Efficient State Space Reduction for Automata by Fair Simulation

Jin Yi[1,2] and Wenhui Zhang[1]

[1] Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] Graduate University of the Chinese Academy of Sciences, Beijing, China
yijin,zwh@ios.ac.cn

Abstract. State space reduction for automata is important to automata-theoretic model checking. Optimizing automata by simulation relation is a practical method. There are several simulation concepts for Büchi automata, among which fair simulation has the advantage that it allows a larger number of pairs of states to be related and merged. We propose an approach to simplify Büchi automata by fair simulation method, which is based on integrating the method of [GBS02] and conditions of [SB00]. The approach can optimize an automaton without changing the language of each state and apply the optimization immediately after finding one pair of states with fair simulation equivalence. The experimental result shows our approach needs less time and reduces more states and transitions than that of the method [GBS02].

1 Introduction

State space reduction for automata is important to automata-theoretic model checking. Such model checking approaches are based on checking the language emptiness for an automaton which is the product of two automata, one for modeling the system, the other for the negation of the property of the system.

For improving the efficiency of model checking, much research is devoted to the optimization of automata. For LTL model checking, there are two possibilities, one is to simplify the property automaton transformed from a formula or provided by an user, the other is to optimize the transformation procedure from an LTL formula to an automaton [GO01]. Simulation relation is a preorder relation of states. It implies language containment and can be computed in polynomial time, therefore most of these optimizations use the simulation method.

The general simulation notion for LTS (Label Transition System) in [Mil89] has been studied thoroughly. If simulation relation considers the automaton's accepting condition, we get the following simulation concepts: direct simulation [DHW91], delayed simulation [EWS05] and fair simulation [HKR97]. These simulations can be used to reduce state space of generalized Büchi automata [JP06], Büchi automata [SB00,EWS05,GBS02,Ete02,BG04], and alternating automata [FW02,Fri03].

Among these concepts, direct simulation is the most restrictive. Given two states p, q of an automaton A, it requires that p simulates q, and if the i-th state of a path from q is an accepting state, then the i-th state of p's matching path is also an accepting state. So it can be used to reduce states and transitions safely [EH00,SB00]. That is, if p and q directly simulate each other, merging these two states does not affect the language of A. Moreover if q is directly simulated by p and they can be reached by a predecessor s with the same label, then the transition from s to q can be deleted without changing the language of A.

Delayed simulation relaxes a little restriction on the acceptance condition: if the i-th state of a path from q is accepting, then there is a $j \geq i$ such that the j-th state of p's matching path is accepting. Therefore we can find more pairs of states with delayed simulation relation, and merge them safely. However, it does not guarantee that we can delete transitions without changing the original language.

Fair simulation has the least restriction on the acceptance condition: if the path from q has infinitely many states which are accepting, then the matching path from p is also has infinitely many states in accepting set. Although fair simulation by itself is not a sufficient condition for collapsing two states or deleting one transition without affecting the original language, the experiment in [GBS02] shows that given a Büchi automaton, there are more pairs of states with fair simulation than that of direct or delayed simulation, and according them, we can reduce states and transitions safely.

There are two approaches to exploit the full power of fair simulation of Büchi automata. [SB00] provided conditions to delete states and transitions based on the language containment relation. Since fair simulation implies language containment, we can reduce the state space by fair simulation with the additionally given conditions. [GBS02] provided an approach to find all pairs with fair simulation first, then merge states or delete transitions based on these pairs, and check the correctness of the optimization. That is, it checks whether the new reduced automaton A' has the same language as that of the original A. Since computing language containment is PSPACE-complete problem [DHW91], and fair simulation implies language containment, [GBS02] checked $L(A) = L(A')$ by fair simulation.

In our paper, we also use fair simulation to minimize Büchi automata, the goal is to efficiently reduce the number of states and transitions by fair simulation. We propose an approach based on integrating the method of [GBS02] and conditions of [SB00], such that each state in the reduced automaton has the same language as that of the corresponding state in the original. Moreover, we can apply the optimization technique immediately after finding a candidate which is a pair of states with fair simulation. Since we find each candidate from the reduced automaton and may find a candidate which belongs to the reduced automaton but not to the original, thus our approach has less time and better performance than the method which needs finding all candidates before applying optimization.

Section 2 is the background knowledge. In Section 3, in order not to change the language of each state after optimizing, we restrict the criteria to judge the

correctness of each optimizing operator. We also prove that using conditions in [SB00] to reduce the state space of automaton can also preserve the language of each state. Moreover, we show that based on above two methods, we can apply the optimization without finding all candidates first. Section 4 describes the whole optimizing procedure. In Section 5, the experimental result shows the efficiency of our approach. Concluding remarks are given in Section 6.

2 Preliminaries

2.1 Büchi Automaton

A Büchi automaton is a tuple $A = \langle Q, \Sigma, s_0, \Delta, F\rangle$ where Q is a finite set of states, Σ is the input alphabet, $s_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation. We use $\delta : Q \times \Sigma \to 2^Q$ to denote the transition function defined by: $q \in \delta(q', a)$ iff $(q, a, q') \in \Delta$. A run of A on an infinite word $\alpha = \alpha(0)\alpha(1)\cdots$ is a sequence $r = r(0)r(1)\cdots$ such that $r(0) = s_0$, and for every $i \geq 0$, $r(i+1) \in \delta(r(i), \alpha(i))$. Let $inf(r)$ be the set of states that r visits infinitely often. If $inf(r) \cap F \neq \emptyset$, then the run r is an *accepting run* and α is in the language of A. The language of A is denoted by $L(A)$. For convenience, we write $q \in A$ for q being a reachable state of A. We write Q^A, Δ^A, F^A for respectively the set of states of A, the set of transition relations of A, and the set of accepting states of A. If $A = \langle Q, \Sigma, s_0, \Delta, F\rangle$ is an automaton and $q \in Q$, then $A[q]$ denotes the modified automaton $\langle Q, \Sigma, q, \Delta, F\rangle$. In our paper, the language of q of A means the language of $A[q]$.

