p \text{ odd}} V_p$. We also write vE for the set $\{w \mid (v, w) \in E\}$.

A *play* of G is an infinite path $\pi = v_0 v_1 v_2 \dots$ through G starting in v_0. Intuitively, a play is constructed through interaction between the players: suppose the play has been partially constructed as $v_0 \dots v_i$. Then we have $v_i \in V_x$ for some $x \in \{\exists, \forall\}$, and player x chooses a $w \in V$ with $(v, w) \in E$ and the construction of the play continues with $v_0 \dots v_i w$.

Given a play $\pi = v_0 v_1 \dots$ let $\inf \pi = \{v \in V \mid$ there are infinitely many $i \in \mathbb{N}$ s.t. $v = v_i\}$. Remember that we assumed parity games to be finite, hence $\inf \pi \neq \emptyset$. Player $\exists$ wins the play $\pi = v_0 v_1 \dots$ if $\min\{\Omega(v) \mid v \in \inf \pi\}$ is even. If it is odd then player $\forall$ wins the play π.

Example 2 There are two types of plays in the game G shown in Fig. 1 starting in v_0.

a) Plays that stay in the upper four nodes forever.
b) Plays that eventually proceed into the lower four nodes and stay there forever.

Player $\exists$ wins any play of type (a). An odd priority 3 must be immediately followed by a smaller 2 or 1, and the other odd priority 1 is immediately followed by an even 0. Hence, the least priority occurring infinitely often must be even.

For plays of type (b) note that all priorities seen in the upper part do not influence the winner of this play because they can be seen at most finitely often. Hence, we can divide the plays of type (b) into two categories

b_1) those that traverse through the node with priority 1 infinitely often, and
b_2) those that traverse through the node with priority 1 only finitely often.

Then player $\forall$ must win plays of type (b_1), while player $\exists$ wins plays of type (b_2) since in the lower part of the game there is no odd priority other than 1.

A *strategy* for player x is a function $\sigma : V^* V_x \to V$, s.t. for all $v_0 \dots v_i$ with $v_i \in V_x$ we have $(v_i, \sigma(v_0 \dots v_i)) \in E$. Intuitively, such a strategy tells player x which choice to make depending on the current construction of a play.

A strategy is called *positional* if for all $\alpha, \beta \in V^*$ and all $v \in V_x$ we have $\sigma(\alpha v) = \sigma(\beta v)$. Hence, the choices made according to a positional strategy only depend on the last node visited. In such a case we will rather use $\sigma : V_x \to V$.

Positional strategies on finite games have the distinct advantage of being finite themselves. Furthermore, Thm. 4 below states intuitively that positional strategies are as powerful as arbitrary strategies. Hence, we will only consider positional ones and may simply speak of strategies when in fact we mean positional strategies.

A play $\pi = v_0 v_1 \ldots$ is called *conforming* to a (possibly positional) strategy σ for player x if for all $i \in \mathbb{N}$ we have: $v_i \in V_x$ implies $v_{i+1} = \sigma(v_0 \ldots v_i)$. A strategy σ for player x is called a *winning strategy* if every play conforming to σ is won by player x.

Example 3 Consider the following positional strategy σ_0 for player $\exists$ on the game G from Fig. 1 defined as $\sigma_0(v_0) = v_1$, $\sigma_0(v_2) = v_3$, $\sigma_0(v_6) = v_7$, and $\sigma_0(v_7) = v_4$. Every play starting in v_0 and conforming to σ_0 stays within the upper part of G from Fig. 1. Since only edges from $V_\exists$ lead out of the upper part and Ex. 2 established that all plays staying in that upper part are won by player $\exists$ we know that σ_0 is a winning strategy for player $\exists$.

Consider on the other hand any strategy σ_1 for player $\forall$ such that $\sigma_1(v_4) = \sigma_1(v_5) = v_6$, and the parity game G with any starting node among $\{v_4, \ldots, v_7\}$. Any play conforming to σ_1 will visit v_7 as the node with the minimum priority infinitely often. According to Ex. 2, the play is won by player $\exists$ and, hence, σ_1 is not a winning strategy for player $\forall$. However, player $\forall$ does indeed have a winning strategy for the game starting in v_i for $i = 4, \ldots, 7$, for example any strategy σ_2 such that $\sigma_2(v_4) = v_5$ and $\sigma_2(v_5) = v_4$.

Given a parity game G and a positional strategy σ for player $\exists$ we write $G|_\sigma$ for the parity game that is induced by σ on G. Formally, $G|_\sigma = (V, E \cap (V_\forall \times V \cup \{(v, \sigma(v)) \mid v \in V_\exists\}), v_0, \Omega)$. Note that $G|_\sigma$ is indeed a subgame of G, i.e. every play π in $G|_\sigma$ with winner P is also a play in G that is won by player P.

The problem of solving a parity game $G = (V, E, v_0, \Omega)$ is to determine whether or not player $\exists$ has a winning strategy for G. We formulate this as the decision problem PARITY defined as $\{G \mid \text{player } \exists \text{ has a winning strategy for } G\}$.

Theorem 4 [4] *Given a parity game G,*
(a) player $\exists$ has a winning strategy for G iff player $\forall$ does not have a winning strategy for G;
(b) a player has a winning strategy for G iff she has a positional winning strategy for G.

