Weak Parity Games and Language Containment of Weak Alternating Parity Automata

Jan Hoffmann
Institut für Informatik
LMU München

March 13, 2006

1 Introduction

Optimisations of the state space of weak alternating parity automata (WAPA) are needed for example to speed up μTL model-checking algorithms which involve WAPA. It is assumed that deciding language containment for WAPA is helpful to perform such optimisations. In this paper the problem of language containment is reduced to the problem of computing winning sets in weak parity games. For the latter a linear time algorithm is presented.

This section gives some notes on previous work on language containment and optimisations of finite automata on finite and infinite words as well as an overview of the text.

In the second section one finds the definition of parity games and weak parity games as well as an algorithm that computes the winning sets in weak parity games in linear time. This matches the best known algorithms for deciding winning sets in this games. An algorithm that computes winning set in arbitrary parity games can be found in [Jur00]. As a special case the algorithm of Jurdzinski ([Jur00]) solves weak parity games in linear time too but it is not as intuitive and easy too implement as the algorithm which is presented here.

In the third section the definition of WAPA is repeated and some notes on the optimisation of the state space with the help of language containment in this automata are made.

The last section describes two different reductions of language containment of WAPA into deciding winning sets in weak parity games which delivers a polynomial time algorithm for language containment that is sound but not compete and a PSPACE algorithm for language containment that is sound and complete. The algorithms have been implemented in the μSabre project of Martin Lange and Jan Johannsen at the University of Munich.

It is well known that the problem to decide weather $L(A_1)$ is a subset of $L(A_2)$ or not for given deterministic finite automaton (DFA) A_1 and A_2 is quite easy. One only has to construct a DFA for the language $L(A_1) \cap \overline{L(A_2)}$ and test if there is a final state reachable from the initial state. In fact this can be done in time $O(n^2)$. An algorithm is for example given in [Hop02].

A generalisation to nondeterministic finite automaton (NFA) makes the language containment problem quite difficult: language containment for NFA is

NP-complete (see [Hop02] for a proof). Therefore it is not surprising that the language containment problem for Büchi automata and for weak alternating parity automata are not easy either: PSPACE-complete [Sis87].

There have been efforts to reduce the state space in Büchi automata efficiently [Hen97, Hen00] with the help of algorithms for language containment. Thereby the notions of fair and direct simulation have turned out to be helpful. The best known algorithms for fair and direct simulation in Büchi automata run in cubic time [Ete01]. In these terms this paper introduces a fair simulation algorithm for WAPA which runs in polynomial time. A similar algorithm for that purpose has been given in [Fri05, Fri02] before.

2 Parity Games and Winning Sets

A parity game is a directed graph in which the vertices are labeled with two values: an integer which is the priority of the vertex and a value $p \in \{0, 1\}$ that represents the player of the vertex. This graph can be thought of as an infinite game of two players.

Definition 1 (Parity Game) A *parity game* $\Gamma = (V, E, p, (V_0, V_1))$ consists of a directed graph (V,E) in which every vertex $v \in V$ has an outgoing edge $(v, u) \in E$, a priority function $p : V \to \mathbb{N}$ and a partition (V_0, V_1) of V (i.e. $V = V_0 \cup V_1$ and $V_0 \cap V_1 = \emptyset$).

A parity game is played by to players P_0, P_1 as follows: The play starts at a fixed vertex $v_0 \in V$. If the play has reached a vertex $v \in V_i$ then player P_i chooses a vertex $w \in V$ with $(v, w) \in E$ and the play is continued at w. This leads to an infinite path $\pi = v_0 v_1 v_2 ...$ of vertices which is called a play in Γ.

Write $Inf(\pi)$ for the set of vertices which occur infinitely often in π. If $min\{p(v) \mid v \in Inf(\pi)\}$ is even then P_0 wins π else P_1 wins π.

A strategy for a player P in a parity game is a function that is defined for all vertices on wich it is P's turn and that returns a successor of the input vertex. If a player P who follows a strategy wins every game that starts from a fixed vertex v then the strategy is a winning strategy for P from v.

Definition 2 (Winning Strategy) Let $\Gamma = (V, E, p, (V_0, V_1))$ be a parity game and $\pi = v_0 v_1 v_2 ...$ a play in Γ.

$\sigma : V_i \to V$ is called a *strategy* for player P_i if $(v, \sigma(v)) \in E$ for all $v \in V_i$. π is *consistent* with σ if $v_{k+1} = \sigma(v_k)$ for all $v_k \in V_i$.

σ is a *winning strategy* for P_i from $W \subseteq V$ if every play that starts at a vertex in W and that is consistent with σ is won by P_i.

As a result of the following theorem for a given Vertex v in a parity game exactly one player has a winning strategy from v.

Theorem 1 (Memoryless Determinacy [Eme91, Mos91]) *For every parity game, there is a unique partition (W_0, W_1) (the winning sets) of the set of vertices of its game graph, such that there is a winning strategy for player P_0 form W_0 and a winning strategy for player P_1 from W_1.*

The problem of computing winning sets in parity games has an interesting status from the point of view of structural complexity theory. On the one hand

it is very unlikely that the problem is NP-complete because it is known to be in NP ∩ co-NP ([Eme91]). On the other hand no polynomial time algorithm for the problem is known.

As mentioned in the introduction there exist algorithms to compute the winning sets in parity games. By now the best known approach is the algorithm of Jurdzinski ([Jur00]) which runs in space $O(dn)$ and time

$$O\left(dm \cdot \left(\frac{n}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}\right),$$

where n is the number of vertices, m is the number of edges and d is the number of priorities of the parity game.

The reduction from language containment in WAPA to deciding winning sets, presented in the last section, requires only a fragment of the parity games: weak Büchi games. These are games which consist only of vertices with two priorities that satisfie a structural restriction to the game graph.