2.2 Fair Simulation and Parity Game

Now we give the definition of fair simulation from the game perspective [EWS05]. Let $q_0 \in A$ and $q'_0 \in A'$, fair simulation game $G^f_{A,A'}(q_0, q'_0)$ is played by two players: Spoiler and Duplicator. At the first round, Spoiler puts a red pebble on q_0 while Duplicator puts a blue pebble on q'_0. Suppose in the ith round, Spoiler is in q_i, and moves the pebble to q_{i+1} according to $(q_i, \alpha_i, q_{i+1}) \in \Delta^A$, Duplicator must have a matching transition $(q'_i, \alpha_i, q'_{i+1}) \in \Delta^{A'}$ to move the pebble from q'_i to q'_{i+1}. If someone cannot move, then the game halts and the one who cannot move loses. Otherwise, there are two infinite paths $\pi = q_0...q_i...$ and $\pi' = q'_0...q'_i...$, we call (π, π') an outcome of the game. Then the outcome is winning for Duplicator iff there are infinitely many j in π' such that $q'_j \in F^{A'}$, or there are finite many i in π such that $q_i \in F^A$.

A strategy for Duplicator is a partial function $f : Q(Q'\Sigma Q)^* \to Q'$. It determines the next move of Duplicator according to the history of the play. That is, $f(q_0) = q'_0$ and $q'_j = f(q_0 q'_0 a_0 q_1 q'_1 a_1 \cdots a_{j-1} q_j)$ where $(q_i, a_i, q_{i+1}) \in \Delta^A$ and $(q'_i, a_i, q'_{i+1}) \in \Delta^{A'}$ for $i < j$. A strategy for Duplicator is a winning strategy if whenever $\pi = q_0 a_0 q_1 \cdots$ is a run of A and $\pi' = q'_0 a_0 q'_1 \cdots$ is a run defined by $q'_{i+1} = f(q_0 q'_0 a_0 q_1 q'_1 \cdots q_{i+1})$, then (π, π') is winning for Duplicator.

Definition 1. *[EWS05] Let $q \in A$ and $q' \in A'$, q' fair simulates q if there is a winning strategy for Duplicator in $G^f_{A,A'}(q, q')$. We denote such relation by $q \leq^f q'$.*

For convenience, we use $q \sim^f q'$ to denote $q' \leq^f q'$ and $q' \leq^f q$.

Proposition 1. *[EWS05] Let $q \in A$, $q_1 \in A_1$ and $q' \in A'$*

1. *if $q \leq^f q_1$ and $q_1 \leq^f q'$, then $q \leq^f q'$*
2. *if $q \leq^f q'$, then $L(A[q]) \subseteq L(A'[q'])$*

Definition 2. *[GTW02] A parity game graph is a tuple $G = \langle V, V_0, V_1, E, p\rangle$, where V is the set of vertexes, V_0 and V_1 are two disjoint sets such that $V = V_0 \cup V_1$, $E \subseteq V \times V$ is the set of edges, and p is a priority function that maps V to $\{0, ..., d-1\}$, $d \in \mathbf{N}$.*

A *play* of G is one of two kinds of paths: (1) An infinite path $\pi = v_0 v_1 \cdots \in V^\omega$ with $(v_i, v_{i+1}) \in E$ for all $i \in \omega$ (*infinite play*). (2) A finite path $\pi = v_0 v_1 \cdots v_l \in V^*$ for all $i < l$, $(v_i, v_{i+1}) \in E$ (*finite play*).

There are two players: One and Zero in G. At the beginning, Zero puts a pebble on v_0, then the players play the game according to the following rule: if the pebble is currently on v_i, and $v_i \in V_0(V_1)$, Zero(One) moves the pebble to the next position v_{i+1}, where $(v_i, v_{i+1}) \in E$. The *winning conditions* which judge the winner of a play are: (1) If a player cannot move the pebble, then this player loses and this is a finitely play in G. (2) Otherwise, there is an infinite path $\pi = v_0 v_1 \cdots$ in G. We denote by $inf(\pi)$ the set of vertexes that appear infinitely in π, and $\forall v_i \in inf(\pi)$, $p(v_i)$ is the priority of v_i. Let k_π be the minimal number of all $p(v_i)$. Then Zero wins the play if k_π is even, whereas One wins if k_π is odd.

A strategy for Zero is a partial function $f : V^* \times V_0 \to V$, given the history of the game up to a certain point, it determines the next move of Zero. In fact, we can get a strategy f which is memoryless strategy for Zero [GTW02], f selects the next move without looking at the history ($f : V_0 \to V$). A play $\pi = v_0 v_1 ...$ is an f-conform play if whenever $v_i \in V_0$ we have $v_{i+1} = f(v_0 ... v_i)$. The strategy f is winning on v if every f-conform play that starts in v is winning for Zero, and we call v a *winning state* for Zero. The set of winning states for Zero is called *wining region* for Zero. The winning region for Zero and One partition the set of vertexes of the game (Determinacy). So a state is winning for one player iff it is losing for the other. We can define the above concepts for One analogously.

We represent fair simulation by a parity game with three priorities.