Theorem 5 [5] *The problem of solving a parity game is in NP$\cap$ co-NP.*

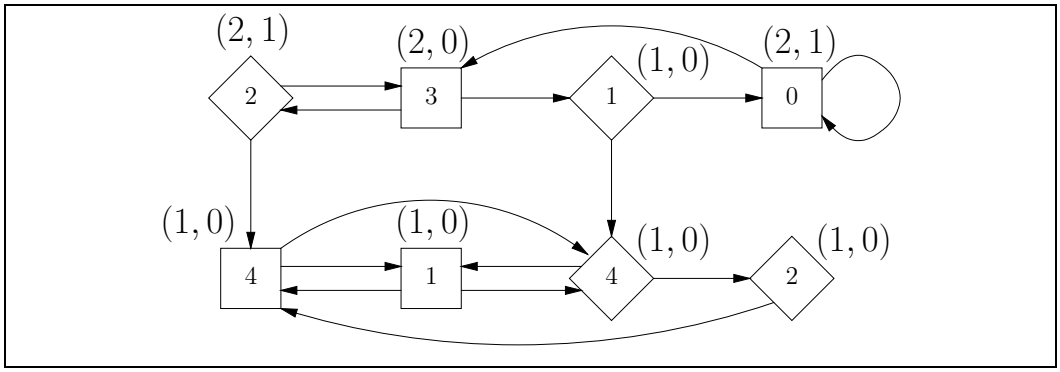

Fig. 2. A parity game with a μ-annotation.

3 Characterising Winning Strategies Locally

Given a parity game $G = (V, E, v_0, \Omega)$, let $Odd(G) = \{p \mid p \text{ is odd and } \Omega(v) = p \text{ for some } v \in V\}$, $Odd^{<p}(G) = Odd(G) \cap \{i \mid 1 \leq i < p\}$, and $mop(G) = \max\ Odd(G)$. Remember that G is assumed to be finite and non-empty. Thus, $mop(G)$ exists uniquely.

A μ-tuple for G is an $\overline{a} = (a_1, a_3, \ldots, a_{mop(G)}) \in \mathbb{N}^{\lceil mop(G)/2 \rceil}$. Given two μ-tuples $\overline{a} = (a_1, \ldots, a_{mop(G)})$ and $\overline{b} = (b_1, \ldots, b_{mop(G)})$ for G and a $p \leq mop(G)$ (not necessarily odd), we define

$$\overline{a} \trianglelefteq_p \overline{b} \quad \text{iff} \quad \begin{cases} \text{for all } i \in Odd^{<p}(G) : a_i \leq b_i & \text{if } p \text{ is even,} \\ a_p < b_p \text{ and for all } i \in Odd^{<p}(G) : a_i \leq b_i & \text{o.w.} \end{cases}$$

If $\overline{a} = (a_1, a_3, \ldots, a_{mop(G)})$ and $p \in Odd(G)$ then $\overline{a}^{(p)}$ denotes the p-component of $\overline{a}$, i.e. a_p.

A μ-annotation for G is a function η that assigns to each $v \in V$ a μ-tuple. It is called *successful*, iff for all $v \in V$:

- if $v \in V_\forall$ then for all $w \in vE$: $\eta(w) \trianglelefteq_{\Omega(w)} \eta(v)$, and
- if $v \in V_\exists$ then there is a $w \in vE$: $\eta(w) \trianglelefteq_{\Omega(w)} \eta(v)$.

Example 6 Consider, again, the parity game G depicted in Fig. 1. We have $Odd(G) = \{1, 3\}$ and $mop(G) = 3$. Thus, μ-tuples for G are of the form (a_1, a_3). A μ-annotation η of G is shown in Fig. 2. This time we do not show the nodes' names anymore. Instead, the number in a node denotes its priority, and its μ-tuple is shown as a label on that node.

It is not hard to see that η is not successful but the only node violating success is v_4 with its outgoing edge (v_4, v_5). According to the rules for success we would have to have $(1, 0) \trianglelefteq_1 (1, 0)$ which translates into $1 < 1$ and $0 \leq 0$. Remember

that it is crucial for player $\forall$ to move from v_4 to v_5 as shown in Ex. 3.

It is also easily verified that there are no μ-tuples $\eta(v_4) \trianglelefteq_4 \eta(v_5)$ and $\eta(v_5) \trianglelefteq_1 \eta(v_4)$ because this would entail $\eta(v_4)^{(1)} \leq \eta(v_5)^{(1)}$ while on the other hand $\eta(v_5)^{(1)} < \eta(v_4)^{(1)}$. Hence, there is no successful μ-annotation for G. However, take player $\exists$'s strategy σ_0 from Ex. 3 and consider the subgame $G|_{\sigma_0}$. It consists of the upper part $\{v_0, \ldots, v_3\}$ only, and the restriction of η to $G|_{\sigma_0}$ is indeed a successful μ-annotation for $G|_{\sigma_0}$.

This example insinuates that successful μ-annotations for subgames are closely related to winning strategies. In the following we will show that this is indeed the case.

Lemma 7 *Let $G = (V, E, v_0, \Omega)$ be a parity game, η be a successful μ-annotation for G, and $\pi = v_0 v_1 \ldots$ be a play of G. If for all $i \in \mathbb{N}$: $\eta(v_{i+1}) \trianglelefteq_{\Omega(v_{i+1})} \eta(v_i)$ then the minimal priority occurring infinitely often in π is even.*

PROOF. Suppose that the minimal priority p that occurs infinitely often in π is odd. Then there are infinitely many nodes $v_{i_1}, v_{i_2}, \ldots$ on π, s.t. $\eta(v_{i_1})^{(p)} > \eta(v_{i_2})^{(p)} > \ldots$ since eventually there is no lower even priority anymore that would allow $\eta(v_{i_j})^{(p)} < \eta(v_{i_{j+1}})^{(p)}$ for some $j \in \mathbb{N}$. But then we cannot have $\eta(v_{i+1}) \trianglelefteq_{\Omega(v_{i+1})} \eta(v_i)$ for all $i \in \mathbb{N}$, because the natural numbers are well-founded. $\square$

Theorem 8 *Let G be a parity game and σ be a positional strategy for player $\exists$. There is a successful μ-annotation for $G|_\sigma$ iff σ is a winning strategy.*

PROOF. ($\Rightarrow$) Suppose there is a successful μ-annotation η for the game $G|_\sigma = (V, E_\sigma, v_0, \Omega)$. Now take any play $\pi = v_0 v_1 \ldots$ that conforms to σ. Hence, π is also a play in $G|_\sigma$. But then we have $\eta(v_{i+1}) \trianglelefteq_{\Omega(v_{i+1})} \eta(v_i)$ for all $i \in \mathbb{N}$. According to Lemma 7 the minimal priority occurring infinitely often in π is even. Hence, player $\exists$ wins every such π and σ must indeed be a winning strategy.

