Antichain Optimization using Simulation Relations for Context-Free Games

Fajar Haifani

August 1, 2017

Master's Thesis
Technical University of Kaiserslautern
Department of Computer Science

First Reviewer	:	Prof. Dr. Klaus Schneider
Second Reviewer	:	Prof. Dr. Roland Meyer
Supervisor	:	Sebastian Muskalla, M. Sc.

I certify that the content of this master thesis has not been submitted for any degree requirements. This is an original work and any sources have been properly acknowledged.

Fajar Haifani

Acknowledgement

Firstly, I would like to express my gratitute towards Prof. Roland Meyer as my mentor and thesis reviewer. I am grateful for being able to work on an interesting topic in his group. It has been a great experience not only to work on this thesis, but also to learn theoretical computer science in general from him.

I would also like to thank Prof. Klaus Schneider as the first reviewer.

A special thanks goes to Sebastian Muskalla, Msc as my supervisor for his guidance, advices, and patience while working with me. Despite being in a tight schedule, he has been able to make time to discuss my thesis.

I would like to thank Markus, Constantin, and Felix for their support with the implementation.

I would also like to thank my parents and family for their supports and prayers.

Lastly, it is not at all possible for this thesis to get to this present state without the help of many people whom I can not mention here. Nevertheless, I would like to sincerely thank them all.

Contents

Abstract

Methods from verification have been lifted to synthesis. One of the techniques is context-free games [5]. Finding winning strategy for the games can be carried out by using fixed-point iteration. This fixed-point iteration computes the set of plays starting from a given initial sentential form. This set of plays is represented by a *negation-free Boolean formula* with implication as the order. In this thesis, we propose a set of heuristic optimizations by weakening the notion of implication for the context-free games. For this, we adapt simulation relations for inclusion testing of Büchi automata [1]. This will reduce the number of implication checks and reduce memory consumption. In addition, we also propose a formula reduction technique which will further improve runtime. This may give us significant improvement, especially if successful reductions happen early in the iteration. This is because it will also reduce the computational costs of all future steps of the iteration. In addition, a significant decrease in formula size will both decrease memory consumption and contributes extra speed up. We test our optimizations by solving instances with NFAs and CFGs generated according to Tabakov-Vardi model [10]. Not all of the game instances generated are optimizable. This is due to the fact that our optimizers rely heavily on the structure of the NFAs and CFGs. However, for most cases, we are able to get better performance.

Chapter 1

Introduction

Synthesis and verification problems for recursive programs are a pair of problems often discussed together in the context of automatic programming. This is the task of automatically discovering parts of a program that are still missing in order to satisfy some specification. Possible applications are helping end-users automate repetitive tasks [2], automating geometry constructions [3], etc.

Synthesis can carried be out by *saturation* or *summarization*. When using *saturation* [7], we perform a backward reachability analysis from a given set of regular configurations. This results in a pre*-image consisting of all configurations which can reach the set of regular configurations. We then check whether the initial configuration is in the pre*-image. *Summarization* amounts to combining parts of the computations with similar input-output pairs. In other words, we analyze a program in terms of procedure summaries.

A recent summarization approach [5], models the synthesis problem by using an inclusion game on the derivation tree of a context-free grammar. Demonic and angelic non-determinism are modeled as choices for two players respectively: *refuter* and *prover*. Synthesis amounts to finding a winning strategy for *prover*. One significant part of computing this strategy is the fixed-point iteration based on the implication relation. This part determines the runtime of a context-free game instance and is suitable for optimizations. In [5], the authors also mentioned the possibility of heuristics that can be used for optimizations: an antichain optimization and a lazy evaluation strategy. In this thesis, we try to apply the subsumption relations used in [1] to obtain antichain optimizations.

1.1 Contribution

The contribution of this thesis is a set of optimization techniques for context-free games. The optimizations work by adapting the simulation relations used in [1]. In the paper, the simulation relations were used to avoid a part of the search space that is subsumed by the other part when performing inclusion check for Büchi automata. We make use of the relations for NFAs. In total, we explore four relations: subset relation, forward simulation relation, backward simulation relation, and forward-backward simulation relation. This is designed to stop the fixed-point iteration earlier and reduce the size of the search space representation.

For the search space representation, we use *negation-free Boolean formulas*. By using the simulation relations, we can reduce these formulas. Formula reductions have an important advantage for the fixed-point iteration's runtime. An early successful reduction will help to reduce the computational costs of the later steps of the iteration. This is due to monotonicity, a property that we will later explain in detail. Due to this property, a reducible part of a formula will remain reducible throughout the iteration despite being transformed into different formulas. If this part is reduced, the later steps of iteration will also be affected. Two key operations which enjoy this benefit are implication checks and formula compositions.

We devise experiments to compare the performance of our simulation based solvers with a CNF solver without simulation relations. We add our solvers into a c++ program developed at TU Kaiserslautern. Many of the functionalities are already available. These include instance generation according to the Tabakov-Vardi model [10], a CNF based solver, a worklist based Kleene iteration algorithm, etc. The results show that our solvers improve the runtime of many instances that are in line with the properties of our simulation relations. As an observation, a DFA can not be used in conjunction with the subset relation.

1.2 Structure

This thesis is organized in five parts. After the introduction, we first introduce some concepts necessary for the development of the simulation relations in Chapter 2. This part includes the concept about context-free games in general, the domain of the search space, and the fixed-point iteration. We leave some concepts such as strategy synthesis as they are not within the scope of our work. Interested readers can refer to [5].

In Chapter 3, we show how to adapt simulation relations in [1] for our case. This includes adapting the different notations and redefining the acceptance condition. We use boxes instead of supergraphs and we need an acceptance condition for NFAs instead of NBAs. The domain of the fixed-point iteration is vectors of formulas. We also show how this can be used to reduce the representation of the formulas to speed up the fixed-point iteration.

Then, Chapter 4 is devoted to showing experimental evidence. We use a program developed in the Concurrency Theory Group at TU Kaiserslautern [9] and develop additional subroutines to support our work. For the simulation relations, we also use some implementation tricks to decrease the overhead caused by the additional computations. These include using indices to represent more complex data structure, hashing of the simulation relations, using pointers whenever possible, and taking advantage of the relation between different simulation relations.

In Chapter 5, we present the conclusion and future work. We revisit the work that we have done. Afterwards, we present the possible future improvements and workarounds. Some of them are using SAT solvers and the analysis of instance generation.

In addition, we put the results of the experiments in the Appendix. In total, there are 160 context-free game instances. We present the results in two tables. The first table contains results of some representative individual instances and the second table contains the aggregate results.

Chapter 2

Preliminary Concepts

In this chapter, we first get into the basic idea of context-free games and introduce some notations necessary for the optimizations. We will make use of the simulation relations from [1] and argue why we can use them in the next chapter.

2.1 Context-Free Games

Given a program template P and a specification ψ, we would like to find an instantiation $P@i$ which satisfies ψ. This is the essence of program synthesis. The program template is modelled as context-free grammar and the specification can be modelled as regular language. We can then see this synthesis problem as an inclusion problem. We want to determine whether a context-free language is in some regular language.

This problem is closely related to verification problem. We can see this when we consider the complement of the inclusion problem. More specifically, verification problem amounts to finding a word that is outside of a regular language. With this, we can check whether the program template violates the specification.

One way that is often used to represent a regular language is by using nondeterministic finite automata [6]. This is the representation of regular languages that we will use for the simulation relations.

Definition 2.1.1 (NFA)**.** *A Nondeterminstic Finite Automata NFA A over a finite alphabet Σ is a triple $(Q, \rightarrow, F)$ with a single initial state $q_0 \in Q$ where:*

1. *Q is a finite set of states.*
2. *$F \subseteq Q$ is a finite set of final states.*
3. *$\rightarrow: Q \times \Sigma \rightarrow P(Q)$ is a transition function that determines which states the automata can move to based on the current letter.*

On the other hand, we use a context-free grammar to represent the program. For context-free games, we consider CFGs where the nonterminals are divided into two sets.

Definition 2.1.2 (Context-free grammar)**.** *A context-free grammar G with ownership partitioning is a triple (N, Σ, P). where:*

1. *$N = N_{\Box} \uplus N_{\bigcirc}$ is a finite set of nonterminals*
2. *Σ is a finite set of terminals*
3. *$P \subseteq N \times (N \cup \Sigma)^*$ is a finite set of production rules.*

The ownership partitioning is also extended to sentential forms $\vartheta = (N \cup \Sigma)^*$. The ownership of $\alpha = wX\beta$ depends on the leftmost nonterminal X. α belongs to *prover* if $X \in N_{\Box}$ and *refuter* if $X \in N_{\bigcirc}$.

Given a context-free grammar G and an NFA A, a context-free game instance $(\vartheta, \Rightarrow_L, \alpha)$ is a triple with ϑ the set of sentential forms in G, $\Rightarrow_L$ is a left derivation relation based on the production rules of G, and $\alpha \in \vartheta$ is the initial position. *Prover* (*refuter*) has to do a leftmost derivation starting from α. *Prover* (*refuter*) takes turn when she owns the leftmost nonterminal of the sentential form and proceeds by choosing one of the right hand side choices. *Prover* wins either by enforcing an infinite play or deriving a word inside $L(A)$ while *refuter* wins by deriving a word outside of $L(A)$.

Note that inclusion games are determined [4]. For any sentential form $\alpha \in (N \cup \Sigma)^*$, exactly one of the players has a winning strategy. For our purpose, we use the non-inclusion game where we want to find the strategy for *refuter*.

Fixed-point iteration is used to compute a representation of the set of all plays starting a sentential form $\alpha \in (N \cup \Sigma)^*$. Then, we can determine the winner by evaluating this representation.

2.2 Domain

Here, we try to explore the domain over which the fixed-point iteration operates. The fixed-point iteration works on the representation of the set of all plays starting from all of the nonterminals. A play $p = p_0 p_1 ...$ is a sequence of sentential forms p_i such that $p_i \Rightarrow_L p_{i+1}$. The set of plays starting from a sentential form α forms a tree with α as root. Branches in the tree represent possible choices either for *refuter* or *prover*. Based on [4], we can see this tree as a *negation-free Boolean formula* over the set of words where inner nodes are disjunctions (conjunctions) when they are owned by *refuter* (*prover*). They also provided a way to represent it in a finite way by representing infinite plays as *false*. This means that neither *refuter* nor *prover* derives a word. This happens for emptiness games where the regulars specification is an empty language. This finite Boolean formula will be the domain.

Now, we need to describe the propositions for the formulas. This should represent the terminal words derivable in a play. Usually, the language of an NFA is infinite. In order to get a finite representation, we put the words in the language into equivalence classes based on their induced state changes. This is due to the fact that what matter for acceptance condition are the state changes. Two words inducing similar state changes will have the same behaviour in terms of the acceptance condition of the NFA. So, they are defined to be equivalent.

Definition 2.2.1 (Transition equivalence $\backsim_A$)**.** *Given NFA* $A = (Q, \rightarrow, F)$, $u, v \in \Sigma^*$. $u \backsim_A v$ *if for all* $p, q \in Q$, $p \xrightarrow{u} q$ *iff* $p \xrightarrow{v} q$.

We treat the finite number of equivalence classes as propositions in our formulas. We call these equivalence classes *boxes*. For a word $w \in \Sigma$, the box ρ_w is the set of state pairs which transitions are induced by the word w. If there is another word v such that $v \backsim_A w$, then, based on the definition of $\backsim_A$, we get $\rho_w = \rho_v$.

Definition 2.2.2 (Box)**.** $\rho_w = \{(p, q) | p \xrightarrow{w} q\}$.

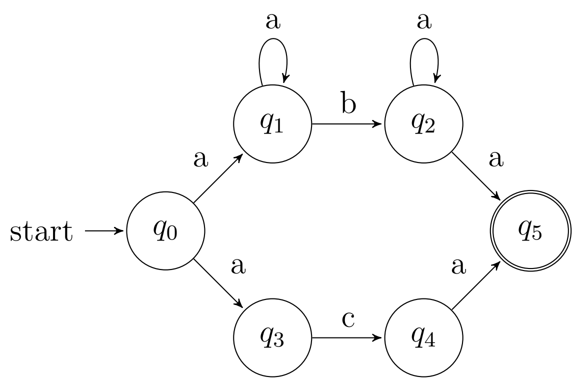

Figure 2.1: NFA A

Graphically, a box is drawn as a rectangle with arcs inside representing state changes. The boxes in Figure 2.2 are based on the NFA A in Figure 2.1 representing the language $aa^*ba^*a + aca$. The language of a box is the set of words inducing state changes in the box. For example, $L(\rho_{ba}) = a^*ba^*a$.

When we compose two plays together, we need to maintain that the terminal word of the resulting play still induces state changes. For this, we have a *composition* operation [5]. An arc (p, q) exists in the composition of two boxes ρ and τ if there is a state r such that $(p, r) \in \rho$ and $(r, q) \in \tau$. The set of all boxes equipped with the composition operation and ρ_ϵ as an identity element forms a monoid.

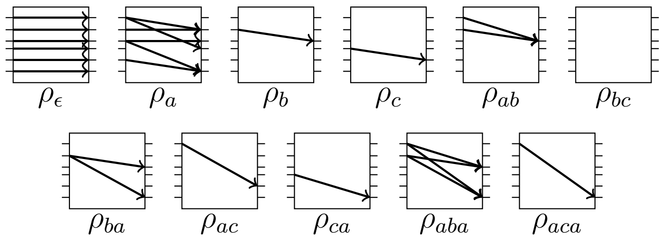

Figure 2.2: Boxes for NFA A

Definition 2.2.3 (Composition). $\rho_v; \rho_w = \{(p, q)|$ *there is* $r \in Q(p, r) \in \rho_v$ *and* $(r, q) \in \rho_w\} = \rho_{vw}$.

When we deal with context-free games, determining the winner relies on the acceptance condition of the NFA. This is formalized as Definition 2.2.4 [4]. The idea is to categorize boxes according to whether their languages contain accepted words. Since we see this game from the perspective of *refuter*, we see a box as a proposition with Boolean value *true* if *refuter* can derive any words in the language of the box to win.

Definition 2.2.4 (Rejecting). *A box* τ *is rejecting if there is no* $(q_0, q_f) \in \tau$ *for any* $q_f \in F$*. We assign true to rejecting boxes for the evaluation of a formula.*

Now that we have boxes as propositions, we are ready to form the formulas. We use the perspective of the non-inclusion game for *refuter*. In order for the *refuter* to win, she needs to be able to choose production rules within a play which will eventually lead to a terminal word outside of $L(A)$ given *prover*'s move. The choices of *prover* are manifested as disjunction $\vee$ and the choices of *refuter* are manifested as conjunction $\wedge$.

Definition 2.2.5 (Formula). *A Formula F is defined inductively as*

$$F := \rho | G \wedge G' | G \vee G' | false$$

with $false \wedge F = false$ *and* $false \vee F = F$.

The composition between two formulas is again a formula. This is lifted from the composition of boxes and used to reflect the composition of the plays represented by the two formulas. The composition happens in two different manners. The first one is when the left formula operand does not consist of a single box only. In other words, it consists of smaller formulas connected by a logical operator. For this, we simply distribute the composition of the right formula operand to the two smaller formulas from the left operand. The next case is when the left operand is a box and the right operand consists of smaller formulas connected by a logical operater. Just like the first case, we distribute the composition of the box over the two smaller formulas from the right operand. Both occur without changing the logical operator. Since the composition operation for boxes is non-commutative, The order of the composition for formulas should also be intact.