Definition 3. *[EWS05] Given Büchi automata*

$$A_1 = \langle Q_1, \Sigma, s_1^0, \Delta_1, F_1\rangle \text{ and } A_2 = \langle Q_2, \Sigma, s_2^0, \Delta_2, F_2\rangle$$

we define a parity game graph $G_{A_1,A_2} = \langle V, V_0, V_1, E, p\rangle$, where

- $V_0 = \{v_{(q_1,q_2,a)} \mid q_1 \in Q_1, q_2 \in Q_2, a \in \Sigma \wedge \exists q_1''(q_1'', a, q_1) \in \Delta_1\}$

- $V_1 = \{v_{(q_1,q_2)} \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $E = \{(v_{(q_1,q_2)}, v_{(q_1',q_2,a)}) \mid (q_1, a, q_1') \in \Delta_1\}$
  $\cup \{(v_{(q_1,q_2,a)}, v_{(q_1,q_2')}) \mid (q_2, a, q_2') \in \Delta_2\}$
- $p(v) = \begin{cases} 0, \textit{ if } v = v_{(q_1,q_2)} \textit{ and } q_2 \in F_2 \\ 1, \textit{ if } v = v_{(q_1,q_2)}, q_1 \in F_1 \textit{ and } q_2 \notin F_2 \\ 2, \textit{ otherwise} \end{cases}$

According to Definition 3, we see that the game between One and Zero on G_{A_1,A_2} represents the fair simulation game play between Spoiler and Duplicator on $G^f_{A,A'}$. Therefore One represents Spoiler, and Zero represents Duplicator.

Jurdzinski's lifting algorithm [Jur00] can be used to solve the parity game on G_{A_1,A_2}. In the algorithm, each vertex is assigned a *progress measure* $\rho(v)$ where $\rho : V \to D$ and $D = \{0, ..., n_1\} \cup \{\infty\}$($n_1$ is the number of vertex v such that $v \in V_1$ and $p(v) = 1$). At the beginning of the algorithm, each vertex's measure is 0, then a "lift" operator will change vertex's measure until reaching a fixed point. Therefore, if $\rho(v) \neq \infty$, v has a winning strategy for Zero. [EWS05] has implemented the algorithm with complexity being $O(m \cdot (n_1 + 1))$ where m is the number of edges in G_{A_1,A_2}.

3 Reduction of States and Transitions

3.1 Motivation

[GBS02] provided an approach to reduce state space of Büchi automata by fair simulation. First it finds all pairs of states with fair simulation, which are candidates for merging states and deleting transitions. Second according to each candidate, it optimizes the automaton and checks whether the optimization is correct, i.e. whether there exists a fair simulation equivalence between the initial state of the reduced automata A' and that of the original automata A. One of important contributions of this approach is that it does not create a parity game graph $G_{A,A'}$ or $G_{A'A}$ for each candidate when checking the correctness of the optimization, $G_{A,A'}$ or $G_{A'A}$ is implemented by adding or deleting the edges from the game graph $G_{A,A}$. Therefore it is an efficient approach.

However, there exist two problems in this approach. The first one is that after merging two states or deleting one transition, it only guarantees that the language of the initial state of A' is the same as that of A, the language of other states of A' may not be the same as that of the states of A. Then we can't use this method to reduce the state space of a Büchi automaton as a model of some system, because we may need to verify the system from any given state, therefore we must preserve the langauge of every state after optimization.

The second is that it must find all candidates in advance. However, the method that simplifies the automaton as soon as a candidate is detected is more efficient, because if merging or deleting successes at this time, then at the next time, we can find a candidate from a new Büchi automaton with fewer states or transitions. Moreover, we even find more candidates than that of [GBS02], since there may exist some candidates only belonging to the new reduced automaton.

In the rest part of this section, we will resolve these two problems respectively.

3.2 Merging and Deleting States

Given a pair of states with fair simulation equivalence, we may merge them after that correctness checking is successful [GBS02] or delete one state according to the condition in [SB00]. In this subsection, we first introduce a new criterion to judge the correctness of merging two states, base on this criterion, we can easily to prove the language of each state in the reduced automaton keeps unchanged. Then we prove that deleting one state directly by the condition in [SB00] also has this property. Additionally, the new reduced automaton constructed by above two methods preserves the fair simulation relation of the original automaton.

Merging Two States For convenience, we use $[s_1, s_2]$ to denote a new state in A_m which is an automaton created by merging s_1 and s_2, where (s_1, s_2) is a pair in the fair simulation equivalence relation of A. The following is the formal definition of A_m for merging two states.

Definition 4. *Given a Büchi automaton* $A = \langle \Sigma, Q, s_0, \Delta, F \rangle$*, if* $\exists s_1, s_2$ *such that* $s_1 \sim^f s_2$*. Then* $A_m = \langle \Sigma, Q', s'_0, \Delta', F' \rangle$ *with*

- $Q' = Q \setminus \{s_1, s_2\} \cup \{[s_1, s_2]\}$*;*
- $s'_0 = [s_1, s_2]$ *if* $s_1 = s_0$ *or* $s_2 = s_0$*, otherwise* $s'_0 = s_0$*;*
- $\Delta' = \{(s, a, s') \mid (s, a, s') \in \Delta \wedge s, s' \neq s_i\}$
  $\cup \{([s_1, s_2], a, s') \mid (s_i, a, s') \in \Delta \wedge s' \neq s_i\}$
  $\cup \{(s, a, [s_1, s_2]) \mid (s, a, s_i) \in \Delta \wedge s \neq s_i\}$
  $\cup \{([s_1, s_2], a, [s_1, s_2]) \mid (s_i, a, s_i) \in \Delta\}$ *where* $i = 1, 2$*;*
- $F' = F$ *if none of* s_1 *and* s_2 *is in* F*, otherwise* $F' = F \setminus \{s_1, s_2\} \cup \{[s_1, s_2]\}$.