($\Leftarrow$) Suppose σ is a winning strategy for the parity game $G = (V, E, v_0, \Omega)$. Let $G|_\sigma = (V, E_\sigma, v_0, \Omega)$. For every $p \in Odd(G)$ let $E_{\sigma,p} = E_\sigma \cap \{(v, w) \in E \mid \Omega(w) \geq p\}$ be the set of edges in $G|_\sigma$ that lead to nodes with priorities not less than p whilst conforming to σ. Furthermore, for every $v \in V$ let $W_v^p = \{w \mid (v, w) \in E_{\sigma,p}^+\} \cap V_p$ be the set of nodes that have priority p and are reachable from v via this relation, where $E_{\sigma,p}^+$ is the transitive closure of $E_{\sigma,p}$.

Now define a μ-annotation η for $G|_\sigma$ as $\eta(v)^{(p)} = |W_v^p|$ for every $v \in V$ and every $p \in Odd(G)$. It remains to be seen that η is successful.

Suppose it is not, then there a $v \in V$ and a $w \in vE_\sigma$ s.t. $\eta(w) \not\trianglelefteq_{\Omega(w)} \eta(v)$. Note that, if $v \in V_\exists$ then w is uniquely determined by σ. Otherwise the existence of such a w is guaranteed by totality.

Since $w \in vE_\sigma$, i.e. w is reachable from v whilst conforming to σ, we have $W_w^p \subseteq W_v^p$ for every $p \geq \Omega(w)$ and, hence, $\eta(w)^{(p)} \leq \eta(v)^{(p)}$ for all $p \leq \Omega(w)$. Since $\eta(w) \not\trianglelefteq_{\Omega(w)} \eta(v)$ by assumption it must be the case that $\Omega(w)$ is odd and $\eta(w)^{(\Omega(w))} \not< \eta(v)^{(\Omega(w))}$. Then it must be the case that $\eta(w)^{(\Omega(w))} = \eta(v)^{(\Omega(w))}$, in other words $W_w^{\Omega(w)} = W_v^{\Omega(w)}$. This means, in particular, $(w, v) \in E^+_{\sigma,\Omega(w)}$, i.e. v is reachable back from w through edges that are induced by the strategy σ whilst not seeing a priority greater than $\Omega(w)$. But then there is a play which conforms to σ on which the least priority seen infinitely often is $\Omega(w)$ which is odd. Thus, this play is won by player $\forall$ and, since it conforms to σ, this contradicts the fact that σ is a winning strategy. We conclude that η must indeed be a successful μ-annotation. $\square$

A direct consequence of this proof is the fact that the domain of annotation values can be bounded. A similar but slightly less optimising observation has also been made regarding progress measures [17]. Again, for a parity game $G = (V, E, v_0, \Omega)$ and a $p \leq mop(G)$ let E_p^+ denote the transitive closure of the relation $E \cap \{(v, w) \mid \Omega(w) \leq p\}$.

Corollary 9 *Let $G = (V, E, v_0, \Omega)$ be a parity game. There is a successful μ-annotation for G iff there is a successful μ-annotation η for G s.t. for all $v \in V$ and all $p \in Odd(G)$: $\eta(v)^{(p)} \leq |\{w \in V \mid (v, w) \in E_p^+\}|$.*

The μ-annotation values can furthermore be bounded by considering strongly connected components (SCC) of the game graph. Note that any infinite play on a finite game ultimately gets trapped in an SCC of the game. Furthermore, the winner of that play is entirely determined by the priorities of nodes from that SCC. This is because every suffix of an infinite play has the same set of infinitely occurring nodes as the play itself.

Let $G = (V, E, v_0, \Omega)$ be a parity game, $p \leq mop(G)$ and $scc : V \to \mathbb{N}$ a function that assigns to each node a unique index identifying the SCC containing that node. I.e. we have $scc(v) = scc(w)$ iff v is reachable from w and vice-versa. Furthermore, let E_p^+ be the transitive closure of the relation $E \cap \{(v, w) \mid scc(v) = scc(w) \text{ and } \Omega(w) \leq p\}$. Then Cor. 9 holds with this interpretation of E_p^+ as well, but a μ-annotation η is now successful iff for all $v \in V$:

- if $v \in V_\forall$ then for all $w \in vE$: $scc(v) = scc(w)$ implies $\eta(w) \trianglelefteq_{\Omega(w)} \eta(v)$, and
- if $v \in V_\exists$ then there is a $w \in vE$ s.t. $scc(v) \neq scc(w)$ or $\eta(w) \trianglelefteq_{\Omega(w)} \eta(v)$.

4 Difference Logic

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of Boolean variables and $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ a set of integer variables. The set of atomic formulas of difference logic consists of propositions in $\mathcal{P}$ and integer constraints of the forms $(x_i \geq x_j)$ and $(x_i > x_j)$ with $x_i, x_j \in \mathcal{X}$. The set $\mathcal{F}$ of all difference logic formulas [1] is the smallest set containing the atomic formulas which is closed under negation and conjunction:

- if $\phi \in \mathcal{F}$, then $\neg\phi \in \mathcal{F}$, and
- if $\phi \in \mathcal{F}$ and $\psi \in \mathcal{F}$, then $(\phi \wedge \psi) \in \mathcal{F}$.

The Boolean connectives $\vee, \rightarrow, \leftrightarrow$ are defined in the usual way in terms of $\neg$ and $\wedge$.