Definition 2.2.6 (Composition between formulas)**.** $(F \stackrel{\wedge}{\vee} F'); G = (F; G \stackrel{\wedge}{\vee} F'; G)$ *and* $\tau; (F \stackrel{\wedge}{\vee} F') = (\tau; F \stackrel{\wedge}{\vee} \tau; F')$

For the implication check, we need a notion of evaluation of formulas just like in propositional logic. Firstly, we need a box assignment which assigns either *true* or *false* to boxes. This assignment is also used for box evaluation. Then, we define formula evaluation by setting up the rules for operators $\wedge$ and $\vee$.

Definition 2.2.7 (Evaluation)**.** *Given box assignment* $v = M_A \rightarrow \{false, true\}$ *we define evaluation of formula as* $e_v(\rho) = v(\rho)$, $e_v(F \wedge G) = e_v(F) \wedge e_v(G)$ *and* $e_v(F \vee G) = e_v(F) \vee e_v(G)$.

Now, we need to understand what implication means when dealing with formulas representing plays. Intuitively, $F \Rightarrow G$ means *refuter* has more choices from the set of plays represented by G [5]. The set of formulas BF with this relation is still a quasi order where two different formulas can imply each other. Therefore, we use $\Leftrightarrow$ to form a finite equivalence classes of formulas $BF_{/\Leftrightarrow}$ to get the *antisymmetry* property. We will later perform fixed-point iteration on $(BF_{/\Leftrightarrow}, \Rightarrow)$. The monotonic function for the iteration reflects all possible steps *prover* and *refuter* can take from a given sentential form.

2.3 Fixed-Point Iteration

With a monotonic function [5] and that $BF/_{\Leftrightarrow}$ is a lattice, we can use fixed-point iteration to solve a context-free game instance. The existence is due to Theorem 2.3.1. The result of the iteration is a vector of formulas representing the set of all plays starting from all nonterminals. To determine the winner, we assign true to rejecting boxes and evaluate the formulas. If the resulting formula evaluates to true, then the winner is *refuter*. Otherwise, it means *prover* has a winning strategy.

Theorem 2.3.1 (Knaster, Tarski '55)**.** *Let* $(D, \leq)$ *be a complete lattice and* $f : D \to D$ *a monotonic function, then*

$$x = \sqcap\{d \in D | f(d) \leq d\}$$

is the least fixed point, and

$$y = \sqcup\{d \in D | f(d) \leq d\}$$

is the greatest fixed point.

Theorem 2.3.1 only states about the existence of the least fixed point. In order to find it, we need an algorithm based on Kleene fixed-point theorem [8].

Theorem 2.3.2 (Kleene fixed-point theorem)**.** *Let* $(D, \leq)$ *be a complete lattice with a least element* $\perp \in D$ *and* $f : D \to D$ *a monotonic function, then we can get a chain*

$$\perp \leq f(\perp) \leq f(f(\perp)) \cdots \leq f^n(\perp) \leq \dots$$

and obtain the least fixed-point of f

$$lfp(f) = \sqcup(f^n(\perp) | n \in \mathbb{N})$$

The monotonic function for our context-free games reflects one step of a play considering all of the choices for the current player [5]. This function is based on the production rules. It takes a vector of formulas as an input. A formula Δ_{X_i} in the vector represents the set of plays starting from a nonterminal X_i. Given a production rule $X_i \to \eta_1|...|\eta_l$ of X_i, we calculate a new formula $\Delta_{X_i} = f_{X_i}(\Delta_{X_1}, ..., \Delta_{X_k})$ with

$$f_{X_i}(\Delta_{X_1}, ..., \Delta_{X_k}) = \Delta_{\eta_1} \wedge ... \wedge \Delta_{\eta_1}$$

if X_i belongs to *prover* and

$$f_{X_i}(\Delta_{X_1}, ..., \Delta_{X_k}) = \Delta_{\eta_1} \vee ... \vee \Delta_{\eta_1}$$

otherwise. Initally, we use $\Delta_{X_1} = ... = \Delta_{X_k} = false$ because $false$ is the bottom element in $BF/_{\Leftrightarrow}$. We perform this until fixed-point.

There are several variations of the fixed-point iteration. For our purpose, we use worklist based Kleene iteration algorithm. This choice, however, does not affect our results in a meaningful way. Even when we choose arbitrary formulas as in chaotic iteration [4], the results and performances should remain stable.

Chapter 3

Proposed Optimizations

The fixed-point iteration algorithm for context-free games is doubly exponential in terms of the number of states of the automaton. It is therefore expected to be beneficial to have optimizations that take into account the states of the NFA. These optimizations should also be consistent with the result of the fixed-point iteration. In other words, the winner of a game instance does not change whether we use the optimizations or not. Since the game is basically checking for language inclusion, the behaviour that we are interested in is related to the acceptance condition. In particular, the optimizations should take into account Definition 2.2.4.

This fixed-point iteration operates on the set of *negation-free Boolean formulas* (up to logical equivalence) $BF_{/\Leftrightarrow}$ which forms a partial order with implication as the ordering. In the original formulation, there are no implication relations between different boxes. Here, we try to weaken this notion of implication such that we have more relations between boxes. First, we explore how subset relation between boxes can be used. Then, we adopt the relations from [1] which take advantage of the simulation relations between states. With these relations, we will have more implication relations between formulas. Therefore, we can possibly have more successful implication checks and reduce the number of iterations. Also, we can reduce the size of the Boolean formulas. Whenever there are simulation relations between boxes within a clause, we can perform reductions until any two boxes in the clause do not have implication relation. We also do the same with clauses such that in the end, there are no two clauses having implication relation.

3.1 Subset Relation between Boxes

The first relation that we want to consider is the subset relation. Since a box is basically a set of arcs, a simple subset relation would be sufficient for our first notion of implication. If a box is a subset of another box, then its behaviour is subsumed by the other box. In particular, when the set of state changes in a box is a superset of those in another box, then the smaller one will be rejecting if the superset box is rejecting.

Definition 3.1.1. *Let ρ, τ boxes. $\rho \sqsubseteq_s \tau$ if $\rho \subseteq \tau$.*

Here, the notion of implication is that non-existence of any arc (q_0, q_f) for $q_f \in F$ in a box ρ means that any boxes that are subsets of ρ will also not contain (q_0, q_f). Therefore, rejection of ρ also means rejection of the other smaller boxes.

Lemma 3.1.2. *Let ρ, τ boxes, if $\rho \sqsubseteq_s \tau$ and τ is rejecting, then ρ is rejecting*

Proof. Since τ is rejecting, then, for any $q_f \in F$, $(q_0, q_f) \notin \tau$. Because $\rho \sqsubseteq_s \tau$, it means $\rho \subseteq \tau$. Therefore $(q_0, q_f) \notin \rho$. Hence, ρ is rejecting. □

For NFA A in Figure 2.1, the example is $\rho_b \sqsubseteq \rho_{aba}$. This is illustrated in Figure 3.1. We can see that box ρ_b only contains arc (q_1, q_2) and this arc is also contained in ρ_{aba}. It means that $\rho_b \subseteq \rho_{aba}$. Suppose ρ_{aba} is rejecting. It means there is no arc (q_0, q_f) for any $q_f \in F$ in ρ_{aba}. Therefor, it means $(q_1, q_2) \neq (q_0, q_f)$. So, ρ_b is also rejecting.

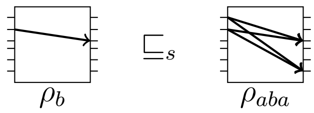

Figure 3.1: Subset relation example for NFA A in Figure 2.1.

This will enable us to remove one of two boxes related by $\sqsubseteq_s$ in a formula. We will revisit this after introducing the other relations.

3.2 Simulation Relations

Other kinds of relation deal with simulation of the states of an NFA with respect to its acceptance condition. For this, we use simulation relations that were successfully used for Büchi automata in [1]. These relations are on the states of NBAs and we will use it for NFAs with different acceptance condition (i.e. existence of accepting run instead of lasso). However, since our domain is a transition monoid consisting boxes, we will also lift these relations into boxes. We do this by maintaining that simulated boxes have subsumed behaviours.

There are three main relations that we want to consider: forward, backward, and forward-backward. The first relation is forward simulation relation. This relation is based on the future behaviour of a program, given two states the program can take. In terms of the acceptance condition, forward reachability of final state is a necessary behaviour which should be reflected in the definition. The next relation is backward simulation relation. In contrast with forward simulation relation, backward simulation relation is based on the past behaviour of a program. In terms of acceptance condition, backward reachability of initial state should be reflected in the definition. Finally, forward-backward simulation relation combines both of the previous relations.

3.2.1 Simulation Relations between States

Before we define our simulation relations between boxes, we start with simulation relations between states. The first simulation relation that we would like to evaluate is the forward simulation relation. if a state p is forward simulated by another state q, it means that the behaviour of the automata starting from p subsumes the behaviour starting from q.

Definition 3.2.1. *Let NFA $A = (Q, \to, F)$ and $p, r \in Q$, we have $p \preccurlyeq_f r$ only if*

$$\forall p \xrightarrow{a} p' \exists r \xrightarrow{a} r' \text{ s.t. } p' \preccurlyeq_f r' \text{ and } p \in F \Rightarrow r \in F$$

From the definition, we can see that for any state change from state p, there is also a state change induced by the same letter from q. Moreover, whenever p is a final state, q will also be a final state mimicking the accepting behaviour of p. For the NFA in Figure 2.1, the simulation relation based on $\preccurlyeq_f$ is $q_4 \preccurlyeq_f q_2$.

Now, we define the backward simulation relation between states. However, this is different from the original definition [1] where forward reachability of final states is also taken into account.

Definition 3.2.2. *Let NFA $A = (Q, \to, F)$ and $p', r' \in Q$. $p' \preccurlyeq_b r'$ only if*

$$\forall p \xrightarrow{a} p' \exists r \xrightarrow{a} r' \text{ s.t. } p \preccurlyeq_b r \text{ and } (p' = q_0) \Rightarrow (r' = q_0)$$

The original definition also requires $p' \in F \Rightarrow r' \in F$. This is due the difference in the acceptance conditions. For Büchi automata, the acceptance condition is the existence of lasso: there is a path from the initial state to some final state and the set of final states is visited infinitely often. For NFA A, the example is $q_3 \preccurlyeq_b q_1$.

3.2.2 Simulation Relations between Arcs in Boxes

In order to lift the relations between states to boxes, we have an additional intermediary step. A box is essentially a set of arcs and here, we define relations between arcs. These relations between arcs will then be used for relations between boxes.

Firstly, we define forward simulation relation for arcs. The idea is that if the right endpoint state q of an arc (p, q) is simulated by another right endpoint state s of another arc (r, s), then the behaviour of the arc (p, q) can be simulated by the behaviour of the arc (r, s). Here, p should equal r. In the same line of argument for the relation between states, this behaviour is still reachability of final states. In other words, we lift the behaviour of forward simulation between states into the behaviour of forward simulation between arcs.

Definition 3.2.3. *Let ρ, τ boxes with $(p, q) \in \rho$ and $(r, s) \in \tau$, then*

$$(p, q) \sqsubseteq_f (r, s) \text{ if } p = r \text{ and } q \preccurlyeq_f s$$

For the NFA in Figure 2.1, we have $(q_0, q_4) \sqsubseteq_f (q_0, q_2)$. We get this relation because we have $q_4 \preccurlyeq_f q_2$ from the previous subsection.

Next, we define backward simulation relation for arcs. In line with the forward simulation relation, the idea is that if the left endpoint state p of an arc (p, q) is simulated by another left endpoint state r of another arc (r, s), then the behaviour of the arc (p, q) can be simulated by the behaviour of the arc (r, s). Here, q should equal s. In contrast with the forward simulation relation, it is the behaviour of backward simulation between states that we lift to arcs. This behaviour is backward reachability of initial state q_0.

Definition 3.2.4. *Let ρ, τ boxes, $(p, q) \in \rho$ and $(r, s) \in \tau$, then*

$$(p, q) \sqsubseteq_b (r, s) \text{ if } q = s \text{ and } p \preccurlyeq_b r$$

For the NFA A in Figure 2.1, we have $(q_3, q_5) \sqsubseteq_b (q_1, q_5)$. As we already know from the previous subsection that $q_3 \preccurlyeq_b q_1$.

Lastly, by using both forward and backward simulation between states, we define the forward-backward simulation relation between arcs. More specifically, an arc is simulated by another arc if both of its respective endpoint states are simulated by the other arc. The left and right endpoint states should be both simulated.

Definition 3.2.5. *Let ρ, τ boxes, $(p, q) \in \rho$ and $(r, s) \in \tau$. $(p, q) \sqsubseteq_{fb} (r, s)$ if $q \preccurlyeq_f s$ and $p \preccurlyeq_b r$*

3.2.3 Simulation Relations between Boxes

Now, we are ready to define simulation relation between boxes. We define forward simulation between boxes by using the arcs. A box ρ is simulated by τ if all of ρ's arcs are simulated by some of τ's arcs.

Definition 3.2.6. *Let ρ, τ boxes, then $\rho \sqsubseteq_f \tau$ if for all $(p, q) \in \rho$, there is $(r, s) \in \tau$ such that $(p, q) \sqsubseteq_f (r, s)$*

For the NFA A in Figure 2.1, $\rho_{ac} \sqsubseteq_f \rho_{ab}$. As we can see from Figure 3.2, ρ_{ac} only has one arc (q_0,q_4). This arc is simulated by one of the arcs in ρ_{ab}. (q_0,q_4) is forward simulated by (q_0,q_2).

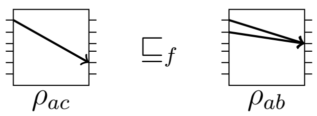

Figure 3.2: Forward simulation relation example for NFA A in Figure 2.1

Next, we show that this forward simulation behaves like implication when we use the notion of rejection from Definition 2.2.4. It means that if a box ρ is forward simulated by another box τ, then nonexistence of accepted words in $L(\rho)$ will imply nonexistence of accepted words in $L(\tau)$.

Lemma 3.2.7. *Let ρ, τ boxes, if $\rho \sqsubseteq_f \tau$ and τ is rejecting, then ρ is rejecting*

Proof. Since τ is rejecting, then for any $q_f \in F$, $(q_0, q_f) \notin \tau$. Assume ρ is not rejecting, then $(q_0, q_f) \in \rho$ and because $\rho \sqsubseteq_f \tau$, it means $(q_0, q_f) \in \tau$ which is a contradiction. □

The second relation that we need to have is backward simulation between boxes. Again, we define backward simulation between boxes by using the arcs. A box ρ is backward simulated by τ, if all of ρ's arcs is simulated by some of τ's arcs.

Definition 3.2.8. *Let ρ, τ boxes, $\rho \sqsubseteq_b \tau$ if for all $(p, q) \in \rho$, there is $(r, s) \in \tau$ such that $(p, q) \sqsubseteq_b (r, s)$.*

Just as for forward simulation relation, we need to show that if τ is rejecting, then ρ is rejecting. This is expressed in the next lemma. In line with the idea of proof for Lemma 3.2.7, we argue using the arcs and show that nonexistence of accepting arc in one box implies nonexistence of accepting arc in the other box.