The above definition means that $[s_1, s_2]$ replaces s_1 and s_2, and all transitions that point to s_1 and s_2 are redirected to $[s_1, s_2]$, and all outgoing transitions of these two stats are also outgoing transitions of $[s_1, s_2]$. Therefore it may form some new cycles in A_m when s_1 reaches s_2 or s_2 reaches s_1 in A. So if there exists an accepting state in these new cycles, then $L(A[s_2]) = L(A[s_1]) \subseteq L(A_m[[s_1, s_2]])$ and the inclusion may be a proper one. Thus the other state's language in A_m which can reach $[s_1, s_2]$ may be changed even $L(A_m) = L(A)$. Therefore, in order to keep the language of remaining state unchanged, we need compute whether $[s_1, s_2]$ is fair simulated by s_1 (or s_2). Since a state s may belong to A, A_m and A_d (built by deleting states or transitions from A) at the same time, for convenience, we use s_A, s_{A_m} and s_{A_d} to denote $s \in A$, $s \in A_m$ and $s \in A_d$ respectively.

Proposition 2. *If* $[s_1, s_2]_{A_m}$ *is fair simulated by* s_1 *of* A*, then* $s_{A_m} \sim^f s_A$.

Proof. By the definition of A_m, $s_A \leq^f s_{A_m}$ is obvious. Now we prove $s_{A_m} \leq^f s_A$. We construct a strategy f for Duplicator on $G^f_{A_m,A}(s, s)$. Let $f(s) = s$ and if $s_i \neq [s_1, s_2]$ then $f(ss\alpha_0 \cdots s_i) = s_i$, which means that if s_{A_m} does not reach $[s_1, s_2]$, then the state chosen by Duplicator is identical with the state chosen by Spoiler. Otherwise, if s_{A_m} reaches $[s_1, s_2]$, then s_A reaches s_1 or s_2. Since $[s_1, s_2]_{A_m} \leq^f$

s_1 and $s_1 \sim^f s_2$, thus there exists a winning strategy f' for Duplicator on $G^f_{A_m,A}([s_1,s_2],s_1)$ or $G^f_{A_m,A}([s_1,s_2],s_2)$, then $f(ss\alpha_0\cdots[s_1,s_2]s_1\cdots\alpha_{i-1}s_i) = f'([s_1,s_2]s_1\cdots\alpha_{i-1}s_i)$, or $f(ss\alpha_0\cdots[s_1,s_2]s_2\cdots\alpha_{i-1}s_i) = f'([s_1,s_2]s_2\cdots\alpha_{i-1}s_i)$ which means the next step of Duplicator is decided by f', which needs the history started from $[s_1,s_2]$. Moreover, since the state chosen by Duplicator is identical with the state chosen by Spoiler or is decided by the winning strategy f', so f is a winning strategy for Duplicator. Therefore$s_{A_m} \leq^f s_A$. □

Therefore, whether $[s_1,s_2]_{A_m}$ is fair simulated by s_1 of A decides the correctness to merge s_1 and s_2. While [GBS02] decide the correctness by whether the initial state of A_m is fair simulated by that of A. Since there may be a situation that two states is merged in [GBS02] but not in our approach, so we restrict the condition to merge states.

Now we introduce an important property of A_m, it shows that A_m preserves the fair simulation relation of A.

Proposition 3. *If $[s_1,s_2]_{A_m}$ is fair simulated by s_1 of A, then $s_A \leq^f q_A$ iff $s_{A_m} \leq^f q_{A_m}$.*

Proof. We first prove if $s_A \leq^f q_A$ then $s_{A_m} \leq^f q_{A_m}$. By Proposition 2, we know $s_{A_m} \leq^f s_A$, then since $s_A \leq^f q_A$, by the transitivity of fair simulation, $s_{A_m} \leq^f q_A$. By Proposition 2, $q_A \leq^f q_{A_m}$, then by the transitivity again, $s_{A_m} \leq^f q_{A_m}$. The other direction can be proved by the same way. □

Deleting One State Given two states with the same language, if they do not reach each other, then we can get a reduced automaton A_d by deleting one state and it's transitions [SB00]. Since fair simulation equivalence implies language equivalence, thus given a candidate with fair simulation equivalence, we may delete one state and it' transitions. It is easy to see the method is efficient for it does not check the correctness and deletes states and transitions at the same time, so after finding a candidate, we first try to use this method to delete one state directly, only when it fails, we use the method described above to merge two states.

Note that [SB00] just proved that $L(A_d) = L(A)$. Now we show that if two states with the same language is judged by fair simulation equivalence, then A_d has two important properties: (1) s_{A_d} and s_A fair simulate each other, therefore any state of A_d has the same language as that of the corresponding state in A. (2) A_d preserves the fair simulation relation of A.

Definition 5. *Given a Büchi automaton $A = \langle \Sigma, Q, s_0, \Delta, F\rangle$ with $s_1, s_2 \in Q$ such that $s_1 \sim^f s_2$ and s_1 can not reach s_2. Then $A_d = \langle \Sigma, Q', s'_0, \Delta', F\rangle$ with*

- $Q' = Q \setminus \{s_2\}$;
- $s'_0 = s_1$ *if* $s_2 = s_0$,*otherwise* $s'_0 = s_0$;
- $\Delta' = \{(s,a,s') \mid (s,a,s') \in \Delta \wedge s, s' \neq s_2\}$
  $\cup\ \{(s,a,s_1) \mid (s,a,s_2) \in \Delta \wedge s \neq s_2\}$.