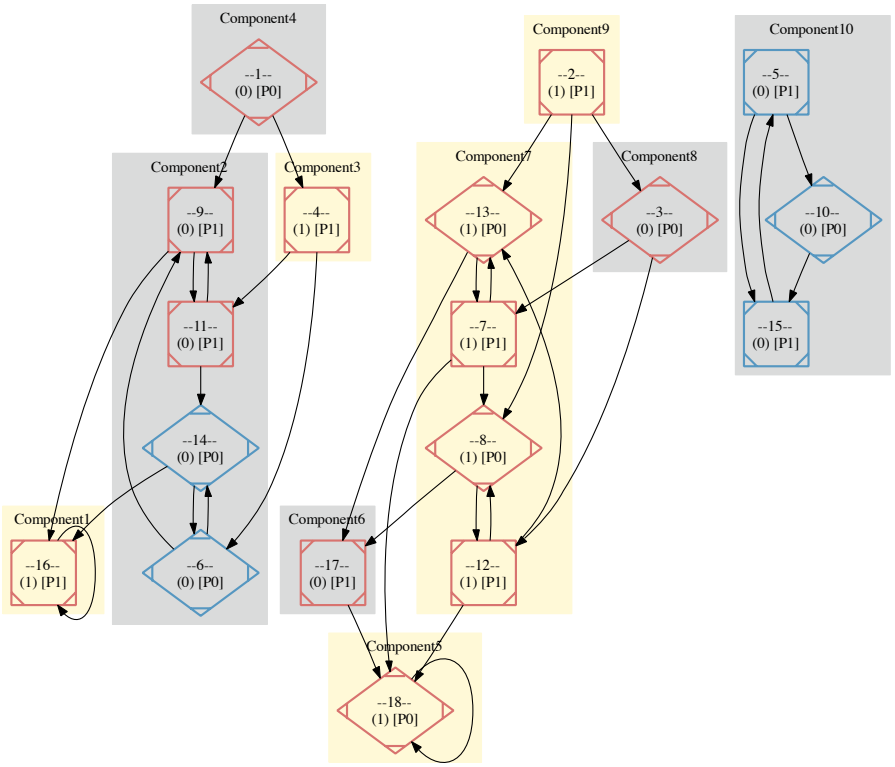

Figure 1: A weak Büchi game with winning positions.

Definition 3 Let $\Gamma = (V, E, p, (V_0, V_1))$ be a parity game. Γ is called *Büchi game* if $p(v) \leq 1$ for all $v \in V$. Γ is called a *weak Büchi game* if Γ is a Büchi game and for every game π in Γ it holds that $p(u) = p(v)$ for all $u, v \in Inf(\pi)$.

Example 1 Figure 1 shows a weak Büchi game with $V = \{1, 2, ..., 18\}$. Vertices that belong to V_0 are diamonds and labeled with $[P_0]$. Vertices that belong to V_1 are rectangles and labeled with $[P_1]$. The priorities are given in round brackets, e.g. (0). Player P_0 has a winning strategy from the vertices 5,6,10,14 and 15 which are blue and player 1 has a winning strategy from the red vertices 1,2,3,4,5,7,8,9,11,12,13,17 and 18.

Lemma 1 *Let $\Gamma = (V, E, p, (V_0, V_1))$ be a weak Büchi game. Let C be a strongly connected component of G=(V,E). Then it holds that $p(v) = p(u)$ for all vertices $u, v \in C$.*

PROOF Let $u, v \in C$. Assume that $p(v) \neq p(u)$. Because v and u are in the same component there exist a path $\mu = uu_1u_2...u_nv$ from u to v and a path $\nu = vv_1v_2...v_ku$ from v to u. Therefor one can construct an infinite game $\pi = \mu\nu\mu\nu\mu....$ By the assumption it follows that $Inf(\pi) = \{0, 1\}$. This is a contradiction to the fact that Γ is finally monotone.□

The algorithm of Jurdzinski can compute winning sets in weak parity games in time $O(m \cdot n)$ and space $O(n)$. The proof of the following theorem contains an algorithm that computes winning sets in weak Büchi games in time and space $O(m)$.

Theorem 2 *The winning sets for a given weak Büchi game can be computed in time and space $O(m)$ where m is the number of edges in the game graph.*

PROOF Let $\Gamma = (V, E, p, (V_0, V_1))$ be a weak Büchi game and $G = (V, E)$ the game graph of Γ.

The following algorithm computes a value $S[v]$ for each $v \in V$ such that $S[v] = i$ if and only if player P_i has a winning strategy from v in Γ. For each $v \in V$, $c[v]$ is an integer which indicates the number edges $(v, u) \in E$ with $S[u] \neq S[v]$.

If $C_1, ..., C_n$ is the decomposition of G into its strongly connected components and $v \in V$ is a vertex then $comp[v] \in \{1, ..., n\}$ denotes the component of v. Let L_i, $i \in \mathbb{N}$, be a FIFO list with the following operations: 'get(L_i)' is the first element of L_i, 'put(v,i)' puts v into the first position of L_i. At the beginning of the algorithm all L_i are empty. For $v \in V$, $deg[v]$ denotes the out-degree of v.

```
WinningSets(Γ)
1    Decompose G=(V,E) into its strongly connected components
2    C_1 < ... < C_n with the topological ordering '<' on the C_i.
3    for each v ∈ V do
4        c[v] ← 0
5        S[v] ← −1
9    for i=n to 1 do
10       while L_i ≠ ∅ do
11           v ← get(L_i)
12           Lift(v)
13       for each v ∈ C_i with S[v] = −1 do
14           S[v] ← p(v)
15           Lift'(v)
```

```
LIFT(v)
1    for each (u,v) ∈ E with S[u] = −1 do
2        if u ∈ V_{S[v]} then
3            S[u] ← S[v]
4            put(u,comp[u])
5        else
6            c[u] ← c[u] + 1
7            if c[u] = deg[u] then
8                S[u] ← S[v]
9                put(u,comp[u])
```

The call of LIFT'(v) in line 15 of WINNINGSETS means a call of LIFT (v) with line 1 replaced with

```
1'   for each (u,v) ∈ E with S[u] = −1 and comp[u] < comp[v] do
```

The decomposition of a graph into its strongly connected components can be done in time $O(|V| + |E|)$ by two DFS searches as e.g. described in [Cor01]. This algorithm can be easily changed to additionally determine a topological ordering without increasing the running time.