Lemma 3.2.9. *Let ρ, τ boxes, if $\rho \sqsubseteq_b \tau$ and τ is rejecting, then ρ is rejecting*

Proof. Since τ is rejecting, then, for any $q_f \in F$, $(q_0, q_f) \notin \tau$. Assume ρ is not rejecting, then $(q_0, q_f) \in \rho$ and because $\rho \sqsubseteq_b \tau$, it means there is an arc $(q_0, q_a) \in \tau$ such that $q_a = q_f$ and q_0 is an initial state. which means τ is not rejecting. Hence, we have a contradiction. □

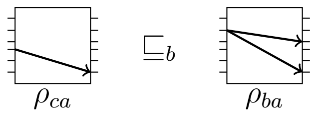

Figure 3.3: Backward simulation relation example for NFA A in Figure 2.1.

Lastly, we define forward-backward simulation between boxes. As the name implies, this is a combination between forward and backward simulation between boxes. For both forward and backward simulation, the related arcs have one common endpoint. In contrast, forward-backward simulation weakens this such that an arc takes advantage of both forward and backward simulation between states on both endpoints.

Definition 3.2.10. *Let ρ, τ boxes, then $\rho \sqsubseteq_{fb} \tau$ if for all $(p, q) \in \rho$, there is $(r, s) \in \tau$ such that $(p, q) \sqsubseteq_{fb} (r, s)$.*

For the NFA A in Figure 2.1, $\rho_c \sqsubseteq_{fb} \rho_b$. This is an example of a forward-backward simulation relation between boxes that is neither a forward nor a backward simulation relation. So, forward-backward simulation relation is not simply a union of forward and backward simulation relations. Since both boxes have only one arc, it is easy to see that $\rho_c \sqsubseteq_{fb} \rho_b$. This is because $(q_3, q_4) \sqsubseteq_{fb} (q_1, q_2)$.

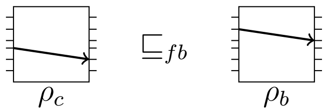

Figure 3.4: Forward-backward simulation relation example for NFA A in Figure 2.1.

Lemma 3.2.11. *Let ρ, τ boxes, if $\rho \sqsubseteq_{fb} \tau$ and τ is rejecting, then ρ is rejecting*

Proof. Since τ is rejecting, then, for any $q_f \in F$, $(q_0, q_f) \notin \tau$. Assume ρ is not rejecting, then $(q_0, q_f) \in \rho$ and because $\rho \sqsubseteq_{fb} \tau$, it means there is $(q_0, q_f) \in \tau$ such that $q_f \in F$ and q_0 is an initial state. It means τ is not rejecting. Hence, we have a contradiction. □

All of the four relations are in fact connected. Due to reflexivity, the forward simulation relation is also a forward-backward simulation relation. This is also the case with the backward simulation relation. However, forward and backward simulation relations are not subset of each other. Lastly, since an arc is both forward and backward simulates itself, the subset relation is both a forward and a backward simulation relation. We will use this observation later to develop one of the implementation tricks to decrease the extra costs caused by our optimization techniques.

3.2.4 Monotonicity for Boxes

For the fixed-point iteration to work, it is necessary that these simulation relations are monotonic with respect to composition. Later, we will use this to show the monotonicity for formulas. In order to make it easier, we first rephrase the definition of the simulation relations by using not only letters but words.

Lemma 3.2.12. *Let* $p, p', q, q' \in Q$*, if* $p \preccurlyeq_f p'$*, for all* $w \in \Sigma^*$ *with* $p \xrightarrow{w} q$*, there is* $p' \xrightarrow{w} q'$ *such that* $q \preccurlyeq_f q'$

Proof. We prove this using induction on length of w.
(**Base case**) Let $w = \epsilon$. Since $p \xrightarrow{\epsilon} p$, there is $p' \xrightarrow{\epsilon} q'$ such that $p' \preccurlyeq_f q'$ where $p' = p$ and $q' = p$.
(**Induction step**) Let $w \in \Sigma^*$ such that $p \xrightarrow{w} p'$. Based on induction hypothesis, there is $q \xrightarrow{w} q'$ such that $q \preccurlyeq_f q'$. Let $a \in \Sigma$ a letter such that $p' \xrightarrow{a} p''$ for some $p'' \in Q$. Since $p' \preccurlyeq_f q'$, it means there must be a transition $q' \xrightarrow{a} q''$ for some $q'' \in Q$ with $p'' \preccurlyeq_f q''$. Therefore, for the transition $p \xrightarrow{w.a} p''$ there is $q \xrightarrow{w.a} q''$ such that $p'' \preccurlyeq_f q''$. □

We also need to rephrase the backward simulation relation. This is stated in Lemma 3.2.13.

Lemma 3.2.13. *Let* $p, p', q, q' \in Q$*, if* $p \preccurlyeq_b p'$*, for all* $w \in \Sigma^*$ *with* $p \xrightarrow{w} q$*, there is* $p' \xrightarrow{w} q'$ *such that* $q \preccurlyeq_b q'$

Proof. Analogous to Lemma 3.2.12, but backwards. □

When we compose boxes, it is necessary that the simulation relations remain consistent. This is important for the fixed point algorithm. The algorithm requires the monotonicity property. The idea is to show that all of the arcs within the composed smaller boxes are simulated by some arcs in the composition of the larger boxes by using the properties of arcs resulting from the composition operation.

We will prove the monotonicity property only for forward-backward simulation relation. However, due to *reflexivity*, the monotonicity of forward, backward, and subset subsumption follows.

Lemma 3.2.14. *Let* τ, τ', ρ, ρ' *be boxes, If* $\tau \sqsubseteq_{fb} \tau'$ *and* $\rho \sqsubseteq_{fb} \rho'$ *then* $\tau; \rho \sqsubseteq_{fb} \tau'; \rho'$

Proof. **(1)**First, we prove that $\tau; \rho \sqsubseteq_{fb} \tau'; \rho$. Let $(p, q) \in \tau; \rho$, it means that there is $r \in Q$ such that $(p, r) \in \tau$ and $(r, q) \in \rho$. Since $\tau \sqsubseteq_{fb} \tau'$, it means there is $(p', r') \in \tau'$ such that $(p, r) \sqsubseteq_{fb} (p', r')$. Let $w \in L(\rho)$. Because $r \preccurlyeq_f r'$, then, there is q' such that $r' \xrightarrow{w} q'$. In other words, $w \in L(\rho)$ also induces state change from r' to q'. Therefore $(r', q') \in \rho$. Moreover, based on Lemma 3.2.12 $q \preccurlyeq_f q'$. Since $(p', r') \in \tau'$ and $(r', q') \in \rho$ it means $(p', q') \in \tau'; \rho$. Also, because $p \preccurlyeq_b p'$ and $q \preccurlyeq_f q'$, it means $\tau; \rho \sqsubseteq_{fb} \tau'; \rho$
(2) Next we prove that $\tau'; \rho \sqsubseteq_{fb} \tau'; \rho'$ Let $(p, q) \in \tau'; \rho$, It means there is $r \in Q$ such that $(p, r) \in \tau'$ and $(r, q) \in \rho$. Since $\rho \sqsubseteq_{fb} \rho'$, it means there is $(r', q') \in \rho'$ such that $(r, q) \sqsubseteq_{fb} (r', q')$. Let $w \in L(\tau')$. Because $r \preccurlyeq_b r'$, then there is p' such that $p' \xrightarrow{w} r'$. In other words, $w \in L(\tau')$ also induces state change from p' to r'. Therefore $(p', r') \in \tau'$. Moreover, based on Lemma 3.2.13 $p \preccurlyeq_b p'$. Since $(p', r') \in \tau'$ and $(r', q') \in \rho'$ it means $(p', q') \in \tau'; \rho'$. Also, because $p \preccurlyeq_b p'$ and $q \preccurlyeq_f q'$, it means $\tau'; \rho \sqsubseteq_{fb} \tau'; \rho'$
(3) By transitivity, $\tau; \rho \sqsubseteq_{fb} \tau'; \rho'$ □

3.2.5 Implication and Monotonicity for Formulas

In this part, we try to weaken the notion of implication between formulas by using the simulation relations we devised in the previous section. The implication between boxes ρ and τ used in [5] works simply by checking whether $\rho = \tau$. Before we do this for formulas, we first define the implication between boxes. Since we already have four simulation relations, we use $r \in \{s, f, b, fb\}$ to indicate which relations an implication is based on.

Definition 3.2.15 (Implication between boxes)**.** *$\rho \Rightarrow_r \tau$ if whenever ρ is rejecting, then τ is rejecting. $\tau \sqsubseteq_r \rho$ if and only if $\rho \Rightarrow_r \tau$.*

Now, we are ready to weaken the notion of implication for formulas. The following definition is mentioned [4] as a future work.

Definition 3.2.16 (Implication between formulas)**.** *We define $F \Rightarrow_r G$ as $G \sqsubseteq_r F$:*

$$\bigwedge_{\rho, \tau \in M_A : \tau \sqsubseteq_r \rho} \rho \Rightarrow_r \tau \models F \Rightarrow_r G \tag{3.1}$$

Here, we lift the definition of $\sqsubseteq_r$ between boxes into relation between formulas. We use the symbol $\Rightarrow_r$ to distinguish it from $\Rightarrow$ where r indicates which relation to use.

Lastly, we need to argue about its correctness. We need to make sure that monotonicity holds when composing formulas. The idea of the proof is the same as the monotonicity proof for $\Rightarrow$ in [5]. We first need to show that some distributivity properties analogous to those of implication also apply for $\sqsubseteq_r$. Firstly, we need to show a property similar to

$$(A \overset{\wedge}{\vee} A' \Rightarrow B) \Leftrightarrow (A \Rightarrow B \overset{\vee}{\wedge} A' \Rightarrow B).$$

Here, we can replace $\overset{\wedge}{\vee}$ with either $\wedge$ or $\vee$ and $\overset{\vee}{\wedge}$ should be replaced by the logical operator not replacing $\overset{\wedge}{\vee}$. The property is formulated in lemma 3.2.17. The idea of the proof is simply to use the definition and argue by using the similar distributivity property for $\Rightarrow$.

Lemma 3.2.17. $(F) \sqsubseteq_r (G \overset{\wedge}{\vee} G') \Leftrightarrow (F \sqsubseteq_r G \overset{\vee}{\wedge} F \sqsubseteq_r G')$

Proof. $(F) \sqsubseteq_r (G \overset{\wedge}{\vee} G') \Leftrightarrow \bigwedge_{\rho,\tau \in M_A : \tau \sqsubseteq_r \rho} \tau \Rightarrow \rho \models G \overset{\wedge}{\vee} G' \Rightarrow F$
$\Leftrightarrow$ For any assignment v such that whenever $e_v(\bigwedge_{\rho,\tau \in M_A : \tau \sqsubseteq_r \rho} \tau \Rightarrow \rho) = true$ we have $(e_v(G) = true \overset{\wedge}{\vee} e_v(G') = true) \Rightarrow e(F) = true$
$\Leftrightarrow$ For any assignment v such that whenever $e_v(\bigwedge_{\rho,\tau \in M_A : \tau \sqsubseteq_r \rho} \tau \Rightarrow \rho) = true$ we have $(e_v(G) = true \Rightarrow e_v(F) = true) \overset{\vee}{\wedge} (e(G') = true \Rightarrow e(F) = true)$
$\Leftrightarrow \bigwedge_{\rho,\tau \in M_A : \tau \sqsubseteq_r \rho} \tau \Rightarrow \rho \models (G \Rightarrow F) \overset{\vee}{\wedge} (G' \Rightarrow F)$
$\Leftrightarrow \bigwedge_{\rho,\tau \in M_A : \tau \sqsubseteq_r \rho} \tau \Rightarrow \rho \models (G \Rightarrow F) \overset{\vee}{\wedge} \bigwedge_{\rho,\tau \in M_A : \tau \sqsubseteq_r \rho} \tau \Rightarrow \rho \models (G' \Rightarrow F)$
$\Leftrightarrow (F \sqsubseteq_r G \overset{\vee}{\wedge} F \sqsubseteq_r G')$

□

Next, we also need to show an distributivity property similar to

$$A \Rightarrow (B \overset{\wedge}{\vee} B') \Leftrightarrow (A \Rightarrow B \overset{\wedge}{\vee} A \Rightarrow B')$$

for $\sqsubseteq_r$. This is formulated in Lemma 3.2.17.

Lemma 3.2.18. $(F \overset{\wedge}{\vee} F') \sqsubseteq_r (G) \Leftrightarrow (F \sqsubseteq_r G \overset{\wedge}{\vee} F' \sqsubseteq_r G)$

Proof. analogous to Lemma 3.2.17 with no change of logical operator □

We now show that the composition of formulas is monotonic with respect to $\sqsubseteq_r$. The idea is the same as Lemma 6 in [5] but we change the base case of phase (1) with Lemma 3.2.14 and use Lemma 3.2.17 for equivalence (i) and Lemma 3.2.18 for equivalence (ii).

Lemma 3.2.19. *Let F, F', G, G' be Formulas. If $F \sqsubseteq_r F'$ and $G \sqsubseteq_r G'$ then $F; G \sqsubseteq_r F'; G'$*

Proof.

1. We perform an induction on G' with base case $F, F', G, G' \in M_A$ which is proven by Lemma 3.2.14. Let $G' = G'_1 \overset{\wedge}{\vee} G'_2$. By Lemma 3.2.17, $G \sqsubseteq_r G'_1 \overset{\wedge}{\vee} G'_2$ is equivalent to $(G \sqsubseteq_r G'_1 \overset{\vee}{\wedge} G \sqsubseteq_r G'_2)$. By induction hypothesis, we get $(F; G \sqsubseteq_r F'; G'_1 \overset{\vee}{\wedge} F; G \sqsubseteq_r F'; G'_2)$. Again by Lemma 3.2.17 and Definition 2.2.6, we have $F; G \sqsubseteq_r F'; G'_1 \overset{\wedge}{\vee} F'; G'_2 \Leftrightarrow F; G \sqsubseteq_r F'; G'$

2. We perform an induction on F' with previous step as base case: $F, F', G \in M_A$ and G' a formula. Let $F' = F'_1 \overset{\wedge}{\vee} F'_2$. By Lemma 3.2.17, $F \sqsubseteq_r F'_1 \overset{\wedge}{\vee} F'_2$ is equivalent to $(F \sqsubseteq_r F'_1 \overset{\vee}{\wedge} F \sqsubseteq_r F'_2)$. By induction hypothesis, we get $(F; G \sqsubseteq_r F'_1; G' \overset{\vee}{\wedge} F; G \sqsubseteq_r F'_2; G')$. Again by Lemma 3.2.17 and Definition 2.2.6, we have $F; G \sqsubseteq_r F'_1; G' \overset{\wedge}{\vee} F'_2; G' \Leftrightarrow F; G \sqsubseteq_r F'; G'$

3. We perform an induction on G with previous step as base case: $F, G \in M_A$ and F', G' formulas. Let $G = G_1 \overset{\wedge}{\vee} G_2$. By Lemma 3.2.18, $G_1 \overset{\wedge}{\vee} G_2 \sqsubseteq_r G'$ is equivalent to $(G_1 \sqsubseteq_r G' \overset{\wedge}{\vee} G_2 \sqsubseteq_r G')$. By induction hypothesis, we get $(F; G_1 \sqsubseteq_r F'; G' \overset{\wedge}{\vee} F; G_2 \sqsubseteq_r F'; G')$. Again by Lemma 3.2.18 and Definition 2.2.6, we have $F; G_1 \overset{\wedge}{\vee} G_2 \sqsubseteq_r F'; G' \Leftrightarrow F; G \sqsubseteq_r F'; G'$

4. We perform an induction on F with previous step as base case: $F \in M_A$ and F, F', G' formulas. Let $F = F_1 \overset{\wedge}{\vee} F_2$. By Lemma 3.2.18, $F_1 \overset{\wedge}{\vee} F_2 \sqsubseteq_r F'$ is equivalent to $(F_1 \sqsubseteq_r F' \overset{\wedge}{\vee} F_2 \sqsubseteq_r F')$. By induction hypothesis, we get $(F_1; G \sqsubseteq_r F'; G' \overset{\wedge}{\vee} F_2; G \sqsubseteq_r F'; G')$. Again by Lemma 3.2.18 and Definition 2.2.6, we have $F_1; G \overset{\wedge}{\vee} F_2; G \sqsubseteq_r F'; G' \Leftrightarrow F; G \sqsubseteq_r F'; G'$

□

3.3 Antichain Optimization

One significant part of the fixed-point iteration is the vector of formulas. In the worst case, this can be of size $k \cdot 2^k$ with $k = 2^{|Q|^2}$ [5]. We try to reduce the size of the formulas by removing some boxes and clauses with the help of the simulation relations. Unlike for SAT, this reduction is expected to boost performance because every step of the iteration will add more information to the vector of formulas. The minimized formula will contain boxes that are not related by $\Rrightarrow_r$. Therefore, we call this antichain optimization.