A_d is constructed by deleting s_2 and it's transitions from A, and all transitions that point to s_2 are redirected to s_1.

Proposition 4. $s_{A_d} \sim^f s_A$

Proof. We first prove $s_A \leq^f s_{A_d}$. We construct a strategy f for Duplicator on $G^f_{A,A_d}(s,s)$. Let $f(s) = s$ and if $s_i \neq s_2$ then $f(ss\alpha_0 \cdots s_i) = s_i$, which means that if s_A does not reach s_2, then the state chosen by Duplicator is identical with the state chosen by Spoiler. Otherwise, if s_A reaches s_2, by the definition of A_d, s_{A_d} must reach s_1. Since $s_2 \leq^f s_1$ in A, there exists a winning strategy f' for Duplicator on $G^f_{A,A}(s_2,s_1)$. Moreover, since s_1 can not reach s_2, thus any path started from s_1 of A and confirmed with f' is also a path of A_d. So we can define $f(ss\alpha_0 \cdots s_2 s_1 \cdots \alpha_{i-1} s_i) = f'(s_2 s_1 \cdots \alpha_{i-1} s_i)$, which means if s_A reaches s_2, then the next step of Duplicator is decided by f'. Since the state chosen by Duplicator is identical with the state chosen by Spoiler or is decided by the winning strategy f', so f is a winning strategy for Duplicator. Therefore $s_A \leq^f s_{A_d}$.

Then we prove $s_{A_d} \leq^f s_A$. We construct a strategy f for Duplicator on $G^f_{A_d,A}(s,s)$. Let $f(s) = s$ and if s_{A_d} does not reach s_1 by $(s_{i-1}, \alpha_{i-1}, s_1) \notin \Delta^A$, then $f(ss\alpha_0 \cdots s_i) = s_i$, which means that the state chosen by Duplicator is identical with the state chosen by Spoiler. Otherwise, if s_{A_d} reaches s_1 by $(s_{i-1}, \alpha_{i-1}, s_i) \notin \Delta^A$ where $s_i = s_1$. By the definition of A_d, s_A reaches s_2 by $(s_{i-1}, \alpha_{i-1}, s'_i) \in \Delta^A$ where $s'_i = s_2$. Since $s_1 \leq^f s_2$ in A, then there exists a winning strategy f' for Duplicator on $G^f_{A,A}(s_1,s_2)$. Thus We define $f(ss\alpha_0 \cdots s_1 s_2 \cdots \alpha_{j-1} s_j) = f'(s_1 s_2 \cdots \alpha_{j-1} s_j)$.

Now we show the correctness of defining f according to f'. Since any path of A_d started from $s_i (= s_1)$ has no such transition that points to s_1 and belongs to Δ^{A_d} but not Δ^A, this is because if we assume $(s_k, \alpha_k, s_1)(k > i)$ is the first such transition, then by the definition of A_d, there exists $(s'_k, \alpha_k, s_2) \in \Delta^A$, then we get a path from $s_i (= s_1)$ to s_2 in A, which contradicts with s_1 cannot reach s_2. Therefore, any path of A_d started from $s_i (= s_1)$ is also the path of A and has a corresponding path of A with respect to f'. Thus we can define $f(ss\alpha_0 \cdots s_1 s_2 \cdots \alpha_{j-1} s_j) = f'(s_1 s_2 \cdots \alpha_{j-1} s_j)$, which means the next step of Duplicator is decided by f'.

Since the state chosen by Duplicator is identical with the state chosen by Spoiler or is decided by the winning strategy f', so f is a winning strategy for Duplicator. Therefore $s_{A_d} \leq^f s_A$. □

By the transitivity of fair simulation, we have the following property of A_d.

Proposition 5. $s_A \leq^f q_A$ *iff* $s_{A_d} \leq^f q_{A_d}$.

3.3 Deleting Transitions

In this subsection, we provide a method to delete one transition like the section 3.2, i.e. it combines the method of [GBS02] with the condition in [SB00].

Given A and a pair of states (s, s') with $s \leq^f s'$(which implies $L(A[s]) \subseteq L(A[s'])$), we first find a state $p \in A$ such that $(p, a, s), (p, a, s') \in \Delta^A$, $a \in \Sigma^A$. Then if s' cannot reach p, we can construct A_d by deleting the transition (p, a, s) directly according to the condition of [SB00]. Since we judge language containment by fair simulation, so we strengthen the conclusion $L(A_d) = L(A)$ in [SB00] by the the following proposition, it shows that each state of A_d has the same language as that of A.

Proposition 6. *Given A and $s \leq^f s'$, let p be a state such that (p, a, s), $(p, a, s') \in \Delta^A$, if s' cannot reach p, then $t_{A_d} \sim^f t_A$, where A_d is constructed by deleting (p, a, s) from A.*

Proof. By the definition of A_d, it is easy to prove that $t_{A_d} \leq^f t_A$. Now we prove $t_A \leq^f t_{A_d}$. We construct a strategy f for Duplicator on $G^f_{A,A_d}(t, t)$. Let $f(t) = t$ and if t_A does not reach s by the transition $(p, a, s) \in \Delta^A$, then $f(tt\alpha_0 \cdots t_i) = t_i$, which means the state chosen by Duplicator is identical with the state chosen by Spoiler. Otherwise, if t_A reaches s by (p, a, s), according to the definition of A_d, t_{A_d} reaches s' by (p, a, s'). Since $s \leq^f s'$ in A, so there exists a winning strategy f' for Duplicator on $G^f_{A,A}(s, s')$. Moreover, since s' cannot reach p, any path of A that started from s' and confirmed with f' is also a path of A_d, therefore, we can define $f(tt\alpha_0 \cdots ss' \cdots \alpha_{i-1}t_i) = f'(ss' \cdots \alpha_{i-1}t_i)$, which means the next step of Duplicator is decided by f'. Since the state chosen by Duplicator is identical with the state chosen by Spoiler or is decided by the winning strategy f', so f is a winning strategy for Duplicator. Therefore $t_A \leq^f t_{A_d}$. □