It is easy to see that a vertex v is only put into a list L_i if $S[v] = -1$ and then $S[v]$ is changed to a value $k \neq -1$. Therefore each vertex is put at most once into a list and the lines 11-12 of WINNINGSETS are executed at most $|V|$ many times. Obviously the lines 4-5 and 14-15 are repeated at most $|V|$ times too.

By the discussion above it is also clear that LIFT(v) is called exactly once for each $v \in V$. For that reason the lines 2-9 of LIFT are executed at most $|E|$ times.

Together this yields the upper bound $O(|V| + |E|)$ on the running time of WINNINGSETS. Except for some counters for the loops the only thing that has to be stored in a run of the algorithm is the array c which consists of a counter $c[v]$ for each vertex $v \in V$. Since $c[v] \leq deg[v]$ for every $v \in V$, c can be stored in space $O(|V| + |E|)$. Because of the assumption that every vertex has an outgoing edge it holds that $O(|V| + |E|) = O(|E|)$. That is why $O(|E|)$ is a bound on the time and space needed by WINNINGSETS.

To show the correctness of the algorithm it is sufficient to show that in every case the assignment "$S[v] \leftarrow i$" is correct. First one sees easily that the algorithm sets $S[v]$ in the reversed topological order, i.e. if an assignment $S[v] \leftarrow i$ is made for a vertex $v \in C_k$ then for every $k < j \leq n$ and for all $u \in C_j$ the value $S[u]$ has been already set.

Suppose now for a given $u \in C_k$ the assignment $S[u] \leftarrow i$ is made. If $k = n$ then $S[u]$ is set in the first run of the outer loop of WINNINGSETS. Therefore it holds that $i = p(u)$ and the correctness follows by Lemma 1. If $k < n$ then we can assume by induction hypothesis that all values which were set before have been correct. There are two cases in which i could be assigned to $S[u]$:

- In a call of LIFT(v): Then there is an edge $(u, v) \in E$ and $i = S[v]$. Recall that LIFT(v) is only called if $S[v] \leftarrow j$ already has been set.

  If $S[u]$ is assigned in line 3 then u is in $V_{S[v]}$. So if a play is at position u then player $S_{S[v]}$ can chose v to be the next position. By ind. hyp. player $S_{S[v]}$ has a winning strategy from v. Therefore the other player can not

have a winning strategy from u and then it follows by Theorem 1 that $S_{S[v]}$ has one.

If $S[u]$ is assigned in line 8 then u is not in $V_{S[v]}$. But then it follows by ind. hyp. and an induction over $deg[u]$ that $S_{S[v]}$ has a winning strategy from all successors of u. By a similar argument as above $S_{S[v]}$ therefore has a winning strategy from u.

- In line 14 of WinningSets: Then $i = p(u)$. If $u \in V_{p(u)}$ then there must be a successor w of u in C_k with $S[w] = -1$. By Lemma 1 it follows that $p(w) = p(u)$. If u is not in $V_{p(u)}$ then it must hold for all successors w of u that either $S[w] = p(u)$ or $w \in C_k$ and $S[w] = -1$ (recall that $S[v]$ has been set already for all vertices in topological higher components). That is why one can use Theorem 1 to see that there must be a winning strategy for player $S_{p(u)}$ for all $u \in C_k$ with $S[u] = -1$.

Because all assignments "$S[v] \leftarrow i$" are made if and only if player S_i has a winning strategy from v, WinningSets is correct. □

3 Optimisation of Weak Alternating Parity Automata

Since J. R. Büchi has introduced finite automata on infinite words to decide monadic second order logic ([Büc62]) a lot of similar automata have been presented. One of these automata models are weak alternating parity automata (WAPA) which have been analysed at first in a work of Muller, Saoudi and Schupp ([Mul86]). Weak alternating parity automata are interesting because they define acceptance only in terms of reachability but nevertheless have the same expressive power than Büchi automata, i.e. they are able to describe the ω-regular properties ([Kup01]).

Apart from other advantages they allow a simple complementation ([Löd00]) and are closely related to the linear time μ-calculus ([Lan04]). Since there are efficient translation from the linear time μ-calculus to weak alternating parity automaton they are being used in model-checking tools for regular properties ([Lan04],[μSabre]).

In this section the definition of weak alternating parity automata is given. Afterwards some notes on problems that occur in optimisation of the state space with the help of language containment are made.

3.1 Weak Alternating Parity Automata

Alternating parity automata allow both universal and existential states which occur e.g. in Büchi automata. The intuition of a run of an automaton on a word is as follows: If the automaton reaches an existential state it has to follow one of the edges given in the transition function. If it reaches an universal state it has to follow all edges given in the transition function.

Here we do not distinguish between universal and existential states but generalise the idea with the help of positive boolean formulas in which "$\wedge$" indicates an universal branching and "$\vee$" indicates an existential branching.

Definition 4 Let Q be a set. The set $\mathcal{B}^+(Q)$ of *positive Boolean formulas* over Q is the smallest set which satisfies

1. $Q \subseteq \mathcal{B}^+(Q)$ and
2. if $\varphi, \psi \in \mathcal{B}^+(Q)$ then $\varphi \vee \psi \in \mathcal{B}^+(Q)$ and $\varphi \wedge \psi \in \mathcal{B}^+(Q)$.

Given $\varphi \in \mathcal{B}^+(Q)$ we inductively define $St(f)$ as $St(q) := \{q\}$ for each $q \in Q$, and $St(\psi_1 \vee \psi_2) = St(\psi_1 \wedge \psi_2) := St(\psi_1) \cup St(\psi_2)$.

A *simple* positive Boolean formula is of the form q, $p \vee q$ or $p \wedge q$ for some $p, q \in Q$.