Definition 3.3.1 (Antichain). *Let $(D, \leq)$ be a partial order. $C \subseteq D$ is an antichain if any two elements in C are not comparable.*

The optimization is done in two steps. The first step is reduction of clauses by removing subsumed boxes. The second step is reduction of formulas by removing subsumed clauses. This optimization should also preserve logical equivalence in terms of $\Lleftarrow\!\!\Rrightarrow_r$. This is important to preserve the existence of winning strategies.

3.3.1 Implication between Boxes

In a clause $K = \bigvee \rho_i$, sometimes there are i and j such that box ρ_i is simulated by ρ_j. One of them therefore can be removed. In order to do this, we need an algorithm to check subsumption relation between boxes. This algorithm is based on Lemma 3.2.11. Note that the relation $\preccurlyeq_r$ has to be precomputed.

Algorithm 3.3.2 (Implication check between boxes). *Given boxes τ, ρ, and $\preccurlyeq_r$*

(1) for all pair $(q_a, q_b) \in \tau$ check if there is $(q'_a, q'_b) \in \rho$ such that $(q_a, q_b) \preccurlyeq_r (q'_a, q'_b)$

(2) return true iff (1) is true

Algorithm 3.3.2 will be used to check every pair of boxes within a clause to determine which boxes to eliminate. We will later show how this algorithm is used for removing boxes.

3.3.2 Implication between Clauses in CNF

For checking implication between formulas in CNF, we also need to define implication between clauses. This will not only useful for stopping the iteration earlier, but also reducing the size of the formulas. When we encounter two clauses K and L in a formula with $K \Rightarrow_r L$, we can remove L. For this, we need to know how implication between clauses and simulation relations between boxes are connected. This is expressed in Lemma 3.3.3.

Lemma 3.3.3. *Let K,L be clauses. $K \Rightarrow_r L$ if only if for all $\rho \in K$ there is $\tau \in L$ s.t. $\tau \sqsubseteq_r \rho$*

Proof. ($\Leftarrow$) Let e_v be an evaluation for both K and L such that $e_v(K) = true$. Since a clause is a conjunction of negation free boxes, there exists a box $\rho \in K$ such that $v_e(\rho) = true$. Based on the assumption that for ρ, there is $\tau \in L$ such that $\tau \sqsubseteq_r \rho$ and Lemma 3.2.11, $e_v(\tau) = true$. Again, since L is also a conjunction of negation free boxes and $\tau \in L$, τ causes $e_v(L) = true$.
($\Rightarrow$) Assume that there is $\rho \in K$ such that for any $\tau \in L$, $\tau \not\sqsubseteq_r \rho$. We can have an assignment v giving the value $true$ to ρ and $false$ to other boxes such that $e_v(K) = true$. We are then free to assign $false$ to all boxes in L with v so that $e_v(L) = false$. □

Then, based on Lemma 3.3.3, we need an algorithm to check whether $K \Rightarrow L$. This is a naive algorithm where we simply loop through all of the boxes in both clauses. In the worst case, it is quadratic in the number of boxes.

Algorithm 3.3.4 (Implication check between clauses)**.** *Given 2 clauses K and L,*

(1) For every $\rho \in K$, check whether there is $\tau \in L$ s.t. $\tau \sqsubseteq_r \rho$ by using Algorithm 3.3.2

(2) return true iff (1) is true

3.3.3 Implication between Formulas in CNF

Finally, we lift this weaker implication checks on boxes to formulas. This will be used in the fixed-point iteration. In the end, $BF/_{\Leftrightarrow_r}$ will have less equivalence classes than $BF/_{\Leftrightarrow}$. With this, we will have more implication relations between formulas. This will hopefully improve the runtime of the fixed-point iteration as we can stop the fixed-point iteration earlier.

Lemma 3.3.5. *Let F, G be formulas, $F \Rightarrow_r G$ if and only if for all clause $L \in G$ there is $K \in F$ such that $K \Rightarrow_r L$*

Proof. ($\Rightarrow$) Assume there is a clause $L \in G$ such that for all $K \in F$, $K \not\Rightarrow L$, we can have a box assignment v such that $e_v(F) = true$, but $e_v(L) = false$. This leads to a case where $e_v(F) = true$ but $e_v(G) = false$.
($\Leftarrow$) If F evaluates to $true$, then all clauses in F evaluates to $true$. Since we assume that for any clause $L \in G$ there is $K \in F$ such that $K \Rightarrow L$, it means all clauses in G evaluates to $true$. Therefore, $F \Rightarrow G$ □

Now, we design the algorithm to check whether a formula implies another formula based on Lemma 3.3.5. We will use this to stop the fixed-point iteration based on the simulation relations.

Algorithm 3.3.6 (Implication check between formulas)**.** *Given 2 formulas F and G,*

(1) For every $L \in G$, check whether there is $K \in F$ s.t. $K \Rightarrow L$ by using Algorithm 3.3.4

(2) return true iff (1) is true

3.3.4 Minimization of Formulas

In the previous section, we have weakened the notion of implication so that we have a smaller number of equivalence classes. Now, we want to further improve the fixed-point iteration algorithm by performing formula reductions. Given a formula F, we want to minimize it into $min(F)$ such that $min(F) \Leftrightarrow F$ and the size of $min(F)$ is smaller than F. This will hopefully improve the runtime, because in every step of the fixed-point iteration algorithm, new information will be added to the current vector of formulas [5]. More specifically, the implication check for smaller formulas is cheaper and minimization in an iteration will affect the runtime of implication checks afterwards.

Algorithm 3.3.7 (Formula minimization)**.** *Given a formula F and an NFA A,*

(1) Precompute $\preccurlyeq_r$ for A

(2) For every clause $C \in F$, remove box $\tau \in C$
if, based on Lemma3.2.11, there is a box $\rho \in C$ such that $\tau \Rightarrow_r \rho$.

(3) Remove clauses $L \in F$
if, based on Lemma 3.3.3, there is a clause $K \in F$ such that $K \Rightarrow_r L$

For correctness, one important thing that has to be satisfied is logical equivalence between a formula and its minimized version. If a formula F and its minimization $min(F)$ are logically equivalent, any assignment v will make both formulas evaluate to $true$. Therefore, the winner of a context-free game instance will remain the same whether we use F or $min(F)$.

Lemma 3.3.8. *Given a formula F and its minimization $min(F)$, $min(F) \Leftrightarrow_r F$*

Proof. **((1) box elimination)** Given $\tau \Rightarrow_r \rho$, we show that $\tau \vee \rho \Leftrightarrow_r \rho$.

($\Leftarrow$) clear.

($\Rightarrow$) Assume $\rho = false$, then $\tau = false$ because $\tau \Rightarrow_r \rho$. So, $\tau \vee \rho = false$

((2) clause elimination) Given $K \Rightarrow_r L$, $K \wedge L \Leftrightarrow_r K$,

($\Rightarrow$) clear.

($\Leftarrow$) $K \wedge (K \Rightarrow_r L) \Leftrightarrow K \wedge (\neg K \vee L) \Leftrightarrow (K \wedge \neg K) \vee (K \wedge L) \Leftrightarrow K \wedge L$

□

Chapter 4

Experiments

4.1 Implementation

For the implementation of context-free games, we use the c++ program written in the Concurrency Theory Group at TU Kaiserslautern. The original program is not one of the contributions of this thesis. This program can already handle context-free game instances with some optimizations. For our purpose, we develop additional subroutines for the precomputations, implication checks, and formula minimizations.

When using our optimizations, we also introduce extra computational costs during the fixed-point iteration. These costs mainly come from the implication checks between boxes. Therefore, in order to improve the runtime, we need to use some techniques that are not necessarily clear from the theory part. These include using efficient data structure and memory management.

4.1.1 Program

The program works by generating instances of context-free games, solving it, and checking its correctness. There are already some solvers and optimizations and we use the CNF based solver as a baseline. We will later compare our optimization results with this solver.

The instance generation algorithm is based on the Tabakov-Vardi model [10]. When the generation is done naively, we will have an NFA with unreachable states and states that do not reach final states. Thus, the size of an NFA may be a misleading indicator for the context-free games. Moreover, some types of computations will be more expensive. Therefore, the implementation was modified. We will explore this in the next subsection.

4.1.2 Instance Generation

Instance generation is not the scope of this thesis. However, due to its importance, we try explore the implementation. The generation of NFAs works by iteratively creating transitions from a uniformly random reachable states to a new state. This guarantees that the NFA is connected. This happens in lines 5-11 in Code 4.1. Then, new random transitions are added with some probability. This is done in lines 13-21. In our case, the probability is 0.5. Line 25 sets state 0 to be the initial state. Additionally, we can also have multiple initial states. However, in our experiment, we only fix one initial state. Lastly, a state will be set to a final state based on Bernoulli distribution with probability 0.5.

Code 4.1: NFA generation

```
1 unique_ptr<NFA<size_t,size_t>> generateFixedTransitionNFA(size_t alphabetSize, size_t states, size_t transitions, unsigned int flags, double finalStateProbability)
2 {
3  unique_ptr<NFA<size_t,size_t>> nfa(new MatrixNFA(states, alphabetSize));
4  double transitionProbability = 0.5;
5  if (flags & GEN_CONNECTED){
6   for (size_t to = 1; to < states; to++){
7    size_t from = uniform_int_distribution<size_t>(0, to-1)(generator);
```

```
      size_t letter = uniform_int_distribution<size_t>(0, alphabetSize-1)(generator);
      nfa->createTransition(from, letter, to);
    }
  }

  RandomSetGenerator setgen(alphabetSize*states*states);
  for (size_t i = (flags & GEN_CONNECTED) ? states-1 : 0; i < transitions && !setgen.empty(); i++){
    size_t x = setgen.draw(generator);
    size_t from = x%states;
    x /= states;
    size_t to = x%states;
    x /= states;
    size_t letter = x;
    nfa->createTransition(from, letter, to);
  }

  //state 0 is always initial
  nfa->setStateInitial(0);
  if (flags & GEN_MULTIPLE_INITIAL){
    double initialStateProbability = 0.5;
    for (size_t i = 1; i < states; i++){
      nfa->setStateInitial(i, bernoulli_distribution(initialStateProbability)(generator));
    }
  }

  for (size_t i = 0; i < states; i++){
    nfa->setStateFinal(i, bernoulli_distribution(finalStateProbability)(generator));
  }
  return nfa;
}
```

Next, we explore how a context-free grammar is generated. This is also according to the Tabakov-Vardi model [10]. The input of the algorithm is the number of terminals, nonterminals, and productions. Firstly, we produce transitions to epsilon from the set of nonterminals and set the owner by using bernoulli distribution with probability 0.5. This occurs in line 4 Code 4.2. This is a first step to guarantee that a terminal word can be derived from any nonterminals. Then, lines 7-21 are for generating the production rules. Everytime a production rule for a nonterminal is added, the algorithm makes sure that the nonterminals on the right hand side can already derive terminal words. The variable x sums up this choice. x encodes both terminals and two nonterminals. If x is less than terminals, then x represents a terminal symbol. If x==terminals, then x represents an epsilon. If x is greater than terminals, then x%i represents the first nonterminal in the RHS and x/i represents the second one. Lastly, line 31-49 makes sure that the number of production rules equals some predetermined input production_count. In our case, the default value for the number of production rules is twice the number of nonterminals [9].

Code 4.2: CFG generation

```
1 unique_ptr<Grammar> generateGrammarChomsky(size_t terminals, size_t nonterminals, size_t production_count, unsigned int flags){
2  vector<Grammar::ntspec> ntspecs;
3  for (size_t i = 0; i < nonterminals; i++){
4   ntspecs.push_back({bernoulli_distribution(0.5)(generator) ? Grammar::ntspec::PROVER : Grammar::ntspec::REFUTER, {}});
5  }
6  if (flags & GEN_NONEMPTY){
7   for (size_t i = 0; i < nonterminals; i++){
8    size_t x = uniform_int_distribution<size_t>(0, terminals+i*i)(generator);
9    vector<Grammar::rhs> production;
10   if (x < terminals){
11    production = {{Grammar::rhs::TERMINAL, x}};
12   }else if (x == terminals){
13    production = {};
14   }else{
15    x -= terminals+1;
```

```
      size_t n1 = x%i;
      x /= i;
      size_t n2 = x;
      production = {{Grammar::rhs::NONTERMINAL, n1}, {Grammar::rhs::NONTERMINAL, n2}};
    }
    ntspecs[i].productions.push_back(production);
  }
  if (production_count >= nonterminals)
    production_count -= nonterminals;
  else
    production_count = 0;
  }
  if (flags & GEN_CONNECTED){
  }

  RandomSetGenerator setgen(nonterminals*(terminals+1+nonterminals*nonterminals));
  for (size_t i = 0; i < production_count && !setgen.empty(); i++){
    size_t x = setgen.draw(generator);
    size_t from = x%nonterminals;
    x /= nonterminals;
    vector<Grammar::rhs> production;
    if (x < terminals){
      production = {{Grammar::rhs::TERMINAL, x}};
    }else if (x == terminals){
      production = {};
    }else{
      x -= terminals+1;
      size_t n1 = x%nonterminals;
      x /= nonterminals;
      size_t n2 = x;
      production = {{Grammar::rhs::NONTERMINAL, n1}, {Grammar::rhs::NONTERMINAL, n2}};
    }
    ntspecs[from].productions.push_back(production);
  }

```

```
unique_ptr<Grammar> grammar(new Grammar(ntspecs,
    uniform_int_distribution<size_t>(0, nonterminals-1)(
    generator)));
return grammar;
}
```

4.1.3 Implementation Tricks

Our set of optimizations introduce additional computational costs for the fixed-point iteration. This may affect the overall runtime. Therefore, we carefully develop the subroutines for our solvers by using some implementation tricks to reduce the costs.

Firstly, we want to avoid recomputing box relations. As we will later explore, checking simulation relations takes some extra computational costs due to its arcs evaluations. Moreover, in the fixed-point iteration, we may want to know the relation between two boxes that were previously computed. So, instead of recomputing it, we store the result of the first check of two boxes in some sort of hash map. The immediate structure that we could use for its key is the pair of boxes itself. However, hashing a complex structure such as boxes is still a costly operation. Therefore, we assign a unique id to a box and use this id for hashing. In order to do this, we use DummyBox class. This class is also from the original program. Then, we make a map with a pair of indices as key and Boolean value as relation flag.