A $(\mathcal{P}, \mathcal{X})$ valuation consists of two functions [2] $\beta : \mathcal{P} \rightarrow \{\bot, \top\}$ and $\beta : \mathcal{X} \rightarrow \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers. The valuation is extended to all formulas in $\mathcal{F}$ by defining $\beta(x_i \geq x_j) = \top$ iff $\beta(x_i) \geq \beta(x_j)$, $\beta(x_i > x_j) = \top$ iff $\beta(x_i) > \beta(x_j)$, and applying the usual rules for the Boolean connectives. A formula $\phi \in \mathcal{F}$ is satisfied by a valuation iff $\beta(\phi) = \top$, and it is satisfiable if there exists a satisfying valuation. Given a formula $\phi \in \mathcal{F}$, the satisfiability problem is to decide whether or not ϕ is satisfiable.

Theorem 10 [25,20] *The satisfiability problem for difference logic is NP-complete.*

PROOF. NP-hardness follows directly from the fact that our logic subsumes propositional logic and membership in NP from the fact that the full difference logic is in NP, see e.g., [25]. □

An alternative way of showing membership in NP and to demonstrate the use of (our chosen subset of full) difference logic in an automata theoretic setting is the following nondeterministic algorithm. First replace each integer constraint $(x_i \geq x_j)$ and $(x_i > x_j)$ with new propositional variables $p_{(x_i \geq x_j)}$ and $p_{(x_i > x_j)}$. Guess a truth assignment for the resulting propositional formula, and check in polynomial time that it is satisfiable in the propositional sense. If it is, we still need to check that the conjunction of the integer constraints $(x_i \geq x_j)$ such that $\beta(p_{(x_i \geq x_j)}) = \top$ and $(x_i > x_j)$ such that $\beta(p_{(x_i > x_j)}) = \top$

[1] Our logic is actually a proper subset of the standard definition of difference logic over integers which allows integer constraints of the form $(x_i + k \geq x_j)$, where k is an arbitrary integer constant, see e.g., [23].

[2] Notice that the function β is overloaded for notational convenience.

is satisfiable. We can actually reduce this problem into the problem of Büchi automata emptiness with acceptance conditions on arcs as follows. Create a Büchi automaton with the state set $\mathcal{X}$, where all states are initial. The alphabet is a singleton set $\Sigma = \{a\}$, and the arcs are defined as follows. For each constraint $(x_i \geq x_j)$ such that $\beta(p_{(x_i \geq x_j)}) = \top$ add a non-accepting arc (x_i, a, x_j), and for each constraint $(x_i > x_j)$ such that $\beta(p_{(x_i > x_j)}) = \top$ add an accepting arc (x_i, a, x_j). Now it is straightforward to prove that the conjunction of the integer constraints induced by the propositional model will be satisfiable iff the Büchi automaton is empty. Because clearly checking the emptiness of a Büchi automaton can be done in polynomial time, the whole algorithm is in NP.

5 Encoding Winning Strategies in Difference Logic

Given a parity game $G = (V, E, v_0, \Omega)$ we build a difference logic formula Φ_G that is satisfiable iff player $\exists$ has a winning strategy in the game G. It contains Boolean variables S_v for every $v \in V$ and $T_{v,w}$ for every $(v, w) \in E$. They are used to guess a subgame of G inducing a positional strategy σ for the player $\exists$ in G.

In addition, Φ_G contains integer variables x_p^v for every $v \in V$ and every $p \in Odd(G)$ in order to model a μ-annotation. Φ_G is defined to be

$$(S_{v_0} \wedge \Phi_\exists \wedge \Phi_\forall \wedge \Phi_V \wedge \Phi_A).$$

Here, the subformulas are defined as follows:

$$\begin{aligned}
\Phi_\exists &= \bigwedge_{v \in V_\exists} (S_v \rightarrow \bigvee_{(v,w) \in E} T_{v,w}), \\
\Phi_\forall &= \bigwedge_{v \in V_\forall} (S_v \rightarrow \bigwedge_{(v,w) \in E} T_{v,w}), \\
\Phi_V &= \bigwedge_{v \in V, v \neq v_0} ((\bigvee_{(w,v) \in E} T_{w,v}) \rightarrow S_v), \text{ and} \\
\Phi_A &= \bigwedge_{(v,w) \in E} (T_{v,w} \rightarrow \Psi_{v,w}),
\end{aligned}$$

where $\Psi_{v,w}$ is given by

$$\Psi_{v,w} = \begin{cases} \bigwedge_{p \in Odd^{<\Omega(w)}(G)} (x_p^v \geq x_p^w) & \text{if } \Omega(w) \text{ even,} \\ (x_{\Omega(w)}^v > x_{\Omega(w)}^w) \wedge \bigwedge_{p \in Odd^{<\Omega(w)}(G)} (x_p^v \geq x_p^w) & \text{otherwise.} \end{cases}$$

Theorem 11 *Player $\exists$ has a winning strategy in the game G iff the difference logic formula Φ_G is satisfiable.*

PROOF. ($\Rightarrow$) Suppose player $\exists$ has a winning strategy σ for the game $G = (V, E, v_0, \Omega)$. This gives rise to an assignment β of the propositional variables S_v and $T_{v,w}$ for any $(v, w) \in E$: $\beta(S_v) = \top$, resp. $\beta(T_{v,w}) = \top$, if there is a play conforming to σ which visits the node v, resp. traverses the edge (v, w). It is not hard to see that the conjuncts S_{v_0}, $\Phi_\exists$, $\Phi_\forall$, and Φ_V are satisfied by β.

According to Theorem 8 there is a successful μ-annotation η for $G|_\sigma$. This gives rise to an assignment β to the non-propositional variables x_p^v for all the nodes of the play conforming to σ defined by $\beta(x_p^v) = \eta(v)^{(p)}$. Since η is successful, we have $\eta(w) \trianglelefteq_{\Omega(w)} \eta(v)$ for all $(v, w) \in E_\sigma$, and hence, the conjunct Φ_A is also satisfied. Altogether, there is a satisfying assignment for Φ_G.