Definition 5 Let Q be a set, $\varphi \in \mathcal{B}^+(Q)$. The set $Sub(\varphi)$ of *sub-formulas* of a is

1. $\{q\}$ if $\varphi = q$ for some $q \in Q$
2. $\{\varphi\} \cup Sub(\psi_1) \cup Sub(\psi_2)$ if $\varphi = \psi_1 \Theta \psi_2$ for some $\psi_1, \psi_2 \in \mathcal{B}^+(M)$ and $\Theta \in \{\vee, \wedge\}$

The word weak in the name WAPA refers to the accepting condition of this kind of automata. A WAPA accepts a word if the lowest priority on every path in the run of the automaton is even. It does not have to visit a state with an even priority (or a final state like in Büchi automata) infinitely often as in alternating parity automaton to accept a word. That is why acceptance is a reachability problem.

To obtain the expressive power of Büchi automata the universal branching is needed.

Definition 6 (WAPA) A *weak alternating parity automaton* (WAPA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Omega)$ where

- Q is a finite set (of states)
- Σ is a finite alphabet
- $q_0 \in Q$ is the initial state
- $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is the transition function
- $\Omega : Q \to \mathbb{N}$ maps the states to their priorities.

A run of $\mathcal{A}$ on a word $w = a_0 a_1 a_2 ... \in \Sigma^\omega$ is a tree whose vertices are labeled with elements of Q such that

- the root is labeled with q_0
- if a vertex v on level n is labeled with q then its sons on level $n+1$ form a minimal model of $\delta(q, a_n)$

A run is called *accepting* if the smallest priority of the labels on every path is even. $\mathcal{A}$ accepts w, denoted $w \in L(\mathcal{A})$, if there is an accepting run of $\mathcal{A}$ on w.

For every $q \in Q$ let $\mathcal{A}[q] := (Q, \Sigma, q, \delta, \Omega)$ be the WAPA that results from $\mathcal{A}$ by changing the starting state to q.

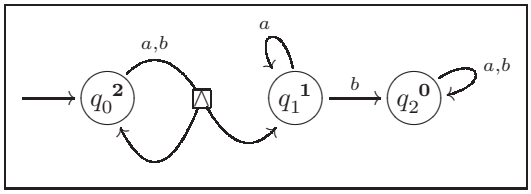

Figure 2: The WAPA given in example 2.

Finally, we define a relation $\models_{\mathcal{A}}$ between words $w \in \Sigma^\omega$ and positive Boolean formulas $\varphi \in \mathcal{B}^+(Q)$.

$$\begin{aligned} w \models q &\text{ iff } w \in L(\mathcal{A}\,[q]) \\ w \models \varphi \vee \psi &\text{ iff } w \models \varphi \text{ or } w \models \psi \\ w \models \varphi \wedge \psi &\text{ iff } w \models \varphi \text{ and } w \models \psi \end{aligned}$$

For the later discussion it is helpful to define a relation "$\leq$" on states of a given automaton.

Definition 7 Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Omega)$ be a WAPA and $p, q \in Q$. Write $p \leq q$ iff $L(\mathcal{A}\,[p]) \subseteq L(\mathcal{A}\,[q])$. Write $p \equiv q$ iff $p \leq q$ and $q \leq p$.

Here is an example of a simple WAPA.

Example 2 Let $\mathcal{A} = (\{q_0, q_1, q_2\}, \{a, b\}, q_0, \delta, \Omega)$ where

- $\delta(q_0, a) = \delta(q_0, b) = q_0 \wedge q_1$
- $\delta(q_1, a) = q_1$, $\delta(q_1, b) = q_2$
- $\delta(q_2, a) = \delta(q_2, b) = q_2$
- $\Omega(q_0) = 2$, $\Omega(q_1) = 1$, $\Omega(q_2) = 0$

$\mathcal{A}$ is shown in figure 2. It holds that $L(\mathcal{A}) = L(\mathcal{A}\,[q_0]) = \{w \mid w$ contains infinitely many b's $\} \subseteq L(\mathcal{A}\,[q_1]) = \{w \mid w$ contains at least one b $\}$ and that is why $q_0 \leq q_1$.

When a WAPA $\mathcal{A}$ visits a state q during a run on a word then the acceptance of the word depends on the priorities of the states that $\mathcal{A}$ will visit in the future as well as on the priorities of the states which $\mathcal{A}$ has already seen. This behaviour is impractical and defers from the behaviour of other (non-weak) automata like Büchi automata in which acceptance depends only on the infinite behaviour of the automaton. Fortunately such a behaviour could be achieved by a normalisation of weak parity automata. One just has to remember the lowest priority seen on a path, which can be encoded in the state space of the WAPA.

Definition 8 A WAPA $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Omega)$ is called *normalised* if for all $p, q \in Q$ and all $a \in \Sigma$: if $p \in St(\delta(q, a))$ then $\Omega(p) \leq \Omega(q)$.

$\mathcal{A}$ is called *simple* if for all $q \in Q$ and all $a \in \Sigma$: $\delta(q, a)$ is a simple positive Boolean formula.

Theorem 3 ([Löd00]) *For every WAPA $\mathcal{A}$ there is a simple and normalised WAPA $\mathcal{B}$ with $L(\mathcal{A}) = L(\mathcal{B})$ and $|\mathcal{B}| = O(|\mathcal{A}|^2)$.*

While normalisation is needed in the following notes on optimisation, simplicity is not. In the last section that deals with language containment of WAPA, normalisation is needed too.

3.2 Optimisation with Language Containment

In this part the following problem is analysed: Given a WAPA $\mathcal{A}$, and the relation "$\leq$" on the state space of $\mathcal{A}$. What algorithms can be used to reduce the complexity of the transition function and the cardinality of the state space efficiently without changing $L(\mathcal{A})$?

At first the problem of reducing the state space of an WAPA is discussed. The obvious way to reduce the state space is to *merge* two states q_1, q_2 of the WAPA into one state q_{12} so that every occurrence of q_1, q_2 on the right side of the transition function is replaced by q_{12} and $\delta(q_{12}) = \delta(q_1) \vee \delta(q_2)$.