However, if s' can reach p, we must check the correctness before deleting one transition. In order to make the reduced automaton preserve the language of each state of A, the following proposition shows that we need a more restricted criterion than that in [GBS02]. That is, our approach requires p_{A_d} fair simulates p_A, while [GBS02] require the initial state of A_d fair simulates the initial state of A.

Proposition 7. *Given A and $s \leq^f s'$, let p be a state such that (p, a, s), $(p, a, s') \in \Delta^A$, if $p_A \leq^f p_{A_d}$, then $t_{A_d} \sim^f t_A$, where A_d is constructed by deleting (p, a, s) from A.*

Therefore, when deleting transitions, Proposition 6 and 7 guarantee that our approach can resolve the first problem mentioned in Section 3.1.

3.4 Efficiently Finding Candidates

In [GBS02], we have to find all candidates before applying any optimization. That is, according to Definition 3, we first create a parity game $G_{A,A}$, where each vertex $v \in V_1$ corresponds to each pair of states in A, then after resolving the game by Jurdzinski's algorithm, we find all pairs with fair simulation.

Our approach aims at reducing states without finding all candidates in advance. As soon as we find one candidate, we try to delete one state or merging

two states by the method in Section 3.1. Thus in case this successes, in the next time, we find new candidates on the reduced automaton A_d or A_m, which needs less time than that on A. Moreover, we may find new candidates from A_m or A_d which only belong to the reduced automaton but not to A, thus we have more opportunities to reduce states and transitions than that in [GBS02].

Since we do not find all candidates from one parity game $G_{A,A}$, we have to build and resolve the parity game each time to find new candidates on the reduced automaton. In order to improve the efficiency of the whole optimization procedure, we reuse the information according to the property of parity game and Proposition 3 and 5.

Given a parity game G, if there exist some vertexes in G such that we know they are winning states for Zero or not, then we can delete edges of these vertexes and get a new parity game G', which has the same wining region for Zero as that of G. Since G' has fewer edges than that of G, thus computing G' needs less time than that of G. See the formal definition of G' and it's property in Appendix.

Now we describe how to reuse information in detail. Given a pair (s_1, s_2) and a Büchi automaton A^i which is built from A^{i-1} by merging states or deleting states according to the method in the section 3.2 (let $A^0 = A$). In order to check whether $s_1 \leq^f s_2$(i.e., whether (s_1, s_2) is a new candidate to be reduced), we create game G. By Propositions 1, 3 and 5, for any $0 \leq j < i$, A^i preserves the fair simulation relation of A^j. So if $s \leq^f s'$ is true in A^j, then $s \leq^f s'$ is true in A^i. Therefore, if $v = v_{(s,s')}$ is one of vertexes of G, then v is the winning state for One in G. Thus we need not compute v's measure when resolving G, for we can reuse the information from some game which computes $s \leq^f s'$ before building A^i. Therefore we save the time to computing fair simulation on G when we check the pair (s_1, s_2) on A^i. Moreover, by Propositions 3 and 5, we can also see that there may exist some pairs which are in the fair simulation relation of A^i but are not in the fair simulation relation of A, then we may reduce more states and transitions than that of the method [GBS02].

4 Algorithm

In this section, we describe the whole optimization procedure in detail. The procedure has two phases: first it tries to delete and merge the pair of stats with fair simulation equivalence, and then it tries to remove the transitions based on fair simulation.

In first stage, we check all pairs of states in A. Suppose the current pair is (s_1, s_2). If we do not know whether $s_1 \sim^f s_2$, we need construct two parity game graphs $G_{A[s_1],A[s_2]}$ and $G_{A[s_2],A[s_1]}$ and compute the progress measure using the Jurdzinski's algorithm, for any $v = v_{(q,q')} \in V_1$, if $\rho(v) \neq \infty$, then $q \leq^f q'$, otherwise $q \not\leq^f q'$. Therefore, if $s_1 \sim^f s_2$, we first check whether s_1 and s_2 can reach each other, if it is not, then we delete one of states and its transitions from A directly. Otherwise, we construct A_m from A by merging s_1 and s_2, and build a parity game $G_{A_m[[s_1,s_2]],A[s_1]}$ to check whether $[s_1, s_2]$ is fair simulated

by s_1. If the candidate (s_1, s_2) is safe, replace A by A_m. So in this stage, we do optimization after finding one suitable candidate. After finishing this stage, we find all pairs of stats in the fair simulation relation of the current A, which is the basis of the next stage.

Before beginning the second stage, we construct a parity game $G_{A,A}$. Then for each pair of A with fair simulation, we do the following work. Suppose the current pair is (s, s'). If $\exists p \in A$ such that $(p, a, s) \in \Delta^A$ and $(p, a, s') \in \Delta^A$, then we first check whether s' can reach p, if it is not, we delete the transition (p, a, s) from A directly. Otherwise, we delete some edges (v, v') from $G_{A,A}$ where $v = v_{(t,p,a)} \in V_0, v' = v_{(t,s)} \in V_1, t \in A$. In fact, now $G_{A,A}$ becomes G_{A,A_d}. Therefore, if the vertex $v = v_{(p,p)}$ of the new $G_{A,A}$ is a winning state for Zero which means $t_A \leq^f t_{A_d}$, then we delete (p, a, s) from A. Otherwise, we must restore the game $G_{A,A}$.