($\Leftarrow$) Suppose β is a satisfying variable assignment for Φ_G. It is easy to derive from this a game $G|_\sigma = (V, E_\sigma, v_0, \Omega)$ as follows: for every node $v \in V_\exists$ such that $\beta(S_v) = \top$ add an arbitrary edge (v, w) to E_σ such that $\beta(T_{v,w}) = \top$, and for every node $v \in V_\forall$ such that $\beta(S_v) = \top$ add all edges $(v, w) \in E$ to E_σ. The conjuncts S_{v_0}, $\Phi_\exists$, $\Phi_\forall$ and Φ_V ensure that suitable edges needed by the construction above exist and that $G|_\sigma$ induced in this way is indeed a subgame of G.[3]

Furthermore, we can extract a μ-annotation η for $G|_\sigma$ defined by $\eta(v)^{(p)} = \beta(x_v^p)$ for any $v \in V$ conforming to σ and any $p \in Odd(G)$. (Should some $\beta(x_v^p)$ turn out to be negative, it is easy to see from our translation that the integer values of a satisfiable model can be made to positive integers by first offsetting every $\beta(x_{v'}^{p'})$ by a large enough positive integer offset o to make all integer variables of an assignment positive, and the formula still remains satisfiable with this assignment consisting of positive values only.) The conjunct Φ_A ensures that η is a successful μ-annotation for $G|_\sigma$ and, according to Theorem 8, σ is a winning strategy for player $\exists$ in the game G. $\square$

Proposition 12 *Given a parity game $G = (V, E, v_0, \Omega)$ with maximal odd priority p_{max}, the size of the difference logic formula Φ_G is $O(|E| \cdot \lceil \frac{p_{max}}{2} \rceil)$.*

The same size bound holds even if Φ_G is required to be in conjunctive normal form (CNF). In fact, by careful analysis of the difference logic formula Φ_G our implementation is able to rewrite it to difference logic in conjunctive nor-

[3] Note that a satisfying truth assignment can give rise to several winning strategies for the player $\exists$. It would be easy to change this by constraining each node of the player $\exists$ to have at most one outgoing edge.

mal form without blowup even without introducing any additional Boolean variables.

6 An Encoding into Propositional Logic

We present an encoding of the formula Φ_G for a parity game G into propositional logic, i.e. the subset of difference logic with Boolean variables only. Clearly, all that remains to be done is to translate the integer variables and constraints on them of the form $(x_p^v \geq x_p^w)$ and $(x_{\Omega(w)}^v > x_{\Omega(w)}^w)$.

Let $G = (V, E, v_0, \Omega)$ be the underlying parity game. By Cor. 9, the domain of the difference logic variables x_p^v for a fixed p and any v can be bounded by $|V_p| + 1$. Let $m_p = \lceil \log(|V_p| + 1) \rceil$ be the number of bits needed for a binary encoding of a value in the range $\{0, \ldots, |V_p|\}$. Hence, a set of propositional variables $x_{p,i}^v$ for $i \in \{0, \ldots, m_p - 1\}$ will be used to encode the value of each integer variable x_p^v.

For any $v, w \in V$, any $p \in \mathit{Odd}(G)$ and any $m \geq 1$ we present recursively defined propositional formulas *GreaterOrEquals* and *StrictlyGreater* parametrised by v, w, p, m and stating $0 \leq x_p^w \leq x_p^v < 2^m$, resp. $0 \leq x_p^w < x_p^v < 2^m$.

$$
\begin{aligned}
\mathit{GreaterOrEquals}(v, w, p, 0) &= x_{p,0}^w \rightarrow x_{p,0}^v \\
\mathit{GreaterOrEquals}(v, w, p, m) &= (x_{p,m}^w \rightarrow x_{p,m}^v) \ \wedge \ \big((x_{p,m}^w \vee \neg x_{p,m}^v) \rightarrow \\
&\quad \mathit{GreaterOrEquals}(v, w, p, m-1)\big) \\
\mathit{StrictlyGreater}(v, w, p, 0) &= x_{p,0}^v \wedge \neg x_{p,0}^w \\
\mathit{StrictlyGreater}(v, w, p, m) &= (x_{p,m}^w \rightarrow x_{p,m}^v) \ \wedge \ \big((x_{p,m}^w \vee \neg x_{p,m}^v) \rightarrow \\
&\quad \mathit{StrictlyGreater}(v, w, p, m-1)\big)
\end{aligned}
$$

Both formulas assert that the mth bit of x_p^v is greater or equals to the mth bit of x_p^w, and if they are equal then the same has to hold recursively for the next lower bit. However, formula *StrictlyGreater* has to ensure in the base case that at least the values of the lowest bits differ unless some higher bits have differed already.

The encodings of the integer constraints in Φ_G are simply replaced by

$$
\begin{aligned}
(x_p^v \geq x_p^w) &\text{ with } \mathit{GreaterOrEquals}(v, w, p, m_p), \text{ and} \\
(x_{\Omega(w)}^v > x_{\Omega(w)}^w) &\text{ with } \mathit{StrictlyGreater}(v, w, \Omega(w), m_{\Omega(w)}).
\end{aligned}
$$

We keep Φ_G as the name of the formula obtained by replacing all integer variables and all integer constraints on them by their Boolean counterparts.

Theorem 13 *Player $\exists$ has a winning strategy in the game G iff the propositional logic formula Φ_G (i.e. Φ_G with only Boolean variables) is satisfiable.*

PROOF. Immediate from the replacement of integer variables and constraints by their Boolean counterparts, together with Thm. 11 and Cor. 9. □

Proposition 14 [26] *Given a parity game $G = (V, E, v_0, \Omega)$ with maximal odd priority p_{max}, the size of the propositional logic formula Φ_G (i.e. Φ_G with only Boolean variables) is $O(|E| \cdot \lceil \frac{p_{max}}{2} \rceil \cdot \lceil \log |V| \rceil)$.*

The same size bound can be obtained also for a formula in conjunctive normal form (CNF) by introducing additional Boolean variables when needed in the CNF conversion process.