One can show that there is no unique possibility to merge states with different priorities, i.e. the following procedures don't work:

- the new state receives the greater priority of the input states
- the new state receives the smaller priority of the input states
- the new state receives the even priority if one of the input states has an odd priority
- the new state receives the odd priority if one of the input states has an even priority
- the new state receives the smaller priority of the input states if both states are odd
- the new state receives the smaller priority of the input states if both states are even

Since the acceptance of a WAPA depends on both, the previous states as well as the following states in a run, $p \equiv q$ cannot be a sound criterion for a merging of p and q. This is shown by the following counterexample.

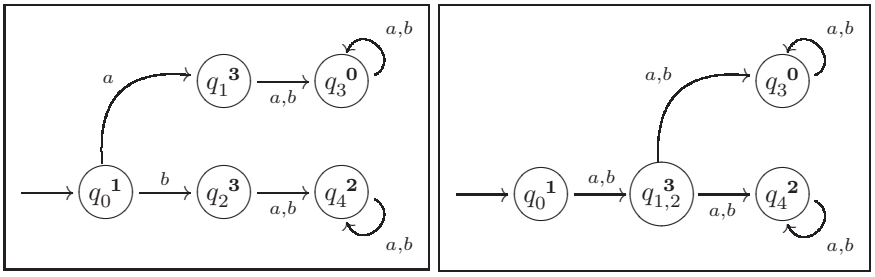

Figure 3: Example 3 before and after the merging of q_1 and q_2.

Example 3 Let $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, q_0, \delta, \Omega)$ be a WAPA with

- $\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_2$
- $\delta(q_1, a) = \delta(q_1, b) = q_3$
- $\delta(q_2, a) = \delta(q_2, b) = q_4$
- $\delta(q_3, a) = \delta(q_3, b) = q_3$
- $\delta(q_4, a) = \delta(q_4, b) = q_4$
- $\Omega(q_0) = 1$, $\Omega(q_1) = 3$, $\Omega(q_2) = 3$, $\Omega(q_3) = 0$, $\Omega(q_4) = 2$

It holds that $q_1 \equiv q_2$ since $L(\mathcal{A}[q_1]) = L(\mathcal{A}[q_2]) = \{a, b\}^\omega$. But merging q_1 and q_2 delivers a WAPA $\mathcal{B}$ with $L(\mathcal{B}) = \{a, b\}^\omega \neq L(\mathcal{A}) = \{\ aw \mid w \in \{a, b\}^\omega\ \}$ as shown in Figure 3.

It seems to be clear that the problem does not occur in normalised WAPA. But the equivalence criterion is not sound for normalised WAPA either as the next counterexample shows.

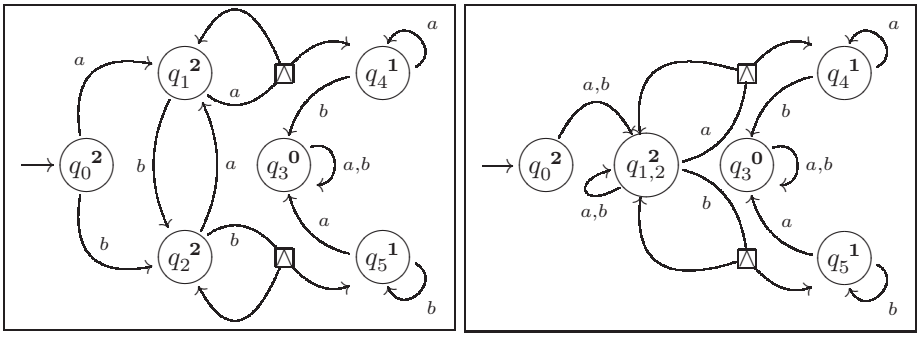

Figure 4: Example 4 before and after the merging of q_1 and q_2.

Example 4 Let $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b\}, q_0, \delta, \Omega)$ be a WAPA with

- $\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_2$
- $\delta(q_1, a) = q_1 \wedge q_4$, $\delta(q_1, b) = q_2$
- $\delta(q_2, a) = q_1$, $\delta(q_2, b) = q_2 \wedge q_5$
- $\delta(q_3, a) = \delta(q_3, b) = q_3$
- $\delta(q_4, a) = q_4$, $\delta(q_4, b) = q_3$
- $\delta(q_5, a) = q_3$, $\delta(q_5, b) = q_5$
- $\Omega(q_0) = 2$, $\Omega(q_1) = 2$, $\Omega(q_2) = 2$, $\Omega(q_3) = 0$, $\Omega(q_4) = 1$, $\Omega(q_5) = 1$

Here is $q_1 \equiv q_2$ because $L(\mathcal{A}[q_1]) = L(\mathcal{A}[q_2]) = L(\mathcal{A})$ and $L(\mathcal{A}) = \{w \mid w \neq b^\omega \text{ and } w \neq a^\omega\}$. But merging q_1 and q_2 delivers a WAPA $\mathcal{B}$ with $L(\mathcal{B}) = \{a, b\}^\omega \neq L(\mathcal{A})$ as shown in Figure 4.

One may have the idea to optimize the transition function of a normalised WAPA $\mathcal{A}$ in the following way: *If there are states q_1, q_2, q with $q_1 \leq q_2$ and a transition $\delta(q, \_) = q_1 \vee q_2$ in $\mathcal{A}$ then it can be optimised to $\delta(q, \_) = q_2$. If there are states q_1, q_2, q with $q_1 \leq q_2$ and a transition $\delta(q, \_) = q_1 \wedge q_2$ in $\mathcal{A}$ then it can be optimised to $\delta(q, \_) = q_1$.*

Here are counterexamples for both cases.