The source code for implication check where we avoid recomputing simulation relations between boxes is Code 4.3. In line 1, we have a variable implicationMap to store the implication relations between boxes. The key is a pair variable of indices for two boxes. Before any other advanced checks, we first check whether the boxes are equal. This is done by checking equality between the indices of the two boxes. Then, we start with accessing box relations that were already stored in implicationMap. Lines 6-10 are used for this. If box1 implies box2 then implicationMap.at(make_pair(box2.getId(),box1.getId()))

will return *true*. Otherwise, the execution continues to line 11. This will call one of the simulation relation checks explained in the next 4 subsections. The variable type will determine whether the relation is inclusion, forward, backward, or forward-backward simulation relations.

Code 4.3: General mapping trick for implication between boxes

```
1  unordered_map< pair<size_t,size_t>, bool>implicationMap;
2  bool implies(const DummyBox &box1, const DummyBox &box2)
      override {
3   if(box1.getId()==box2.getId()){
4    return true;
5   }
6   try{
7    bool s=implicationMap.at(make_pair(box2.getId(),box1.getId
        ()));
8    return s;
9   }catch(const out_of_range &e){
10  }
11  bool s=simulationRelation.isSimulatedBy(this->getActualBox(
       box2), this->getActualBox(box1), type,false);
12  implicationMap[make_pair(box2.getId(),box1.getId())]=s;
13  return s;
14 }
```

Secondly, we take advantage of the fact that the four simulation relations for boxes are related. As the first observation, inclusion is a sub-poset of both forward and backward simulation relations. Both forward and backward simulation relations are sub-posets of forward-backward simulation relations. When checking a simulation relation, we first perform checks on all of the direct subposets of the simulation relation. Before checking forward and backward simulation relation, we first try to check the inclusion relation. Also, before checking forward-backward simulation relation, we first check both forward and backward simulation relations. As an illustration, given an arc (q_a, q_b) in some box ρ, we can skip a whole loop for searching forward simulating arc (q_a, q'_b) in another box τ, if we can ascertain that

$q_b = q'_b$. This is the condition for subset relation.

Thirdly, whenever possible, we take advantage of pointers. This is necessary because we use c++. Whenever we iterate over a complex object such as list of boxes, we need to avoid copying the object because we will get extra computational cost. We do this especially in the formula reduction part where we have to iterate over clauses and boxes. However, for objects using size_t as data type, we simply use it as it is because it is already a data type suitable for indexing. Some of them are states, alphabets, and nonterminals.

4.1.4 Inclusion Check

The inclusion check is used for implication check when using inclusion relation. We have a naive implementation where for any arc in the first box b1, we check in constant time whether this arc also exists in b2. This means that in the worst case, we have to check every arcs in box b1. This yields an algorithm running in quadratic time with respect to the number of states. In practice, this can be more expensive than equality check between boxes. The extra costs come from the weakening of the relation. if an arc does not exist in b1 but exists in b2, we need to continue searching for inclusion. But, for equality check, we can already stop the search.

Code 4.4: Subset relation check between boxes

```
bool isSimulatedBy(const Box &b1, const Box &b2, simtype t)
    {
 if(t==inclusion){
  bool simulated=true;
  for (size_t from = 0; from < n->stateCount(); from++) {
   for (size_t to= 0; to < n->stateCount(); to++) {
    if(!b1.lineExist(from,to)){continue;}
    if(b1.lineExist(from,to) && !b2.lineExist(from,to)){
     //counter example found
```

```
      simulated=false;
      break;
     }
    }
    if(!simulated){
     //outer break
     break;
    }
   }
   return simulated;
  }
}
```

4.1.5 Forward Simulation Relation Check

Implication check with forward simulation relation is more expensive than with inclusion relation. The algorithm runs in $|Q|^3$ in the worst case. This is one order higher than inclusion relation. The extra costs are due to the search of forward simulating state in b2. As previously explained, inclusion relation is a sub poset of forward simulation relation. It means that if box b1 is a subset of box b2, then box b1 is also forward simulated by box b2. We implicitly check for this inclusion in line 7 Code 4.5. When this condition is violated, it means the arc (b1from,b1to) in b1 also exist in b2. Therefore, we skip line 10-16 and continue testing other arcs in b1.

The simulation relation check between arcs itself is in line 13. This is based on Definition 3.2.3. For arc (b1from,b1to), we check whether there is a state b2to, such that (b1from,b2to) $\in$ b2 and b1to $\preccurlyeq_f$ b2to. If this is not satisfied, then we find a proof that the two boxes are not related. So, we can break the whole loop.

Code 4.5: Forward simulation relation check between boxes

```
1 bool isSimulatedBy(const Box &b1, const Box &b2, simtype t){
2 if(t==forward){
```

```
  bool simulated=true;
  //for all lines in b1
  for (size_t b1from = 0; b1from < n->stateCount(); b1from++) {
   for (size_t b1to = 0; b1to < n->stateCount(); b1to++) {
    if(!b1.lineExist(b1from,b1to)){continue;}
    //find a line simulating the line in b1
    if(b1.lineExist(b1from,b1to) && !b2.lineExist(b1from,b1to)){
     bool found=false;
     for (size_t b2to = 0; b2to < n->stateCount(); b2to++) {
      //check condition of forward simulation for lines
      found=b2.lineExist(b1from,b2to) && isSimulatedBy(b1to,b2to,forward);
      if(found){
       break;
      }
     }
     //counter example found
     if(!found){
      simulated=false;
      break;
     }
    }
   }
   if(!simulated){
    //outer break
    break;
   }
  }
  return simulated;
}
```

4.1.6 Backward Simulation Relation Check

Just as forward simulation relation, implication check with backward simulation relation is also more expensive than with inclusion relation. The algorithm runs in $|Q|^3$ in the worst case. With the same reasoning, The extra costs come from the search of backward simulating state in b2. Again, we implicitly check for inclusion in line 7 Code 4.6.

The structure of Code 4.6 is not very different from that of forward simulation relation. What changes is the loop in line 11-17. In particular, we use Lemma 3.2.4 for line 13. Again, If this is not satisfied, then we find a proof that the two boxes are not related and we can break the whole loop.

Code 4.6: Backward simulation relation check between boxes

```
1  bool isSimulatedBy(const Box &b1, const Box &b2, simtype t) {
2   if(t==backward){
3    bool simulated=true;
4    //for all lines in b1
5    for (size_t b1from = 0; b1from < n->stateCount(); b1from++) {
6     for (size_t b1to = 0; b1to < n->stateCount(); b1to++) {
7      if(!b1.lineExist(b1from,b1to)){continue;}
8      //find a line simulating the line in b1
9      if(b1.lineExist(b1from,b1to) && !b2.lineExist(b1from, b1to)) {
10      bool found = false;
11      for (size_t b2from = 0; b2from < n->stateCount(); b2from++) {
12       //check condition of backward simulation for lines
13       found = b2.lineExist(b2from, b1to) && isSimulatedBy(b1from, b2from, backward);
14       if (found) {
15        break;
16       }
17      }
18      //counter example found
19      if (!found) {
20       simulated = false;
21       break;
22      }
23     }
24    }
```

```
    if(!simulated){
     //outer break
     break;
    }
   }
  return simulated;
 }
}
```

4.1.7 Forward-Backward Simulation Relation Check

The implication check with forward-backward simulation relation includes the extra costs from both forward and backward simulation relations. This increases the order of the complexity to $|Q|^4$ in the worst case. Again, with the same implementation trick, in line 2 Code 4.7, we check first for forward simulation relation and backward simulation relation. If either one of them is satisfied, then b1 is forward-backward simulated by b2. Otherwise, the execution moves on to search for forward-backward simulating arc in b2 that is neither forward nor backward simulating arc in line 12-26. When this happens, we also make sure that forward and backward simulation relations are not rechecked by adding line 16. b1from==b2from filters out forward simulation relations and b1to==b2to filters out backward simulation relations.

Code 4.7: Backward simulation relation check between boxes

```
1 bool isSimulatedBy(const Box &b1, const Box &b2, simtype t)
     {
2  if(t==forwardbackward){
3   if(isSimulatedBy(b1,b2,forward)||isSimulatedBy(b1,b2,
       backward))return true;
4   bool simulated=true;
5   //for all lines in b1
6   for (size_t b1from = 0; b1from < n->stateCount(); b1from
       ++) {
7    for(size_t b1to = 0; b1to < n->stateCount(); b1to++) {
```

```
      if (!b1.lineExist(b1from, b1to)) {
       continue;
      }
      if(b1.lineExist(b1from,b1to) && !b2.lineExist(b1from,
         b1to)) {
       //find a line simulating the line in b2
       bool found = false;
       for (size_t b2from = 0; b2from < n->stateCount();
          b2from++) {
        for (size_t b2to = 0; b2to < n->stateCount(); b2to++)
           {
         if(b1from==b2from ||b1to==b2to){continue;}
         found = b2.lineExist(b2from, b2to) && isSimulatedBy(
            b1to, b2to, forward) && isSimulatedBy(b1from,
            b2from, backward);
         if (found) {
          break;
         }
        }
        if (found) {
         //outer break
         break;
        }
       }
       if (!found) {
        simulated = false;
        break;
       }
      }
     }
     if(!simulated){
      //outer break
      break;
     }
    }
    return simulated;
  }
}
```

4.1.8 Precomputation

Implication checks based on simulation relations other than inclusion relation rely on the simulation relations on states $\preccurlyeq_r$. These need to be precomputed. For both the forward and backward simulation relations, we use a naive algorithm. We first explain the precomputation for the forward simulation relations in Code 4.8. We first initialize all relations between states to be true. This happen in line 3-7 Code 4.8. Next, we eliminate the relations that do not satisfy Definition 3.2.1 until there is no change any more. The while loop starting from line 10 keeps track if there is change. Within the loop, we check whether every pair of states s1 and s2 satisfy forward simulation condition. This is done by making sure that for any outgoing transitions from s1 to state bS1, there is transition with similar letter l going out of s2 into state nS2 which forward simulates nS1. Moreover, if s1 is a final state, then s1 is also a final state. This is all checked in line 14-28. The relation, at this point, can only change from true to false. This check is reflected in line 29-31.

Code 4.8: Precomputation for forward simulation relations

```
1  void precomputeF(){
2
3   for (size_t from = 0; from < n->stateCount(); from++) {
4    for (size_t to = 0; to < n->stateCount(); to++) {
5     simulations[simulationIndex(from, forward, to)] = true;
6    }
7   }
8
9   bool changed = true;
10  while (changed) {
11   changed = false;
12   for (size_t s1 :n->getAllStates()) {
13    for (size_t s2 : n->getAllStates()) {
14     //Forward simulation
15     //for all outgoing transition from s1
16     vector<Transition> outTransS1 = n->getOutTransitions(s1
         );
```

```
    for (Transition tr : outTransS1) {
    if (!simulations[simulationIndex(s1, forward, s2)]) { break; }
     size_t l = tr.letter;
     size_t nS1 = tr.to;
     vector<size_t> nextSS2 = n->applyTransition(s2, l);
     //find nS2 such that nS1 is simulated by nS2
     bool found = false;
     for (size_t nS2:nextSS2) {
      //forward simulation condition
      found = simulations[simulationIndex(nS1, forward, nS2)] && (!(n->containsFinalState({s1})) || (n->containsFinalState({s2})));
      if (found) {break;}
     }
     if (simulations[simulationIndex(s1, forward, s2)] != found) {
      simulations[simulationIndex(s1, forward, s2)] = found;
      changed = true;
     }
    }
   }
  }
 }
}
```

The precomputation of the backward simulation relations between states is quite similar to the precomputation of the forward simulation relations between states. We can see this in Code 4.9. The first difference is that it searches through incoming transitions for a violation instead of outgoing transitions. Line 16 and 21 is for getting incoming transitions to s1 and s2 respectively. The second difference is that instead of final states, we use initial state. This is due to Definition 3.2.2.

Code 4.9: Precomputation for backward simulation relations

```
void precomputeB(){

 for (size_t from = 0; from < n->stateCount(); from++) {
  for (size_t to = 0; to < n->stateCount(); to++) {
   simulations[simulationIndex(from, backward, to)] = true;
  }
 }

 bool changed = true;
 while (changed) {
  changed = false;
  for (size_t s1 :n->getAllStates()) {
   for (size_t s2 : n->getAllStates()) {
    //backward simulation
    //for all incoming transition from s1
    vector<Transition> inTransS1=n->getInTransitions(s1);
    for(Transition tr : inTransS1){
     if(!simulations[simulationIndex(s1, backward, s2)]){
        break;}
     size_t l=tr.letter;
     size_t pS1=tr.from;
     vector<size_t> pStateS2 = n->applyTransitionBackward(
        s2, l);
     //find pS2 such that pS1 is simulated by pS2
     bool found=false;
     for (size_t pS2:pStateS2) {
      //backward simulation condition
      found = simulations[simulationIndex(pS1, backward,
         pS2)] && (!(n->isInitialState(s1)) || (n->
         isInitialState(s2)));
      if(found){break;}
     }
     if (simulations[simulationIndex(s1, backward, s2)] !=
        found){
      simulations[simulationIndex(s1, backward, s2)] =
         found;
      changed = true;
     }
    }
```

```
    }
  }
 }
}
```

4.1.9 Formula Reduction

Finally, we implement formula reduction. The first step is to reduce each clause. Again, we do this naively by checking each pair of distinct boxes tau and rho within a clause K and checking whether tau implies rho. If implication holds, we would immediately remove tau so that it will not be checked again in the next iteration. The choice to remove tau is due to Lemma 3.3.8. This algorithm is quadratic in the size of the clause.

An additional trick to improve the performance is by using pointer when iterating over the boxes. This keeps us from copying boxes from the clause everytime we access a box over the iteration. Also, since the two nested loops starting from line 4 and line 7 are actually looping over the same object atoms, we add !(*tau == *rho) as a condition alongside the implication check between boxes in line 8. This is because a box implies itself by reflexivity. If we do not have this additional condition, any box will definitely be eliminated because it has implication relation with itself.

Code 4.10: Clause Reduction

```
1 bool reduce(BoxManager<Box, size_t, size_t> *boxManager) {
2  bool reduced=false;
3  typename std::unordered_set<Box>::iterator tau = atoms.
     begin();
4  while (tau != atoms.end()) {
5   typename std::unordered_set<Box>::iterator rho = atoms.
      begin();
6   bool isRemoved=false;
7   while (rho != atoms.end()) {
8    if (boxManager->implies(*tau, *rho) && !(*tau == *rho))
       {
```

```
    isRemoved=true;
    break;
   } else {
    rho++;
   }
  }
  if(isRemoved){
   reduced=true;
   tau = atoms.erase(tau);
  }else{
   tau++;
  }
 }
 return reduced;
}
```

The second step of the formula reduction is clause elimination. This clause elimination is also based on Lemma 3.3.8. We perform this just like box elimination. For any two distinct clauses K and L in a formula, we check whether K implies L. Here, what we remove is L instead of K. We combine this clause elimination with clause reduction in Code 4.10. We perform this in line 2-7. Clause reduction precedes clause elimination because, in this way, the cost of clause elimination will be reduced.