Although our algorithm has the same worst time complexity as that of [GBS02], since we use the conditions [SB00] which delete states and transitions without any checking, and we find candidates on the reduced automaton which has fewer and fewer states and transitions in the optimizing procedure, thus our algorithm needs less time than that of [GBS02]. Moreover, since we may find more candidates, we can reduce more states and transitions than that of [GBS02], we will show these in the next section.

5 Experiment

We have implemented our algorithm in C which runs on a PC with 3.2 GHz Intel Pentium 4 and 1G RAM. Büchi automata to be optimized are transformed from LTL formulae based on lbtt [TH02] and LTL2BA [GO01]. lbtt is a tool that provides an automated testing environment for LTL-to-Büchi translators. We use it to create random LTL formulae. LTL2BA is a fast and efficient tool to transform a LTL formula to a transition-labeled Büchi automaton.

In our algorithm(*ERSS* Efficiently Reduce State Space), we first use the method described in Section 3.2 to delete or merge states, moreover, we delete one state or merge two states as long as find out one candidate, which is described in Section 3.4. Then after find all candidates out and finish reducing states, we can delete transitions using the method in Section 3.3. For convenience, we use *ERSS*− to denote the algorithm such that it is same as *ERSS* except it can not delete states and transitions directly according to the conditions in [SB00].

We also have implemented the algorithm [GBS02], because the original implementation is based on Wring [SB00] by Perl, which translates a random LTL formula to a generalized state-labeled Büchi automaton, but the algorithm in [GBS02] can only optimize Büchi automata but not the generalized Büchi automata.

We have tested three group automata, each group has 200 Büchi automata. In the following tables, the column marked by *state* and *transition* shows the average number of states and transitions respectively and the column marked by

time is the computing time for the algorithm. The column *Original* is the data obtained using the original Büchi automaton without any optimization.

Table 1 shows the comparison between our algorithm and GBS02. We find ERSS can reduce more states and transitions and need less computing time than GBS02, moreover, these advantages become more obvious when increasing the automaton's states and transitions. Table 2 is the comparison between ERSS− and GBS02. It also shows the efficiency of our algorithm.

Table 1 and Table 2 both show that using conditions in [SB00] to reduce the state space is very efficient, which can delete more transitions and save more computing time.

Table 1. Comparison between ERSS and GBS02

Original		ERSS			GBS02		
state	transition	state	transition	time	state	transition	time
69.865	652.33	51.39	456.11	53.143	52.43	465.65	94.012
140.115	1897.35	96.11	1280.115	611.763	100.31	1329.94	1034.329
241.84	3849.315	146.195	2321.605	1637.018	158.7	2482.325	8872.937

Table 2. Comparison between ERSS− and GBS02

Original		ERSS−			GBS02		
state	transition	state	transition	time	state	transition	time
69.865	652.33	51.455	457.15	66.389	52.43	465.65	94.012
140.115	1897.35	96.71	1290.375	817.312	100.31	1329.94	1034.329
241.84	3849.315	147.955	2360.01	3816.767	158.7	2482.325	8872.937

6 Conclusion and Future Work

We have presented an algorithm that can efficiently reduce state space of Büchi automata. This approach is based on fair simulation relation of Büchi automata like the method in [GBS02], i.e. we use fair simulation to find candidates and check the correctness of optimization.

In our paper, based on [GBS02], we provide the restricted criteria to check the correctness of the optimization. At the same time, we use the conditions in [SB00] to delete states and transitions without any checking. In the whole optimization procedure, the language of each state of the reduced automaton is same as that of the corresponding state of the original one. According to this property, we reduce states without waiting for finding all candidates, thus we find each candidate in

a new reduced automaton with fewer states and transitions, furthermore, in the whole procedure, we may find more candidates from the reduced automaton than that from the original automaton. We have implemented our algorithm (ERSS) described in Section 3. The experimental result shows that ERSS needs less time and reduces more states and transitions than that of [GBS02].

Although we can resolve the fair simulation game in polynomial time, with the number of states and transitions increasing, the time to find candidates increases rapidly and even becomes intolerant. The further work is to research on methods for efficiently optimizing Büchi automata with large state spaces by fair simulation.

Acknowledgments

The Authors would like to thank Zhilin Wu and Xiang Zhou for their useful discussions about this work.

References

[BG04] D. Bustan, O. Grumberg: Applicability of fair simulation. Information and computation, Vol. 194(1), pp. 1-18, 2004.

[DHW91] D. L. Dill, A. J. Hu, and H.Wong-Toi. Checking for language inclusion using simulation preorders. CAV'91, vol. 575 of LNCS, pp. 255-265, 1991.

[EH00] K.Etessami and G. Holzmann. Optimizing Büchi automata. CONCUR'00, vol. 1877 of LNCS, pp. 153-167, 2000.

[Ete02] K. Etessami. A Hierarchy of Polynomial-Time Computable Simulations for Automata. CONCUR'02, vol. 2421 of LNCS, pp. 131-144, 2002.

[EWS05] K. Etessami, Th. Wilke, and R. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. SIAM Journal of Computing, 34(5): 1159–1175, 2005.

[Fri03] C.Fritz. Constructing Büchi Automata from Linear Temporal Logic Using Simulation Relations for Alternating Büchi Automata. CIAA'03, vol. 2759 of LNCS, pp. 35-48, 2003.