Example 5 Let $\mathcal{A} = (\{q_0, q_1, q_2, \}, \{a, b\}, q_0, \delta, \Omega)$ be a WAPA with

- $\delta(q_0, a) = q_0$, $\delta(q_0, b) = q_0 \vee q_1$
- $\delta(q_1, a) = q_2$, $\delta(q_1, b) = q_1$
- $\delta(q_2, a) = \delta(q_2, b) = q_2$
- $\Omega(q_0) = 3$, $\Omega(q_1) = 2$, $\Omega(q_2) = 1$

For $\mathcal{A}$ it holds that $q_1 \leq q_0$ because $L(\mathcal{A}[q_1]) = L(b^\omega) \subseteq L(\mathcal{A}[q_0]) = L(\mathcal{A}) = L((a|b)^* b^\omega)$. An optimisation of $\delta(q_0, b) = q_0 \vee q_1$ leads to a WAPA $\mathcal{B} = (\{q_0\}, \{a, b\}, \delta, \Omega)$ with $\delta(q_0, a) = \delta(q_0, b) = q_0$ and $\Omega(q_0) = 3$. That is why $L(\mathcal{B}) = \emptyset \neq L(\mathcal{A})$.

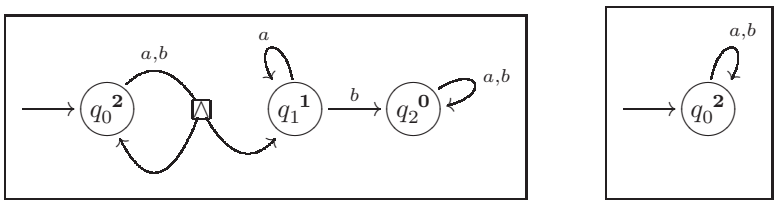

Figure 5: The WAPA $\mathcal{A}$ and $\mathcal{B}$ given in example 6.

Example 6 Let $\mathcal{A} = (\{q_0, q_1, q_2, \}, \{a, b\}, q_0, \delta, \Omega)$ be a WAPA with

- $\delta(q_0, a) = \delta(q_0, b) = q_0 \wedge q_1$
- $\delta(q_1, a) = q_1$, $\delta(q_1, b) = q_2$
- $\delta(q_2, a) = \delta(q_2, b) = q_2$
- $\Omega(q_0) = 2$, $\Omega(q_1) = 1$, $\Omega(q_2) = 0$

For that WAPA one have $L(\mathcal{A}[q_0]) = L(\mathcal{A}) = \{w \mid w$ contains infinitely many b's $\}$ and $L(\mathcal{A}[q_1]) = \{w \mid w$ contains at least one $b\}$. Therefore an optimisation of $\delta(q_o, b) = q_0 \wedge q_1$ leads to a WAPA $\mathcal{B} = (\{q_0\}, \{a, b\}, \delta, \Omega)$ with $\delta(q_0, a) = \delta(q_0, b) = q_0$ and $\Omega(q_0) = 2$. That is why $L(\mathcal{B}) = \{\{a, b\}^\omega\} \neq L(\mathcal{A})$. See Figure 5.

4 Simulation Games

In this section two different algorithms that compute the relation "$\leq$" on the state space of a given WAPA are presented. Both reduce the problem to the problem of computing winning set in weak parity games and use the algorithm introduced in section 2.

4.1 Small Simulation Game

With the small simulation game one can not compute "$\leq$" as defined above but a relation $\leq_s \subseteq \leq$. As tests have shown in practice it often holds that nearly $\leq_s = \leq$. The size of the weak Büchi game obtained from a given WAPA $\mathcal{A}$ is polynomial in the size of $\mathcal{A}$. That is why the algorithm that computes $\leq_s$ has a polynomial running time at all.

The intuition behind the small simulation game is the following: the game is played by two players on two copies of $\mathcal{A}$. One player tries to show that the first copy does not accept a word that the second copy accepts while the other play tries to prevent this.

$$(\forall_1)\ \frac{(f_1 \vee f_2, g, n, m)}{(f_i, g, n, m)}\ \forall i \in \{1,2\} \qquad (\forall_2)\ \frac{(f, g_1 \wedge g_2, n, m)}{(f, g_i, n, m)}\ \forall i \in \{1,2\}$$

$$(\forall_3)\ \frac{(q_1, q_2, n, m)}{(\delta(q_1, a), \delta(q_2, a), n, m)}\ \forall a \in \Sigma$$

$$(\exists_1)\ \frac{(q, g_1 \vee g_2, n, m)}{(q, g_i, n, m)}\ \forall i \in \{1,2\} \qquad (\exists_2)\ \frac{(f_1 \wedge f_2, q, n, m)}{(f_i, q, n, m)}\ \forall i \in \{1,2\}$$

$$(\exists_3)\ \frac{(f_1 \wedge f_2, g_1 \vee g_2, n, m)}{(f_i, g_1 \vee g_2, n, m)}\ \forall i \in \{1,2\}$$

Figure 6: The rules of the small simulation game.

Definition 9 (Small Simulation Game) Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Omega)$ be a WAPA. Define $\mathcal{B}^+(\mathcal{A}) = \{f \mid f \in Sub(g), g \in Im(\delta)\} \cup \{q_0^0, q_0^1\}$. The *small simulation game* for $\mathcal{A}$ and $p, q \in Q$ is defined as $\mathcal{SG}_A(p,q) = (V, E, p, (V_0, V_1))$, where

- $V = \mathcal{B}^+(\mathcal{A})^2 \times Im(p)^2$
- the edges E are given by the game rules in figure 6.
- $p(f, g, n, m) = (m \cdot (1 - n)) \bmod 2$
- $V_0 = \{\, v \mid v$ is of the form $(f_1 \wedge f_2,\ g_1 \vee g_2,\ n,\ m)$, $(f_1 \wedge f_2,\ q,\ n,\ m)$ or $(q,\ g_1 \vee g_2,\ n,\ m)$ for some $q \in Q, f_i,\ g_i \in \mathcal{B}^+(\mathcal{A}),\ n, m \in Im(p)\ \}$
- $V_1 = V - V_0$

Now one can prove that winning positions in simulation games for WAPA induce language containment of this WAPA in the following way.