Code 4.11: Formula Reduction

```
1  bool reduce(BoxManager<Box, size_t, size_t> *boxManager){
2   bool reduced=false;
3   for (Clause c : clauses) {
4    if(c.reduce(boxManager)){
5     reduced=true;
6    }
7   }
8
9   typename std::unordered_set<Clause, typename Clause::
       Hasher>::iterator L = clauses.begin();
10  while (L != clauses.end()) {
11   typename std::unordered_set<Clause, typename Clause::
```

```
      Hasher>::iterator K = clauses.begin();
   bool isRemoved=false;
   while (K != clauses.end()) {
    if(K->implies(*L,boxManager) && !(K==L)){
     isRemoved=true;
     break;
    };
    K++;
   }
   if(isRemoved){
    reduced=true;
    L=clauses.erase(L);
   }else{
    L++;
   }
  }
  return reduced;
}
```

The result of both precomputations are used for simulation relation checks between boxes. Different simulation relations between boxes need them differently. Firstly, inclusion check does not need any precomputation. Forward and backward simulation relation checks need forward precomputation and backward precomputation respectively. Lastly, forward-backward simulation relation check needs both of the precomputations.

4.2 Simulation Setup

In our experiments, 20 instances of context-free games are randomly generated by using Tabakov-Vardi model [10] and we perform fixed-point iteration on them by using worklist based Kleene iteration algorithm. We do this for a parameter set. For each instance, we run 13 solvers. As our baseline solver, we use a CNF Solver without any simulation relations or reductions. For this solver, implication check between boxes only amounts to checking box equality. The checks also take advantage of the box indexing trick we explain in Section 4.

We also compare the results when using different parameter sets. The parameter sets reflect the size of the instances. These include the number of states, the number of alphabets, and the number of nonterminals.

Table 4.1: Solver list. The reduction is done for k=1,2,4.

Solver	**Description**
CNF Baseline	CNF Solver without any simulation relations or reductions.
CNF SRD inclusion	CNF Solver with inclusion relation.
CNF SRD forward	CNF Solver with forward simulation relation.
CNF SRD backward	CNF Solver with backward simulation relation.
CNF SRD fb	CNF Solver with forward-backward simulation relation.
CNF SRD inclRed**k**	CNF Solver with inclusion relation and reduction for every k iteration.
CNF SRD forward**k**	CNF Solver with forward simulation and reduction for every k iteration.
CNF SRD backward**k**	CNF Solver with backward simulation and reduction for every k iteration.
CNF SRD fb**k**	CNF Solver with forward-backward simulation relation and reduction for every k iteration.

When we compare between different techniques for optimizations, it may happen that some of them have timeout and some others don't. This presents a problem when comparing the runtimes. So, we decided to only take into account the instances for which all compared techniques can finish within the time bound for the aggregated results.

Table 4.2: Parameter list.

Parameter	Description	values
Q	The number of states	{10,30}
Σ	The number of alphabets	{2,20}
N	The number of nonterminals	{10,30}

The choice of parameters is based on trial and error. We want to have a small value and a large value for each parameter, but we are limited by the power of the machine. The experiments are run on Intel i7-4700HQ, 2.39 GHz. For the number of states, the smallest number that we could get such that there is a non-reflexive forward-backward simulation relation is 6. However, we increase it to 10 to increase the likelihood that we get such relations. The smallest number of alphabets we could reasonably use is 2. For the small number of nonterminals, we use 10. In contrast, we determine the large values of the parameters simply by picking a number and reducing it if it took a long time or our machine simply stopped responding.

4.3 Benchmarking Results

In this section, we try to see whether our weaker notion of implication has any effects on the performance of fixed-point iteration for context-free games. After running the experiments, the results show that the performance of our solvers depend very much on the instances generated according to the Tabakov-Vardi model [10]. So, we try to analyze it by picking some instances which reflect different outcomes. Table 4.3 shows the quantities that we measure for each instance.

Table 4.3: Column descriptions for result tables.

Column Name	Description	Aggregation
No	Numbering of instance. We can use this to track an instance result in this section to its position in the Appendix.	Union
Q/Σ/N	Q is the number of states, Σ is the number of t, and N is the number of nonterminals.	Key
Solver	The name of the solver	Key
time	The time needed to solve the instance	Avg
time%	The time in percent relative to baseline CNF solver's time.	Avg
ftime	The time needed for precomputation of the forward simulation relations between states.	Avg
btime	The time needed for precomputation of the backward simulation relations between states.	Avg
ic	It consists of two numbers x/y. y represents the total number of implication check. x represents the number of implication check returning true.	Sum of x/Sum of y.
fu	The number of formula updates in the fixed-point iteration.	Sum
fr	The number of successful formula reduction based on the simulation relations.	Sum
fc	The number of forward simulation relations between states.	Sum
bc	The number of backward simulation relations between states.	Sum
to	A flag indicating timeout.	Sum

4.3.1 Improvement

The first instance that we want to show is the one successfully improved by the simulation relations. Besides decreasing the runtime, the number of formula updates and implication checks are also reduced.

Table 4.4: One of the instances with improved running time due to simulation relations.

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc
51	30/2/10	CNF Baseline	20.27ms	100%	-	-	21/26	5	0	-	-
		CNF SRD inclusion	8.90ms	44%	-	-	17/21	4	0	-	-
		CNF SRD forward	2.93ms	14%	120.02ms	-	13/16	3	0	654	-
		CNF SRD backward	7.83ms	39%	-	74.71ms	17/21	4	0	-	0
		CNF SRD fb	2.55ms	13%	116.90ms	74.25ms	13/16	3	0	654	0
		CNF SRD inclRed1	8.18ms	40%	-	-	17/21	4	3	-	-
		CNF SRD fRed1	2.74ms	14%	117.90ms	-	13/16	3	2	654	-
		CNF SRD bRed1	9.66ms	48%	-	78.07ms	17/21	4	3	-	0
		CNF SRD fbRed1	3.25ms	16%	117.26ms	80.92ms	13/16	3	3	654	0
		CNF SRD inclRed2	7.78ms	38%	-	-	17/21	4	3	-	-
		CNF SRD fRed2	2.77ms	14%	112.59ms	-	13/16	3	2	654	-
		CNF SRD bRed2	7.50ms	37%	-	63.64ms	17/21	4	3	-	0
		CNF SRD fbRed2	3.18ms	16%	122.47ms	80.93ms	13/16	3	2	654	0
		CNF SRD inclRed4	8.86ms	44%	-	-	17/21	4	1	-	-
		CNF SRD fRed4	3.03ms	15%	120.92ms	-	13/16	3	0	654	-
		CNF SRD bRed4	9.08ms	45%	-	86.19ms	17/21	4	1	-	0
		CNF SRD fbRed4	2.76ms	14%	109.63ms	72.58ms	13/16	3	0	654	0

Table 4.4 shows the results of one of the instances whose runtime is improved by simulation relations. The baseline CNF solver without simulation relations requires 26 implication checks and 5 formula updates. In contrast, inclusion relation reduced it to 21 and 4, respectively. Forward simulation relation improves it further to 16 implication checks and 3 formula updates. For this instance, the forward simulation relation helps to speed up the fixed-point iteration to be 7 times faster than baseline CNF solver.

Table 4.6: One of the instances with improved running time due to reductions.

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc
		CNF Baseline	9.19ms	100%	-	-	19/27	8	0	-	-
		CNF SRD inclusion	8.50ms	92%	-	-	19/27	8	0	-	-
		CNF SRD forward	7.13ms	78%	340.80ms	-	19/27	8	0	646	-
		CNF SRD backward	7.82ms	85%	-	220.95ms	19/27	8	0	-	483
		CNF SRD fb	7.05ms	77%	335.38ms	222.35ms	19/27	8	0	646	483
		CNF SRD inclRed1	7.07ms	77%	-	-	19/27	8	3	-	-
		CNF SRD fRed1	6.77ms	74%	321.58ms	-	19/27	8	3	646	-
		CNF SRD bRed1	6.85ms	74%	-	228.35ms	19/27	8	3	-	483
49	30/2/10	CNF SRD fbRed1	6.78ms	74%	285.64ms	210.72ms	19/27	8	4	646	483
		CNF SRD inclRed2	6.01ms	65%	-	-	19/27	8	3	-	-
		CNF SRD fRed2	6.04ms	66%	297.32ms	-	19/27	8	3	646	-
		CNF SRD bRed2	5.80ms	63%	-	196.55ms	19/27	8	3	-	483
		CNF SRD fbRed2	6.02ms	65%	285.43ms	200.97ms	19/27	8	3	646	483
		CNF SRD inclRed4	5.78ms	63%	-	-	19/27	8	1	-	-
		CNF SRD fRed4	6.68ms	73%	299.33ms	-	19/27	8	1	646	-
		CNF SRD bRed4	5.47ms	59%	-	189.74ms	19/27	8	1	-	483
		CNF SRD fbRed4	5.93ms	65%	285.12ms	198.13ms	19/27	8	1	646	483

Another important advantage of the simulation relations is that they may still improve runtime when there is no reduction in the number of formula updates and implication checks. The improvement comes from formula reductions. Formula reductions can help to reduce the runtime of formula compositions. This is due to the reduction in the size based on Algorithm 3.3.7. Table 4.6 shows the resulting speed ups. The runtime with forward simulation relation is 92% of the baseline. When using reduction every two iterations, it speeds up to 65%.

However, this advantage occurs rarely in our experiments. In most cases, reductions rarely give significant speed up to our simulation relation based solvers. In order to figure out why, we try to assess the performance of the reductions by keeping track of the number of successful reductions. The column *fr* shows the values for this. It is calculated by increasing a counter whenever an attempt to reduce a formula produces a strictly smaller one. This apparently does not correlate with the runtime. Sometimes, smaller numbers of successful reductions give better results and sometimes it is the other way around.

Therefore, we propose other ways to assess the performance of our reduction algorithm as future work. Firstly, we can record the formula size ratios after and before a reduction. Then, we average these numbers over the number of reduction attempts. This reflects the overall size decrease of formulas throughout the fixed-point iteration. Secondly, we can keep track of the positions of successful formula reductions within the fixed-point iteration. The idea is that early reductions matter more than the later ones. This is also due to the monotonicity property. If we compose two related boxes with another box, then we get new pair of boxes that are still related. In other words, relations are carried until the end of the iteration. An early reduction will ease the burden of the many later implication checks and formula compositions.

Table 4.6 also shows that even without reduction in the number of implication checks, a simulation relation based solver can still perform better. This is apparently a rare case in our experiments. Checking implication in general is more costly when using simulation relations.

4.3.2 Shortcomings

The simulation relation based solvers take advantage of the structure of the instances. Both the NFAs and the CFGs matter. However, the Tabakov-Vardi instance generation algorithm we explained in Subsection 4.1.2 can not guarantee that the instances we get, have the properties we want for the solvers. Particularly, the performance of a simulation relation based solver depends on the existence of simulating pairs of boxes in the NFA and the occurrences of these related boxes in the formulas within the fixed-point iteration. In other words, many instances may be not optimizable. This may result in the simulation relation based solvers having worse performance than the baseline solver.

Table 4.8: One of the instances with worse running time than baseline.

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc
105	10/20/30	CNF Baseline	4.83ms	100%	-	-	51/72	21	0	-	-
		CNF SRD inclusion	4.81ms	100%	-	-	51/72	21	0	-	-
		CNF SRD forward	6.16ms	127%	27.25ms	-	51/72	21	0	0	-
		CNF SRD backward	7.08ms	147%	-	11.73ms	51/72	21	0	-	0
		CNF SRD fb	6.57ms	136%	35.74ms	10.04ms	51/72	21	0	0	0
		CNF SRD inclRed1	7.00ms	145%	-	-	51/72	21	4	-	-
		CNF SRD fRed1	6.84ms	141%	35.72ms	-	51/72	21	4	0	-
		CNF SRD bRed1	6.64ms	137%	-	11.55ms	51/72	21	4	-	0
		CNF SRD fbRed1	13.94ms	288%	40.49ms	13.21ms	51/72	21	12	0	0
		CNF SRD inclRed2	8.72ms	180%	-	-	51/72	21	4	-	-
		CNF SRD fRed2	6.74ms	139%	43.74ms	-	51/72	21	4	0	-
		CNF SRD bRed2	6.79ms	140%	-	11.23ms	51/72	21	4	-	0
		CNF SRD fbRed2	8.73ms	181%	40.35ms	12.96ms	51/72	21	4	0	0
		CNF SRD inclRed4	7.77ms	161%	-	-	51/72	21	0	-	-
		CNF SRD fRed4	6.75ms	140%	38.54ms	-	51/72	21	0	0	-
		CNF SRD bRed4	6.94ms	144%	-	11.17ms	51/72	21	0	-	0
		CNF SRD fbRed4	7.10ms	147%	37.58ms	11.36ms	51/72	21	0	0	0

Table 4.8 illustrates an instance where the baseline CNF solver performs better than all of the simulation relation based solvers. The instance even has a large number of forward simulation relations between states. This does not guarantee the existence of simulating boxes.

4.3.3 Timeout and Memory Consumption

We set the timeout to be 10 seconds. This alows us to solve a lare number of sufficiently difficult instances in reasonable time. However, it is very rare that we solve an instance within 10 seconds but after more than 1 second. Among the total of 160 instances, there is only one instance for which our solvers require around 4 seconds to finish. We can see this in Table 4.10. For this instance, the baseline solver failed. When we try to solve larger instances, at some point, there is a spike in the memory consumption and then, our machine stopped responding. This is most likely due to an encounter with a fixed-point iteration which generates formula that are very large. However, this memory problem is hard to catch and we currently do not have a strong evaluation for this.

Table 4.10: One of the solvable instances with the worst running time.

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc	to
108	10/20/30	CNF Baseline	-	-	-	-	0/0	0	0	-	-	1
		CNF SRD inclusion	5.76s	-	-	-	82/120	38	0	-	-	0
		CNF SRD forward	5.63s	-	8.88ms	-	82/120	38	0	0	-	0
		CNF SRD backward	3.53s	-	-	6.77ms	82/120	38	0	-	0	0
		CNF SRD fb	3.66s	-	5.09ms	4.80ms	82/120	38	0	0	0	0
		CNF SRD inclRed1	4.82s	-	-	-	82/120	38	25	-	-	0
		CNF SRD fRed1	3.97s	-	5.93ms	-	82/120	38	25	0	-	0
		CNF SRD bRed1	4.10s	-	-	4.69ms	82/120	38	25	-	0	0
		CNF SRD fbRed1	4.97s	-	5.47ms	5.12ms	82/120	38	43	0	0	0
		CNF SRD inclRed2	4.12s	-	-	-	82/120	38	25	-	-	0
		CNF SRD fRed2	4.38s	-	5.29ms	-	82/120	38	25	0	-	0
		CNF SRD bRed2	4.34s	-	-	4.23ms	82/120	38	25	-	0	0
		CNF SRD fbRed2	4.64s	-	6.95ms	6.00ms	82/120	38	25	0	0	0
		CNF SRD inclRed4	3.98s	-	-	-	82/120	38	14	-	-	0
		CNF SRD fRed4	4.05s	-	5.81ms	-	82/120	38	14	0	-	0
		CNF SRD bRed4	4.25s	-	-	4.66ms	82/120	38	14	-	0	0
		CNF SRD fbRed4	4.67s	-	4.58ms	4.17ms	82/120	38	14	0	0	0

4.3.4 Average Cases

Despite that the results vary, we can see that the results are actually positive on average. In order to observe this, we average or sum the results for each parameter set. For most parameter sets, we can see improvement. For example, the average time for parameter set 30/20/30 is around 30% faster with the simulation relation based solvers. This is shown in Table 4.12.