[FW02] C. Fritz, T. Wilke, State Space Reductions for Alternating Büchi Automata. FSTTCS 2002, vol. 2556 of LNCS, pp.157-168, 2002.

[GBS02] S. Gurumurthy, R. Bloem, F. Somenzi. Fair Simulation Minimization. CAV'02, vol. 2404 of LNCS, pp. 610-624, 2002.

[GTW02] E. Grädel, W. Thomas, and Th. Wilke, eds., Automata, Logics, and Infinite Games: A Guide to Current Research, vol. 2500 of LNCS, Springer-Verlag, New York, 2002.

[GKSV03] S.Gurumurthy, O.Kupferman, F.Somenzi, M.Y. Vardi. On Complementing Nondeterministic Büchi Automata. CHARME'03:96-110.

[GO01] P.Gastin and D.Oddoux, Fast LTL to Büchi Automata Translation CAV'01, vol. 2102 of LNCS, pp. 53-65, 2001.

[HKR97] T.A.Henzinger, O.Kupferman, and S.K. Rajamani. Fair simulation. CONCUR'97, vol. 1243 of LNCS, pp. 273-287, 1997.

[JP06] S.Juvekar, N. Piterman, Minimizing Generalized Buchi Automata CAV'06, vol. 2102 of LNCS, pp. 53-65, 2001.

[Jur00] M. Jurdzinski. Small progress measures for solving parity games. STACS'00, vol. 1770 of LNCS, pp. 290-301, 2000.
[Mil89] R.Milner. Communication and Concurrency. Prentice-Hall, 1989.
[SB00] F. Somenzi and R.Bloem. Efficient Büchi automata from LTL formulae. CAV'00, vol. 1855 of LNCS, pp. 248-263, 2000.
[TH02] H. Tauriainen and K. Heljanko, Testing LTL formula translation into Büchi automata, International Journal on Software Tools for Technology Transfer (STTT) 4(1):57-70, 2002.

A Property of Parity Game for Reusing Information

Definition 6. *Given a parity game graph $G = \langle V, V_0, V_1, E, p\rangle$, Let $W = W_0 \cup W_1$ where W_0 is the set of vertexes such that each vertex $v \in W_0$ is the wining state for Zero on G, and W_1 is the set of vertexes such that each vertex $v \in W_1$ is the wining state for One on G. Then $G' = \langle V, V_0', V_1', E', p\rangle$ is a parity game graph, with*

- $V_0' = V_0 \setminus W \cup W_1$;
- $V_1' = V_1 \setminus W \cup W_0$;
- $E' = \{(v, v') \mid (v, v') \in E \wedge v \in V \setminus W\}$.

Since we delete the edges of $v \in W$, by the definition of a play in parity game, there are some new finite plays in G' ended with $v \in W$. Let π be such play. If $v \in W_1$, which means v is the wining state for One on G, since $v \in V_0'$, thus *Zero* can not move on v of π and *Zero* loses. Similarly, If $v \in W_0$, which means v is the wining state for Zero on G, since $v \in V_1'$, thus *One* can not move on v in π and *One* loses. Then v is also a winning state for Zero(One) on G' as same as on G.

Now we establish the relation between G and G'. Let $[v^G]_{Zero}$ denote v of G that is the wining state for Zero, $[v^G]_{One}$ denote v of G that is the wining state for One.

Proposition 8. $[v^{G'}]_{Zero}$ *iff* $[v^G]_{Zero}$

Proof. If $v \in W$, by Definition 6, it is easy to see that $[v^{G'}]_{Zero}$ iff $[v^G]_{Zero}$. Otherwise, we first prove if $[v^{G'}]_{Zero}$ then $[v^G]_{Zero}$. We first construct a strategy f'' for Zero on v of G. Since $[v^{G'}]_{Zero}$, there exists a winning strategy f on v of G', if a play from v does not meet $v_i \in W$, then we define $f''(vv_1 \cdots v_i) = f(vv_1 \cdots v_i)(i \geq 1)$. If the play meets $v_i \in W$, since $[v^{G'}]_{Zero}$, thus $[v_i^{G'}]_{Zero}$, that is $[v_i^G]_{Zero}$. Therefore, there exists a winning strategy f' on v_i of G. So we define $f''(vv_1 \cdots v_i \cdots v_j) = f'(v_i \cdots v_j)$ which means after meeting v_i, the next move of Zero is decided by f'. Since f and f' are winning strategies for Zero, so it is easy to see that f'' is also a winning strategy for Zero on v of G.

Now we prove if $[v^G]_{Zero}$ then $[v^{G'}]_{Zero}$. We first construct a strategy f'' for Zero on v of G'. Since $[v^G]_{Zero}$, there exists a winning strategy f for Zero on v of G, if the play on v of G' does not meet $v_i \in W$, then we define $f''(vv_1 \cdots v_i) = f(vv_1 \cdots v_i)$. Otherwise, if $v_i \in W$, since $[v^G]_{Zero}$, so $[v_i^G]_{Zero}$, that is $[v_i^{G'}]_{Zero}$.

Therefore, since f is a winning strategy, thus for any f''-conform play from v which does not meet $v_i \in W$, Zero wins. Moreover, since $[v_i^G]_{Zero}$ thus $v_i \in V_1'$, then for any f''-conform finite play whose end is v_i, One cannot move and Zero wins. Therefore, f'' is a winning strategy for Zero on v of G'. □

Since parity game is determined, so we can easily get the following corollary according to Proposition 8.

Corollary 1. $[v^{G'}]_{One}$ *iff* $[v^G]_{One}$