Theorem 4 *Let $\mathcal{A}_0 = (Q, \Sigma, q_0, \delta, \Omega)$ be WAPA and $p, q \in Q$. Then it holds: if P_0 has a winning strategy from position $(p, q, \Omega(p), \Omega(q))$ in $\mathcal{SG}_A(p, q)$ then $p \leq q$.*

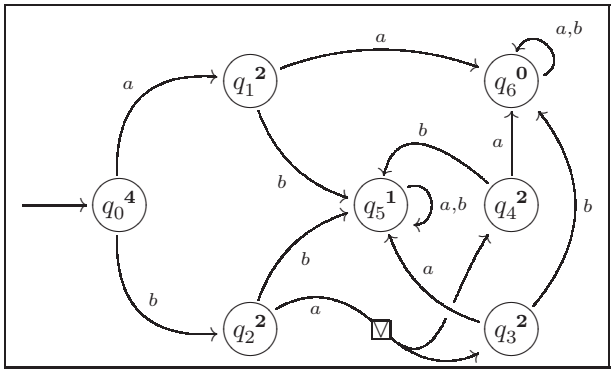

Figure 7: The Small Simulation Game cannot simulate q_1 with q_2.

A very similar result is shown in [Fri05, Fri02]. See there for a proof.

The following example shows that the other direction of Theorem 4 does not hold. That is why the algorithm for language containment which uses the small game is not complete.

Example 7 Let $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{a, b\}, q_0, \delta, \Omega)$ be a WAPA with

- $\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_2$
- $\delta(q_1, a) = q_6$, $\delta(q_1, b) = q_5$
- $\delta(q_2, a) = q_6$, $\delta(q_2, b) = q_3 \vee q_4$
- $\delta(q_3, a) = q_5$, $\delta(q_3, b) = q_6$
- $\delta(q_4, a) = q_5$, $\delta(q_4, b) = q_6$
- $\delta(q_5, a) = \delta(q_5, b) = q_5$
- $\delta(q_6, a) = \delta(q_6, b) = q_6$
- $\Omega(q_0) = 4$, $\Omega(q_1) = \Omega(q_2) = \Omega(q_3) = \Omega(q_4) = 2$, $\Omega(q_5) = 1$, $\Omega(q_6) = 0$

$\mathcal{A}$ is shown in Fig. 7. It is $q_1 \equiv q_2$ because $L(\mathcal{A}[q_1]) = L(\mathcal{A}[q_2]) = \{aw \mid w \in \{a, b\}^\omega\}$. Figure 8 shows the part of $\mathcal{SG}_A(q_1, q_2)$ that is reachable from $(q_1, q_2, 2, 2)$. As one can see there player P_1 has a winning strategy from the vertex $(q_1, q_2, 2, 2)$. But nevertheless it holds that $q_1 \leq q_2$.

Together with Theorem 2, Theorem 4 delivers an $O(|\Sigma| \cdot p^4 \cdot 2^{2b} \cdot n^{2b})$ time optimization algorithm for WAPA where p is the number of priorities, n is the number of states and b is the maximum size of boolean formulas that occur in the transition function. If f is the number of sub-formulas that occur in the image of the transition function one can also bound the running time by $O(|\Sigma| \cdot p^4 \cdot f^4)$ (note that i.g. $f \gg$ #states).

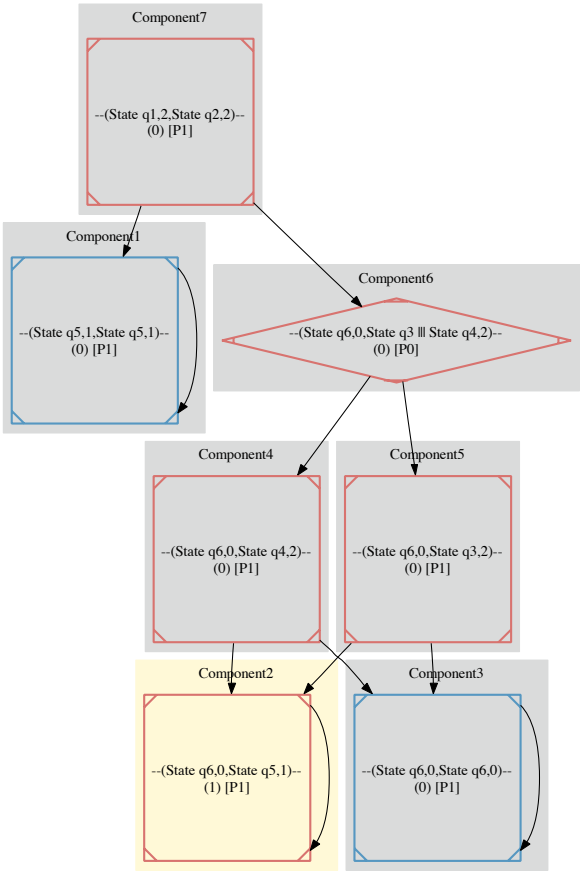

Figure 8: A Part of the small simulation game form the WAPA in figure 7.

4.2 Complete Simulation Game

The complete simulation for a given WAPA $\mathcal{A}$ can compute exactly the relation "$\leq$" on the state space.

Again a weak Büchi game is defined for a $\mathcal{A}$. But this time the size of the game is exponential in the space of the WAPA. All in all one receives a PSPACE algorithm for language containment.

Definition 10 Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Omega)$ be a normalised WAPA. The *complete simulation game* $\mathcal{CG}_{\mathcal{A}}(p, q)$ is played between players $\exists$ and $\forall$ on the game board $\mathcal{P}(Q)^2$. A configuration C is written

$$p_1, \ldots, p_n \vdash q_1, \ldots, q_m$$

and its intended meaning is: for all $w \in \Sigma^\omega$: $w \models \bigwedge_{i=1}^{n} p_i$ implies $w \models \bigvee_{j=1}^{m} q_j$. Its size, $|C|$ is $n + m$ in this case.