7 Experimental Results

We have implemented the translations from PARITY to difference logic and SAT presented in the previous sections. We use the DPLL(T) system [23] as the satisfiability solver for difference logic, and the well established zChaff [21] solver as the SAT solver. We also compare to an alternative approach for solving parity games by Keinänen and Niemelä [27] using the Smodels logic programming system [28], for further details on this approach, see [29].

All results are obtained on a machine with a 2GHz AMD Athlon64 processor and 2GiB of RAM. Times reported are the times in seconds to solve the formulas as reported by the Unix `/usr/bin/time` command. All benchmark runs employ a 1000 s timeout. Further implementation details, test setup, benchmarks, and discussion of the results can be found in [29], from which the following experimental results are from.

As the first set of benchmarks we use the family of *Jurdziński graphs* $J_{d,w}$, with parameters depth $d \in \mathbb{N}$ and width $w \in \mathbb{N}$. The parity game $J_{d,w}$ can be represented as a rectangle of $2d + 1$ rows and $2w$ columns as depicted in Fig. 3. The Corollary below follows directly from Theorem 12 in [17].

Corollary 15 *Given a Jurdziński graph $J_{d,w}$, the running time of the progress measure algorithm on $J_{d,w}$ is exponential in d.*

It is not hard to see – despite these games being difficult to solve for the small progress measures algorithm – that player $\exists$ has a winning strategy from every

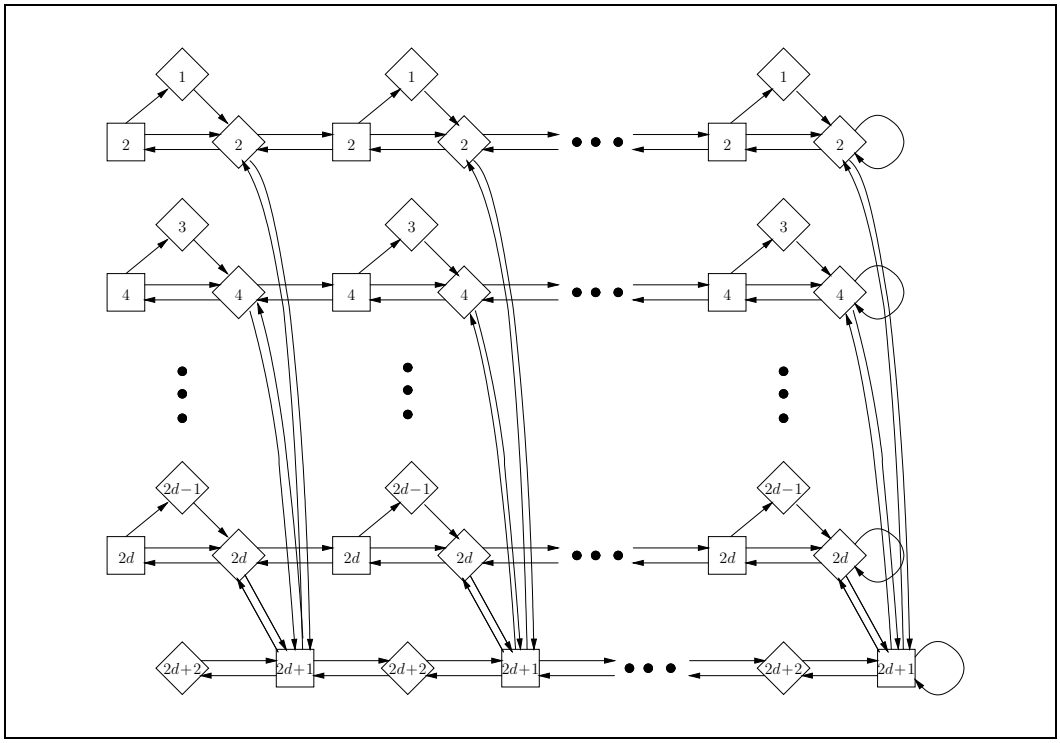

Fig. 3. The Jurdziński graphs $J_{d,w}$.

node in the first $2d$ rows whereas player $\forall$ has a winning strategy from every node in row $2d+1$. These strategies simply consist of moving to the right end of each row for example.

In the following we will always choose as the starting node the top leftmost node of the $\forall$ player. Hence, the resulting formulas under the reductions above are always satisfiable.[4] The results for the benchmarks in classes $J_{d,5}$ and $J_{d,10}$ are reported in Fig. 4. In these benchmarks we observe a better scaling of zChaff and Smodels over the difference logic approach of DPLL(T).

The second set of benchmarks used is a set of randomly generated parity games. These are generated by the following simple algorithm. For a parameter value n, start generating a game with a set of nodes of size n and generate exactly two outgoing edges for each node. Fix the initial state to node n_0, and discard all nodes not reachable from it. Experimentally roughly 80% of all nodes are reachable from n_0 on the average. For all the remaining nodes pick the player of each node with equal probabilities. The maximum priority m is another parameter value. We pick the priority of each node uniformly at random from the set $\{0, 1, \ldots, m-1\}$.