Table 4.12: Summary for parameter 30/20/30.

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr
141-160	30/20/30	CNF Baseline	73.76ms	100%	-	-	903/1186	283	0
		CNF SRD inclusion	53.03ms	72%	-	-	842/1087	245	0
		CNF SRD forward	54.19ms	73%	1.91s	-	842/1087	245	0
		CNF SRD backward	53.89ms	73%	-	207.33ms	842/1087	245	0
		CNF SRD fb	54.78ms	74%	1.95s	213.34ms	842/1087	245	0
		CNF SRD inclRed1	52.11ms	71%	-	-	842/1087	245	130
		CNF SRD fRed1	52.73ms	71%	1.93s	-	842/1087	245	131
		CNF SRD bRed1	52.03ms	71%	-	210.80ms	842/1087	245	130
		CNF SRD fbRed1	56.67ms	77%	2.01s	216.66ms	842/1087	245	245
		CNF SRD inclRed2	54.25ms	74%	-	-	842/1087	245	130
		CNF SRD fRed2	53.94ms	73%	1.97s	-	842/1087	245	131
		CNF SRD bRed2	53.78ms	73%	-	208.01ms	842/1087	245	130
		CNF SRD fbRed2	56.63ms	77%	2.04s	215.27ms	842/1087	245	131
		CNF SRD inclRed4	53.23ms	72%	-	-	842/1087	245	60
		CNF SRD fRed4	53.18ms	72%	1.99s	-	842/1087	245	60
		CNF SRD bRed4	51.00ms	69%	-	207.47ms	842/1087	245	60
		CNF SRD fbRed4	54.43ms	74%	1.97s	209.20ms	842/1087	245	60

There is also a parameter set which aggregates are slightly worse that the baseline's aggregates. Three of our solvers even take 32% to 52% longer than the baseline CNF. This is shown in Table 4.14. The other summaries are shown in the Appendix.

Table 4.14: Summary for parameter 10/20/30.

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	to
101-120	10/20/30	CNF Baseline	10.13ms	100%	-	-	905/1176	271	0	1
		CNF SRD inclusion	9.38ms	93%	-	-	853/1096	243	0	0
		CNF SRD forward	11.59ms	114%	52.24ms	-	853/1096	243	0	0
		CNF SRD backward	15.35ms	152%	-	16.28ms	853/1096	243	0	0
		CNF SRD fb	13.39ms	132%	70.07ms	16.50ms	853/1096	243	0	0
		CNF SRD inclRed1	10.64ms	105%	-	-	853/1096	243	121	0
		CNF SRD fRed1	13.76ms	136%	54.90ms	-	853/1096	243	121	0
		CNF SRD bRed1	11.32ms	112%	-	15.93ms	853/1096	243	121	0
		CNF SRD fbRed1	12.13ms	120%	58.39ms	16.25ms	853/1096	243	250	0
		CNF SRD inclRed2	10.67ms	105%	-	-	853/1096	243	121	0
		CNF SRD fRed2	14.36ms	142%	55.00ms	-	853/1096	243	121	0
		CNF SRD bRed2	10.91ms	108%	-	15.18ms	853/1096	243	121	0
		CNF SRD fbRed2	11.34ms	112%	57.97ms	15.58ms	853/1096	243	121	0
		CNF SRD inclRed4	11.44ms	113%	-	-	853/1096	243	59	0
		CNF SRD fRed4	10.28ms	102%	49.06ms	-	853/1096	243	59	0
		CNF SRD bRed4	11.23ms	111%	-	14.03ms	853/1096	243	59	0
		CNF SRD fbRed4	10.64ms	105%	46.82ms	12.89ms	853/1096	243	59	0

We also try to see whether there is a correlation between one of the parameters and the results. From our experiments, in general, we don't see that an increase of a parameter will affect the results in a certain way. For example, from Table B.1 in the Appendix, we can see that the runtimes for parameter set 10/20/10 are between 40%-74%. When we increase the number of nonterminals to 30, the runtimes are between 93%-142%. This indicates that increasing the number of nonterminals will decrease the performance of the simulation relation based solvers. However, for parameter set 30/20/10 and 30/20/30, the simulation relation based solvers show better performance for the parameter set with 30 nonterminals. For parameter set 30/20/10, the runtimes are between 99%-110%. In contrast, for parameter set 30/20/30, the runtimes are between 69%-74%. This happens not only for the nonterminals, but also for the other parameters.

In terms of the implication checks, we can see that subset relation contributes most of the reductions. For example, the Table 4.16, shows that the subset relation based solver reduces the total number implication checks from 399 to 372. This is 27 implication checks less than the result of the baseline solver. In contrast, forward simulation relation based solver reduces this to 370 implication checks. Subset relation is a subposet of forward simulation relation. This means that the forward simulation relation based solver only cuts off two extra implication checks. We can see that the runtime is even slightly worse than the subset relation based solver. In other words, the extra costs outweigh the extra benefits. This does not occur all the time but in most cases, the weaker the simulation relations, the less improvements we get. In fact, the subset relation based solver reduces implication checks for some instances with all of the parameter sets. In contrast, the other simulation relation based solvers only show this advantage for half of the parameter sets. However, this does not necessarily mean that subset relation is better. We may still want to get improvements even if it is small.

Table 4.16: Summary for parameter 10/2/10.

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr
		CNF Baseline	17.54ms	100%	-	-	244/ 399	155	0
		CNF SRD inclusion	9.17ms	52%	-	-	236/ 372	136	0
		CNF SRD forward	11.21ms	64%	38.60ms	-	236/ 370	134	0
		CNF SRD backward	10.06ms	57%	-	12.26ms	236/ 372	136	0
		CNF SRD fb	12.95ms	74%	41.45ms	13.18ms	236/ 370	134	0
		CNF SRD inclRed1	10.68ms	61%	-	-	236/ 372	136	73
		CNF SRD fRed1	10.24ms	58%	41.62ms	-	236/ 370	134	75
		CNF SRD bRed1	9.18ms	52%	-	12.77ms	236/ 372	136	73
61-80	10/20/10	CNF SRD fbRed1	11.49ms	66%	47.66ms	17.88ms	236/ 370	134	133
		CNF SRD inclRed2	9.53ms	54%	-	-	236/ 372	136	73
		CNF SRD fRed2	9.38ms	53%	41.47ms	-	236/ 370	134	75
		CNF SRD bRed2	8.86ms	51%	-	11.71ms	236/ 372	136	73
		CNF SRD fbRed2	11.31ms	64%	35.92ms	10.58ms	236/ 370	134	75
		CNF SRD inclRed4	8.39ms	48%	-	-	236/ 372	136	37
		CNF SRD fRed4	8.38ms	48%	34.24ms	-	236/ 370	134	38
		CNF SRD bRed4	10.02ms	57%	-	12.23ms	236/ 372	136	37
		CNF SRD fbRed4	9.70ms	55%	39.91ms	12.05ms	236/ 370	134	38

In summary, our solvers show better performances for most cases. This is principally caused by the weakening of the implication relation. This reduction enables us to improve many aspects. Firstly, the number of implication checks can be reduced. Secondly, we can decrease the formula representations and reduce memory consumption. Lastly, due to formula reductions, formula compositions are also affected.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we have adapted the simulation relations used in [1] to improve the runtime of the fixed-point iteration in our context-free games. It was previously used for Büchi Automata and we adjust it for NFA. We also argued why this works by showing the monotonicity property. We showed that we can minimize the vectors of formulas within the fixed-point iteration algorithm. This works by removing some boxes and clauses while maintaining logical equivalence.

In Chapter 4, we tried implementing the simulation relations in c++ by adding additional subroutines to an existing context-free games program from [9]. We set up some experiments to see how well this works. Finally, we concluded that our set of optimization techniques work well for most of the instances. We also argued that the success of our optimizations depend on the generated instances. We used an instance generation algorithm based on Tabakov-Vardi model [10]. This algorithm may produce instances where the NFAs do not have a sufficient amount of relations with which we can speed up the fixed-point iteration. We may also find that the formulas within a fixed-point iteration generated for some CFGs do not have a sufficient number of related boxes. This could mean that the overhead will be bigger than the speed up.

5.2 Future work

Our work about simulation relations on context-free games leads us to consider other ways for further improving the context-free games. Firstly, we can look into algorithms for the precomputation. Our current implementation is done in a naive way. We start with all states set as related, then iteratively eliminate incorrect relations. There are other algorithms that are faster. Using a faster algorithm for precomputations may not improve the runtime of the fixed-point iteration, but we could have better times for the precomputations.

Secondly, we can also perform the implication checks by using a SAT solver. The idea is to form a new formula from two formulas F and G to be checked for implication by adding information about the implication between boxes by using logical operator. We will check the resulting formula using the SAT solver.

Thirdly, we stated in the beginning that formula reductions have some important advantages. Firstly, a successful reduction in the beginning of the iteration will affect the rest of the iteration. This is due to the monotonicity property. For example, if we have a formula $\rho_a \vee \rho_b$ such that $\rho_a \sqsubseteq_f \rho_b$ and we compose it with another formula consisting one box τ, then we have a resulting formula $\rho_a; \tau \vee \rho_b; \tau$ with $\rho_a; \tau \sqsubseteq_f \rho_b; \tau$. If we reduce it to ρ_a based on Lemma 3.3.8, the effect of the reduction is also carried to the formula after the composition. We will get $\rho_a; \tau$. Secondly, the more significant the decrease of the formula size, the higher the performance. These two advantages have not been properly evaluated in this thesis. So, we propose evaluations of the size decrease and the positions of the reductions in the iteration.

One possible analysis of size decrease evaluations is by taking the ratio of the formula size after and before reduction. Then, we average these ratios over all of the reduction attempts. This will in some way reflect how much of the formulas are reduced.

For the positions of the reductions, one possible analysis is to take the average of the iteration step indices where reductions occur. If reductions occurs at step 2,3,5, and 10, then the average is 5. This may be used to reflect the center of the reductions.

Lastly, based on the experiments, we still have a lot of instances which results are not any better by using the simulation relations. One of the reasons is that the simulation relations depend on the structure of the instances that are generated. We can have an NFA with only a small amount of non-reflexive forward and backward relations. In this case, the extra cost of computing the implication checks will outweigh the benefits of the optimizations. The context-free grammars may also generate formulas with little or no subsumed parts. This case will also affect the optimizations in negative way. Therefore, it may be beneficial to also explore whether for real cases, the instances are optimizable. Another aspect that we could consider is the instance generation algorithm. Based on Section 4.1.1, the generation algorithm based on Tabakov-Vardi model does not specifically take into account the simulation relations between states. So, the occurrences of the simulation relations depend on the random generator.

Bibliography

[1] P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Advanced ramsey-based büchi automata inclusion testing. In *Proceedings of the 22Nd International Conference on Concurrency Theory*, CONCUR'11, pages 187–202, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] S. Gulwani. Automating repetitive tasks for the masses. *SIGPLAN Not.*, 50(1):1–2, Jan. 2015.

[3] S. Gulwani, V. A. Korthikanti, and A. Tiwari. Synthesizing geometry constructions. Microsoft Research, June 2011.

[4] L. Holík and R. Meyer. Antichains for the verification of recursive programs. In *Networked Systems - Third International Conference, NETYS 2015, Agadir, Morocco, May 13-15, 2015, Revised Selected Papers*, pages 322–336, 2015.

[5] L. Holík, R. Meyer, and S. Muskalla. Antichains for inclusion games. *CoRR*, abs/1603.07256, 2016.

[6] D. Kozen. *Automata and Computability.* Undergraduate Texts in Computer Science. Springer New York, 2007.

[7] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 49–61, New York, NY, USA, 1995. ACM.

[8] V. Stoltenberg-Hansen, I. Lindström, and E. R. Griffor. *Mathematical Theory of Domains*. Cambridge University Press, New York, NY, USA, 1994.

[9] F. Stutz. Operations on a Symbolic Domain for Synthesis. Master's thesis, TU Kaiserslautern, Kaiserslautern, 2017.

[10] D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In *In LPAR 2005, LNCS 3835*, pages 396–411. Springer, 2005.

Appendix A

Table of Instances

The following table contains the results for two out of 20 instances for every parameter set generated during the experiments. We pick them based on whether we consider that the results can give insights to the readers. The insights include indications of improvements, negative results, or timeouts.

Table A.1: Experiment table.

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc	to
		CNF Baseline	408.42us	100%	-	-	11/12	1	0	-	-	0
		CNF SRD inclusion	407.55us	100%	-	-	11/12	1	0	-	-	0
		CNF SRD forward	409.60us	100%	841.69us	-	11/12	1	0	22	-	0
		CNF SRD backward	448.97us	110%	-	627.52us	11/12	1	0	-	19	0
		CNF SRD fb	415.26us	102%	845.80us	619.09us	11/12	1	0	22	19	0
		CNF SRD inclRed1	651.73us	160%	-	-	11/12	1	2	-	-	0
		CNF SRD fRed1	434.50us	106%	825.11us	-	11/12	1	2	22	-	0
		CNF SRD bRed1	409.73us	100%	-	593.04us	11/12	1	2	-	19	0
1	10/2/10	CNF SRD fbRed1	610.67us	150%	807.38us	596.38us	11/12	1	3	22	19	0
		CNF SRD inclRed2	615.68us	151%	-	-	11/12	1	2	-	-	0
		CNF SRD fRed2	462.87us	113%	866.95us	-	11/12	1	2	22	-	0
		CNF SRD bRed2	438.61us	107%	-	645.17us	11/12	1	2	-	19	0
		CNF SRD fbRed2	652.22us	160%	878.13us	653.74us	11/12	1	2	22	19	0
		CNF SRD inclRed4	501.83us	123%	-	-	11/12	1	1	-	-	0
		CNF SRD fRed4	698.15us	171%	946.34us	-	11/12	1	1	22	-	0
		CNF SRD bRed4	836.11us	205%	-	979.54us	11/12	1	1	-	19	0
		CNF SRD fbRed4	991.55us	243%	1.50ms	1.25ms	11/12	1	1	22	19	0
		CNF Baseline	8.63ms	100%	-	-	22/34	12	0	-	-	0
		CNF SRD inclusion	5.86ms	68%	-	-	19/29	10	0	-	-	0
		CNF SRD forward	2.45ms	28%	8.15ms	-	15/22	7	0	63	-	0
		CNF SRD backward	2.22ms	26%	-	5.56ms	15/22	7	0	-	45	0
		CNF SRD fb	1.92ms	22%	7.36ms	4.70ms	15/22	7	0	63	45	0
		CNF SRD inclRed1	3.56ms	41%	-	-	19/29	10	5	-	-	0
		CNF SRD fRed1	1.80ms	21%	6.51ms	-	15/22	7	1	63	-	0
		CNF SRD bRed1	1.84ms	21%	-	4.03ms	15/22	7	1	-	45	0
2	10/2/10	CNF SRD fbRed1	1.76ms	20%	5.45ms	3.75ms	15/22	7	4	63	45	0
		CNF SRD inclRed2	2.65ms	31%	-	-	19/29	10	5	-	-	0
		CNF SRD fRed2	1.79ms	21%	5.52ms	-	15/22	7	1	63	-	0
		CNF SRD bRed2	1.98ms	23%	-	4.29ms	15/22	7	1	-	45	0
		CNF SRD fbRed2	2.24ms	26%	7.16ms	5.09ms	15/22	7	1	63	45	0