Every play in $\mathcal{CG}_{\mathcal{A}}(p, q)$ starts in the configuration $p \vdash q$. On every game position player $\forall$ chooses the vertex to visit next, i.e. $V_0 = V$ and $V_1 = \emptyset$. There

exist an edge between $p_1, \ldots, p_n \vdash q_1, \ldots, q_m$ and $\Phi \vdash \Psi$ if there exists an $a \in \Sigma$ so that

$$\delta(p_1, a), \ldots, \delta(p_n, a) \vdash \delta(q_1, a), \ldots, \delta(q_m, a) \Longrightarrow^* \Phi \vdash \Psi$$

for the simplify relation $\Longrightarrow$ that is defined by the following rules.

$$\begin{array}{lcll}
\varphi_1 \vee \varphi_2, \Phi \vdash \Psi & \Longrightarrow & \varphi_i, \Phi \vdash \Psi & \forall i \in \{1,2\} \\
\Phi \vdash \psi_1 \vee \psi_2, \Psi & \Longrightarrow & \Phi \vdash \psi_1, \psi_2, \Psi & \\
\Phi \vdash \psi_1 \wedge \psi_2, \Psi & \Longrightarrow & \Phi \vdash \psi_i, \Psi & \forall i \in \{1,2\} \\
\varphi_1 \wedge \varphi_2, \Phi \vdash \Psi & \Longrightarrow & \varphi_1, \varphi_2, \Phi \vdash \Psi &
\end{array}$$

The *priority* of a configuration $C = \Phi \vdash \Psi$ is defined inductively over the structure of positive Boolean formulas as follows.

$$\begin{array}{rcl}
val(q) & := & \begin{cases} 0, & \text{if } \Omega(q) \text{ even} \\ 1, & \text{o.w.} \end{cases} \\
val(\varphi \vee \psi) & := & \min\{val(\varphi), val(\psi)\} \\
val(\varphi \wedge \psi) & := & \max\{val(\varphi), val(\psi)\} \\
val(C) & := & \begin{cases} 0, & \text{if } val(\bigwedge \Phi) = 1 \text{ or } val(\bigvee \Psi) = 0 \\ 1, & \text{o.w.} \end{cases}
\end{array}$$

Player $\exists$ wins the play $C_0, C_1, \ldots$ if there is a $k \in \mathbb{N}$, s.t. for all $i \geq k$: $val(C_i) = 0$. Player $\forall$ wins the play $C_0, C_1, \ldots$ if there is a $k \in \mathbb{N}$, s.t. for all $i \geq k$: $val(C_i) = 1$.

We say that player p *wins* the game $\mathcal{CG}_{\mathcal{A}}(p, q)$ if she has a winning strategy for this game.

As shown in section 2 every play has a unique winner.

Remark 1 Let $\mathcal{A}$ be a normalised WAPA. Player $\exists$ wins the game $\mathcal{CG}_{\mathcal{A}}(p, q)$ iff $p \leq q$.

This theorem is intuitive clear but the proof is technical and long and therefore not given here.

References

[Ete01] K. Etessami, R. Schuller, T. Wilke. *Fair Simulation Relations, parity games, and state space reduction for Büchi automata.* Automata, Languages and Programming (ICALP 2001), vol. 1877 of LNCS, pages 694-707, 2001.

[Hen97] T. Henziger, O. Kupferman, S. Rajamani. *Fair simulation.* Proc. of 9th Int. Conf. on Concurrence Theory, number 1243 in LNCS, pages 290-301. Springer-Verlag, 2000.

[Hen00] T. Henziger, S. Rajamani. *Fair bisimulation.* TACAS, 2000.

[Kup01] O. Kupferman, M. Y. Vardi. *Weak alternating automata are not that weak.* ACM Transactions on Compututional Logic, 2(3), pages 408-429, 2001.

[Jur00] M. Jurdzinski. *Small progress measures for solving parity games.* STACS 2000, 17th Symp. on Theoretical Aspects of Computer Science, volume 1770 of LNCS, pages 290-301, Springer-Verlag, 2000.

[Mos91] A. W. Mostowski. *Games with forbidden positions.* Technical Report 78, University of Gdansk, 1991.

[Cor01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms* MIT Press, pages 525-553, 2001.

[Eme93] E. A. Emerson, C. S. Jutla, A. P. Sistla. *On model-checking for fragments of μ-calculus.* Computer Aided Verification, 5th Intern. Conference, CAV'93, vol. 697 of LNCS, pages 385-963, Springer-Verlag, 1993.

[Büc62] J. R. Büchi. *On a decision method in restricted second order arithmetic.* In Proc. of Congress on Logic, Method, and Philosophy of Science, pages 1-12, Stanford University Press, 1962.

[Löd00] C. Löding, W. Thomas. *Alternating automata and logics over infinite words.* In Prac. IFIP Int. Conf. on Theoretical Computer Scimes, IFIP TCS2000, vol. 1872 of LNCS, pages 521-534, Springer-Verlag, 2000.

[Mul86] D. E. Muller, A. Saoudi, P. E. Schupp. *Alternating automata, the weak monadic theory of the tree and its complexity.* In Proc. of 13th ICALP, LNCS 227, pages 275-283, 1986.

[Sis87] A. P. Sistla, M. Y. Vardi, P. Wolper. *The Complementation Problem for Büchi Automata with Applications to Temporal Logic.* Theoretical Computer Science, vol. 49, pages 217-237, 1987.

[Eme91] E. A. Emerson, C. S. Jutla. *Tree Automata, μ-Calculus and Determinacy.* In Proc. of 32nd Symp. on Foundations of Computer Science, pages, 368-377, IEEE, 1991.

[Lan04] M. Lange. *Weak Automata for the Linear Time μ-Calculus* In Proc.of 6th Int. Conf. on Verification, Model Checking and Abstract Interpretation, VMCAI'05, vol 3385 of LNCS, pages 267-281, 2005.

[Hop02] J. Hopcroft, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1980.

[Fri05] C. Fritz, T. Wilke. *Simulation Relations for Alternating Büchi Automata.* TCS 1-3, vol 338, 2005.

[Fri02] C. Firtz, T. Wilke. *State Space Reductions for Alternating Büchi Automata.* In Proc. of 22nd Int. Conf. on Foundations of Software, vol. 2556 of LNCS, pages 157-168, 2002.

[μSabre] *http://www.tcs.ifi.lmu.de/musabre.*