Fig. 5 reports running times with the parameter settings $m = \lceil \sqrt{n} \rceil$ for 31 ran-

[4] We note that unsatisfiable instances obtained by setting the starting node to be the leftmost node on the last row seem to be easier for the SAT solvers.

d	Jurdziński graph $J_{d,5}$			Jurdziński graph $J_{d,10}$		
	DPLL(T)	zChaff	Smodels	DPLL(T)	zChaff	Smodels
5	0.1	0.0	0.0	0.1	0.0	0.0
10	0.2	0.1	0.0	0.3	0.1	0.1
15	0.3	0.1	0.1	0.8	0.3	0.3
20	0.5	0.2	0.1	2.4	0.6	0.5
25	1.1	0.3	0.2	6.0	0.8	0.7
30	2.4	0.4	0.3	12.3	1.2	1.0
35	5.1	0.6	0.3	22.0	1.6	1.3
40	8.9	0.8	0.4	39.8	2.1	1.7
45	14.3	1.0	0.5	61.1	2.6	2.1
50	21.9	1.2	0.7	93.3	3.2	2.6
55	29.1	1.5	0.8	122.7	3.9	3.2
60	40.3	1.7	0.9	183.3	4.7	3.8
65	59.0	2.0	1.1	51.0	5.4	4.4
70	75.0	2.4	1.2	336.3	6.2	5.1
75	105.8	2.7	1.4	427.7	7.2	5.9
80	134.8	3.1	1.6	505.4	3.8	6.8

Fig. 4. The running times on the Jurdziński graphs $J_{d,5}$ and $J_{d,10}$.

n	#sat/#unsat	DPLL(T) min/median/max	zChaff min/median/max	Smodels min/median/max
100	20 / 11	0.1 / 0.1 / 0.1	0.0 / 0.0 / 0.0	0.0 / 0.0 / 0.7
200	23 / 8	0.1 / 0.1 / 0.1	0.0 / 0.1 / 0.1	0.0 / 0.8 / 265.4
300	23 / 8	0.1 / 0.1 / 0.2	0.1 / 0.1 / 0.3	0.1 / 4.8 / >1000
400	21 / 10	0.1 / 0.2 / 0.2	0.1 / 0.2 / 14.4	0.1 / 13.7 / >1000
500	21 / 10	0.1 / 0.2 / 0.3	0.1 / 0.3 / 1.1	0.3 / 40.9 / >1000
600	21 / 10	0.2 / 0.3 / 0.4	0.2 / 0.4 / 1.7	0.3 / 121.3 / >1000
700	17 / 14	0.2 / 0.4 / 0.7	0.3 / 0.6 / >1000	0.4 / 209.9 / >1000
800	20 / 11	0.2 / 0.5 / 0.7	0.3 / 1.3 / 12.8	0.6 / 362.5 / >1000

Fig. 5. The running times on random parity games with $m = \lceil \sqrt{n} \rceil$.

dom instances for every value of n. In this test setup the Smodels based approach does not scale very well. The zChaff and DPLL(T) based approaches are fairly close, with DPLL(T) winning slightly by having less variance in its running times.

8 Conclusions

We have shown how to reduce the problem of solving parity games both into difference logic and into SAT. Overall, the experimental results are encouraging considering the small amount of tuning effort that has been done so far to improve the translations. The reduction to finding a stable model of a logic program from [27] combined with the Smodels system does surprisingly well on Jurdziński graphs with a large number of priorities. However, the experiments on random graphs show that the approach is not always as robust as the results on Jurdziński graphs would let one believe. In any case, the approach

proved to be a worthy baseline for the new reductions of this work.

The reduction of solving parity games into SAT is mainly interesting because of the availability of highly efficient SAT checkers to solve the generated instances efficiently in practice. Also the rate of improvement in the performance of the state-of-the-art SAT checkers has been very high in recent years. This is expected to make the reductions even more attractive in the future. The reduction to SAT is also interesting because it enables a smooth integration of this kind of a parity game solver to other SAT based technologies. For example, bounded model checking of sequential Boolean circuits could have the property specified using alternating parity automata as in [30]. In a hypothetical bounded model checker for sequential Boolean circuits the encoding into SAT would most probably be dominated by the size of the encoding of the transition relation of the sequential Boolean circuit and the parity game would be a much smaller property checking subproblem. In such an application domain we want to be able to express the problem of solving parity games in SAT but the main difficulty in the problem solved by the SAT solver comes from the encoding of the executions of the sequential Boolean circuit.

It is not hard to extend this reduction to a *global* parity game solver – an algorithm that computes for each node v the player that has a winning strategy for the game starting in this node. Because of Theorem 4 there is always exactly one player who has a winning strategy. The straightforward extension roughly doubles the number of variables and clauses in the formulas. Every node is equipped with two data structures: a μ-annotation and a dually defined ν-annotation. The formula then asserts that, depending on which winning set this node belongs to, it is either the μ- or the ν-annotation that has to be locally successful. Using the duality of the problem could also be exploited during the translation to choose between guessing a strategy for either the existential or the universal player and, for example, always choose the one where the search space of the potential strategies is minimised.

As further work it would be interesting to try how well SAT based approaches could work as subroutines in recursive procedures like [19], in particular as a subroutine to find so called "small dominions" of a parity game [19]. Another line of work based on our new reductions would be to extend bounded model checking algorithms for linear time properties to allow for more efficient handling of alternating parity automata as the specification formalism, continuing work along the lines of [30–32].

Acknowledgements We would like to thank Jan Johannsen for suggesting to solve parity games by doing a piggy-back on the recent advance in SAT solving, Jan Obdržálek for very helpful but unfortunately less successful discussions about solving parity games of bounded tree- and clique-width, and

Marcin Jurdziński for inspiring comments about difficult parity games. We would also like to thank Albert Oliveras and Robert Nieuwenhuis for their excellent support for the DPLL(T) solver.

References

[1] A. Puri, Theory of hybrid systems and discrete event systems, Ph.D. thesis, University of California, Berkeley (1995).

[2] M. Jurdziński, Deciding the winner in parity games is in $UP \cap \text{co-}UP$, Information Processing Letters 68 (3) (1998) 119–124.

[3] U. Zwick, M. Paterson, The complexity of mean payoff games on graphs, Theoretical Computer Science 158 (1–2) (1996) 343–359.

[4] E. A. Emerson, C. S. Jutla, Tree automata, μ-calculus and determinacy, in: Proceedings of 32nd Symposium on Foundations of Computer Science, IEEE, San Juan, Puerto Rico, 1991, pp. 368–377.

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

[6] W. Zielonka, Notes on finite asynchronous automata, R.A.I.R.O. – Informatique Théorique et Applications 21 (1987) 99–135.

[7] M. Jurdziński, J. Vöge, A discrete strategy improvement algorithm for solving parity games, in: E. A. Emerson, A. P. Sistla (Eds.), Proceedings of 12th International Conference on Computer Aided Verification, CAV'00, Vol. 1855 of Lecture Notes in computer Science, Springer, 2000, pp. 202–215.

[8] A. Hoffman, R. M. Karp, On nonterminating stochastic games, Management Science 12 (1966) 359–370.

[9] W. Ludwig, A subexponential randomized algorithm for the simple stochastic game problem, Information and Computation 117 (1) (1995) 151–155.

[10] D. Schmitz, J. Vöge, Implementation of a strategy improvement algorithm for finite-state parity games., in: S. Yu, A. Paun (Eds.), CIAA, Vol. 2088 of Lecture Notes in Computer Science, Springer, 2000, pp. 263–271.

[11] H. Björklund, S. Sandberg, S. G. Vorobyov, A discrete subexponential algorithm for parity games, in: H. Alt, M. Habib (Eds.), STACS, Vol. 2607 of Lecture Notes in Computer Science, Springer, 2003, pp. 663–674.

[12] R. Cleaveland, Tableau-based model checking in the propositional μ-calculus, Acta Informatica 27 (8) (1990) 725–748.

[13] C. Stirling, Local model checking games, in: I. Lee, S. A. Smolka (Eds.), Proceedings of 6th Conference on Concurrency Theory, CONCUR'95, Vol. 962 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 1995, pp. 1–11.

[14] R. Cleaveland, M. Klein, B. Steffen, Faster model checking for the modal mu-calculus., in: G. von Bochmann, D. K. Probst (Eds.), Proceedings of 4th International Conference on Computer Aided Verification, CAV '92, Vol. 663 of Lecture Notes in Computer Science, Springer, 1992, pp. 410–422.

[15] A. Mader, Modal μ-calculus, model checking and Gauß elimination, in: E. Brinksma, R. Cleaveland, K. G. Larsen, T. Margaria, B. Steffen (Eds.), Proceedings of 1st International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'95, Vol. 1019 of Lecture Notes in Computer Science, Springer, 1995, pp. 72–88.

[16] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang, Symbolic model checking: 10^{20} states and beyond, Information and Computation 98 (2) (1992) 142–170.

[17] M. Jurdziński, Small progress measures for solving parity games, in: H. Reichel, S. Tison (Eds.), Proceedings of 17th Annual Symposium on Theoretical Aspects of Computer Science, STACS'00, Vol. 1770 of Lecture Notes in Computer Science, Springer, 2000, pp. 290–301.

[18] H. Seidl, Fast and simple nested fixpoint, Information Processing Letters 59 (6) (1996) 303–308.

[19] M. Jurdziński, M. Paterson, U. Zwick, A deterministic subexponential algorithm for solving parity games, in: Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA'06, ACM/SIAM, 2006, pp. 114–123.

[20] S. A. Cook, The complexity of theorem-proving procedures, in: Conference Rec. 3rd Annual ACM Symposium on Theory of Computing, STOC'71, ACM, Shaker Heights, OH, USA, 1971, pp. 151–158.

[21] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, in: Proceedings of 38th Design Automation Conference, DAC'01, 2001, pp. 530–535.

[22] E. M. Clarke, A. Biere, R. Raimi, Y. Zhu, Bounded model checking using satisfiability solving, Formal Methods in System Design 19 (1) (2001) 7–34.

[23] R. Nieuwenhuis, A. Oliveras, DPLL(T) with exhaustive theory propagation and its application to difference logic, in: K. Etessami, S. K. Rajamani (Eds.), CAV, Vol. 3576 of Lecture Notes in Computer Science, Springer, 2005, pp. 321–334.

[24] E. A. Emerson, Model checking and the μ-calculus, in: N. Immerman, P. G. Kolaitis (Eds.), Descriptive Complexity and Finite Models, Vol. 31 of DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, AMS, 1997, Ch. 6.

[25] M. Mahfoudh, P. Niebert, E. Asarin, O. Maler, A satisfiability checker for difference logic, in: Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing, SAT'02, 2002, pp. 222–230.

[26] M. Lange, Solving parity games by a reduction to SAT, Proceedings of International Workshop on Games in Design and Verification, GDV'05 (2005, unpublished).
URL http://www.tcs.ifi.lmu.de/~mlange/papers/gdv05.pdf

[27] M. Keinänen, I. Niemelä, Solving alternating Boolean equation systems in answer set programming, in: D. Seipel, M. Hanus, U. Geske, O. Bartenstein (Eds.), INAP/WLP, Vol. 3392 of Lecture Notes in Computer Science, Springer, 2004, pp. 134–148.

[28] P. Simons, I. Niemelä, T. Soininen, Extending and implementing the stable model semantics, Artificial Intelligence 138 (1-2) (2002) 181–234.

[29] M. Keinänen, Techniques for solving Boolean equation systems, Ph.D. thesis, Helsinki University of Technology, Laboratory for Theoretical Computer Science (2006).

[30] M. Jehle, J. Johannsen, M. Lange, N. Rachinsky, Bounded model checking for all regular properties, Electronic Notes in Theoretical Computer Science 144 (1) (2006) 3–18.

[31] A. Biere, K. Heljanko, T. Junttila, T. Latvala, V. Schuppan, Linear encodings of bounded LTL model checking, Logical Methods in Computer Science 2 (5:5), (doi: 10.2168/LMCS-2(5:5)2006).

[32] K. Heljanko, T. Junttila, M. Keinänen, M. Lange, T. Latvala, Bounded model checking for weak alternating Büchi automata, in: Proceedings of the 18th International Conference on Computer Aided Verification (CAV'2006), Vol. 4144 of Lecture Notes in Computer Science, 2006, pp. 95–108.