		CNF SRD inclRed4	3.71ms	43%	-	-	19/29	10	3	-	-	0
		CNF SRD fRed4	2.35ms	27%	7.13ms	-	15/22	7	1	63	-	0
		CNF SRD bRed4	2.27ms	26%	-	5.36ms	15/22	7	1	-	45	0
		CNF SRD fbRed4	2.21ms	26%	7.30ms	5.10ms	15/22	7	1	63	45	0

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc	to
		CNF Baseline	2.17ms	100%	-	-	33/34	1	0	-	-	0
		CNF SRD inclusion	2.10ms	97%	-	-	33/34	1	0	-	-	0
		CNF SRD forward	2.98ms	137%	4.17ms	-	33/34	1	0	35	-	0
		CNF SRD backward	2.20ms	101%	-	3.74ms	33/34	1	0	-	5	0
		CNF SRD fb	3.38ms	156%	4.01ms	4.14ms	33/34	1	0	35	5	0
		CNF SRD inclRed1	2.15ms	99%	-	-	33/34	1	0	-	-	0
		CNF SRD fRed1	3.29ms	152%	4.78ms	-	33/34	1	0	35	-	0
		CNF SRD bRed1	2.14ms	99%	-	3.26ms	33/34	1	0	-	5	0
26	10/2/30	CNF SRD fbRed1	2.94ms	136%	4.32ms	4.76ms	33/34	1	1	35	5	0
		CNF SRD inclRed2	2.19ms	101%	-	-	33/34	1	0	-	-	0
		CNF SRD fRed2	2.55ms	118%	4.41ms	-	33/34	1	0	35	-	0
		CNF SRD bRed2	3.58ms	165%	-	3.51ms	33/34	1	0	-	5	0
		CNF SRD fbRed2	2.44ms	113%	4.66ms	4.51ms	33/34	1	0	35	5	0
		CNF SRD inclRed4	2.32ms	107%	-	-	33/34	1	0	-	-	0
		CNF SRD fRed4	2.51ms	116%	4.36ms	-	33/34	1	0	35	-	0
		CNF SRD bRed4	2.06ms	95%	-	3.28ms	33/34	1	0	-	5	0
		CNF SRD fbRed4	2.28ms	105%	4.32ms	3.81ms	33/34	1	0	35	5	0
		CNF Baseline	3.11ms	100%	-	-	31/33	2	0	-	-	0
		CNF SRD inclusion	2.57ms	82%	-	-	31/33	2	0	-	-	0
		CNF SRD forward	2.61ms	84%	6.87ms	-	31/33	2	0	48	-	0
		CNF SRD backward	2.25ms	72%	-	5.30ms	31/33	2	0	-	0	0
		CNF SRD fb	1.88ms	60%	5.93ms	4.77ms	31/33	2	0	48	0	0
		CNF SRD inclRed1	2.00ms	64%	-	-	31/33	2	1	-	-	0
		CNF SRD fRed1	1.97ms	63%	5.55ms	-	31/33	2	1	48	-	0
		CNF SRD bRed1	1.88ms	60%	-	4.45ms	31/33	2	1	-	0	0
30	10/2/30	CNF SRD fbRed1	2.06ms	66%	5.17ms	4.77ms	31/33	2	2	48	0	0
		CNF SRD inclRed2	1.86ms	60%	-	-	31/33	2	1	-	-	0
		CNF SRD fRed2	2.01ms	65%	5.48ms	-	31/33	2	1	48	-	0
		CNF SRD bRed2	2.00ms	64%	-	4.82ms	31/33	2	1	-	0	0
		CNF SRD fbRed2	2.36ms	76%	6.79ms	5.71ms	31/33	2	1	48	0	0
		CNF SRD inclRed4	2.50ms	80%	-	-	31/33	2	1	-	-	0
		CNF SRD fRed4	2.67ms	86%	7.40ms	-	31/33	2	1	48	-	0
		CNF SRD bRed4	3.00ms	96%	-	6.28ms	31/33	2	1	-	0	0
		CNF SRD fbRed4	2.97ms	95%	8.66ms	6.85ms	31/33	2	1	48	0	0

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc	to
		CNF Baseline	20.27ms	100%	-	-	21/26	5	0	-	-	0
		CNF SRD inclusion	8.90ms	44%	-	-	17/21	4	0	-	-	0
		CNF SRD forward	2.93ms	14%	120.02ms	-	13/16	3	0	654	-	0
		CNF SRD backward	7.83ms	39%	-	74.71ms	17/21	4	0	-	0	0
		CNF SRD fb	2.55ms	13%	116.90ms	74.25ms	13/16	3	0	654	0	0
		CNF SRD inclRed1	8.18ms	40%	-	-	17/21	4	3	-	-	0
		CNF SRD fRed1	2.74ms	14%	117.90ms	-	13/16	3	2	654	-	0
		CNF SRD bRed1	9.66ms	48%	-	78.07ms	17/21	4	3	-	0	0
51	30/2/10	CNF SRD fbRed1	3.25ms	16%	117.26ms	80.92ms	13/16	3	3	654	0	0
		CNF SRD inclRed2	7.78ms	38%	-	-	17/21	4	3	-	-	0
		CNF SRD fRed2	2.77ms	14%	112.59ms	-	13/16	3	2	654	-	0
		CNF SRD bRed2	7.50ms	37%	-	63.64ms	17/21	4	3	-	0	0
		CNF SRD fbRed2	3.18ms	16%	122.47ms	80.93ms	13/16	3	2	654	0	0
		CNF SRD inclRed4	8.86ms	44%	-	-	17/21	4	1	-	-	0
		CNF SRD fRed4	3.03ms	15%	120.92ms	-	13/16	3	0	654	-	0
		CNF SRD bRed4	9.08ms	45%	-	86.19ms	17/21	4	1	-	0	0
		CNF SRD fbRed4	2.76ms	14%	109.63ms	72.58ms	13/16	3	0	654	0	0
		CNF Baseline	708.98ms	100%	-	-	25/42	17	0	-	-	0
		CNF SRD inclusion	296.04ms	42%	-	-	20/34	14	0	-	-	0
		CNF SRD forward	51.04ms	7%	82.60ms	-	16/22	6	0	519	-	0
		CNF SRD backward	285.47ms	40%	-	36.59ms	20/34	14	0	-	2	0
		CNF SRD fb	46.17ms	7%	77.22ms	34.03ms	16/22	6	0	519	2	0
		CNF SRD inclRed1	287.74ms	41%	-	-	20/34	14	5	-	-	0
		CNF SRD fRed1	51.83ms	7%	79.94ms	-	16/22	6	4	519	-	0
		CNF SRD bRed1	358.33ms	51%	-	31.50ms	20/34	14	5	-	2	0
52	30/2/10											

		CNF SRD fbRed1	64.62ms	9%	80.24ms	45.25ms	16/22	6	7	519	2	0
		CNF SRD inclRed2	312.12ms	44%	-	-	20/34	14	5	-	-	0
		CNF SRD fRed2	60.66ms	9%	91.70ms	-	16/22	6	4	519	-	0
		CNF SRD bRed2	324.05ms	46%	-	39.24ms	20/34	14	5	-	2	0
		CNF SRD fbRed2	58.61ms	8%	93.40ms	40.21ms	16/22	6	4	519	2	0
		CNF SRD inclRed4	320.24ms	45%	-	-	20/34	14	3	-	-	0
		CNF SRD fRed4	59.75ms	8%	94.84ms	-	16/22	6	2	519	-	0
		CNF SRD bRed4	326.04ms	46%	-	40.32ms	20/34	14	3	-	2	0
		CNF SRD fbRed4	57.56ms	8%	95.37ms	40.79ms	16/22	6	2	519	2	0
No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc	to
		CNF Baseline	9.34ms	100%	-	-	11/23	12	0	-	-	0
		CNF SRD inclusion	8.93ms	96%	-	-	11/21	10	0	-	-	0
		CNF SRD forward	10.62ms	114%	38.79ms	-	11/21	10	0	0	-	0
		CNF SRD backward	9.80ms	105%	-	13.54ms	11/21	10	0	-	0	0
		CNF SRD fb	10.29ms	110%	38.50ms	11.58ms	11/21	10	0	0	0	0
		CNF SRD inclRed1	9.25ms	99%	-	-	11/21	10	3	-	-	0
		CNF SRD fRed1	28.01ms	300%	94.87ms	-	11/21	10	3	0	-	0
		CNF SRD bRed1	15.57ms	167%	-	25.46ms	11/21	10	3	-	0	0
62	10/20/10	CNF SRD fbRed1	11.43ms	122%	39.98ms	10.54ms	11/21	10	7	0	0	0
		CNF SRD inclRed2	9.61ms	103%	-	-	11/21	10	3	-	-	0
		CNF SRD fRed2	10.62ms	114%	33.08ms	-	11/21	10	3	0	-	0
		CNF SRD bRed2	10.88ms	117%	-	14.01ms	11/21	10	3	-	0	0
		CNF SRD fbRed2	11.36ms	122%	44.49ms	16.66ms	11/21	10	3	0	0	0
		CNF SRD inclRed4	9.78ms	105%	-	-	11/21	10	2	-	-	0
		CNF SRD fRed4	9.34ms	100%	35.56ms	-	11/21	10	2	0	-	0
		CNF SRD bRed4	9.15ms	98%	-	12.73ms	11/21	10	2	-	0	0
		CNF SRD fbRed4	14.21ms	152%	37.00ms	17.55ms	11/21	10	2	0	0	0
		CNF Baseline	-	-	-	-	0/0	0	0	-	-	1
		CNF SRD inclusion	1.25s	-	-	-	15/25	10	0	-	-	0
		CNF SRD forward	1.18s	-	8.67ms	-	15/25	10	0	0	-	0
		CNF SRD backward	1.20s	-	-	6.95ms	15/25	10	0	-	0	0
		CNF SRD fb	1.23s	-	9.74ms	8.79ms	15/25	10	0	0	0	0
		CNF SRD inclRed1	1.21s	-	-	-	15/25	10	3	-	-	0
		CNF SRD fRed1	1.22s	-	7.11ms	-	15/25	10	3	0	-	0
		CNF SRD bRed1	1.28s	-	-	7.51ms	15/25	10	3	-	0	0
63	10/20/10	CNF SRD fbRed1	306.99ms	-	9.59ms	8.17ms	15/25	10	8	0	0	0
		CNF SRD inclRed2	1.24s	-	-	-	15/25	10	3	-	-	0
		CNF SRD fRed2	879.27ms	-	5.98ms	-	15/25	10	3	0	-	0
		CNF SRD bRed2	757.66ms	-	-	5.49ms	15/25	10	3	-	0	0
		CNF SRD fbRed2	802.27ms	-	5.27ms	4.70ms	15/25	10	3	0	0	0
		CNF SRD inclRed4	782.72ms	-	-	-	15/25	10	2	-	-	0
		CNF SRD fRed4	794.67ms	-	5.12ms	-	15/25	10	2	0	-	0
		CNF SRD bRed4	790.88ms	-	-	5.47ms	15/25	10	2	-	0	0
		CNF SRD fbRed4	810.37ms	-	5.26ms	4.94ms	15/25	10	2	0	0	0
No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc	to
		CNF Baseline	30.53ms	100%	-	-	65/88	23	0	-	-	0
		CNF SRD inclusion	34.23ms	112%	-	-	56/74	18	0	-	-	0
		CNF SRD forward	35.64ms	117%	410.61ms	-	56/74	18	0	649	-	0
		CNF SRD backward	35.68ms	117%	-	353.86ms	56/74	18	0	-	589	0
		CNF SRD fb	34.90ms	114%	432.90ms	364.93ms	56/74	18	0	649	589	0
		CNF SRD inclRed1	37.74ms	124%	-	-	56/74	18	7	-	-	0
		CNF SRD fRed1	36.05ms	118%	425.33ms	-	56/74	18	7	649	-	0
		CNF SRD bRed1	37.46ms	123%	-	367.29ms	56/74	18	7	-	589	0
81	30/2/30	CNF SRD fbRed1	36.12ms	118%	447.10ms	378.24ms	56/74	18	15	649	589	0
		CNF SRD inclRed2	37.26ms	122%	-	-	56/74	18	7	-	-	0
		CNF SRD fRed2	34.35ms	112%	450.04ms	-	56/74	18	7	649	-	0
		CNF SRD bRed2	37.49ms	123%	-	380.47ms	56/74	18	7	-	589	0
		CNF SRD fbRed2	35.66ms	117%	445.84ms	375.94ms	56/74	18	7	649	589	0
		CNF SRD inclRed4	36.07ms	118%	-	-	56/74	18	3	-	-	0
		CNF SRD fRed4	43.26ms	142%	451.56ms	-	56/74	18	3	649	-	0
		CNF SRD bRed4	37.64ms	123%	-	359.27ms	56/74	18	3	-	589	0
		CNF SRD fbRed4	32.89ms	108%	443.14ms	378.88ms	56/74	18	3	649	589	0

		CNF Baseline	6.79ms	100%	-	-	39/42	3	0	-	-	0
		CNF SRD inclusion	5.07ms	75%	-	-	39/42	3	0	-	-	0
		CNF SRD forward	9.60ms	141%	249.49ms	-	39/42	3	0	646	-	0
		CNF SRD backward	11.28ms	166%	-	284.34ms	39/42	3	0	-	350	0
		CNF SRD fb	10.42ms	154%	370.54ms	317.64ms	39/42	3	0	646	350	0
		CNF SRD inclRed1	9.80ms	144%	-	-	39/42	3	1	-	-	0
		CNF SRD fRed1	7.04ms	104%	208.66ms	-	39/42	3	1	646	-	0
		CNF SRD bRed1	7.60ms	112%	-	200.56ms	39/42	3	1	-	350	0
82	30/2/30	CNF SRD fbRed1	9.16ms	135%	234.73ms	213.63ms	39/42	3	2	646	350	0
		CNF SRD inclRed2	8.71ms	128%	-	-	39/42	3	1	-	-	0
		CNF SRD fRed2	7.06ms	104%	182.11ms	-	39/42	3	1	646	-	0
		CNF SRD bRed2	6.21ms	92%	-	163.55ms	39/42	3	1	-	350	0
		CNF SRD fbRed2	6.26ms	92%	198.40ms	176.18ms	39/42	3	1	646	350	0
		CNF SRD inclRed4	6.21ms	91%	-	-	39/42	3	0	-	-	0
		CNF SRD fRed4	6.34ms	93%	193.18ms	-	39/42	3	0	646	-	0
		CNF SRD bRed4	7.07ms	104%	-	164.93ms	39/42	3	0	-	350	0
		CNF SRD fbRed4	6.87ms	101%	199.92ms	176.05ms	39/42	3	0	646	350	0

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc	to
		CNF Baseline	4.83ms	100%	-	-	51/72	21	0	-	-	0
		CNF SRD inclusion	4.81ms	100%	-	-	51/72	21	0	-	-	0
		CNF SRD forward	6.16ms	127%	27.25ms	-	51/72	21	0	0	-	0
		CNF SRD backward	7.08ms	147%	-	11.73ms	51/72	21	0	-	0	0
		CNF SRD fb	6.57ms	136%	35.74ms	10.04ms	51/72	21	0	0	0	0
		CNF SRD inclRed1	7.00ms	145%	-	-	51/72	21	4	-	-	0
		CNF SRD fRed1	6.84ms	141%	35.72ms	-	51/72	21	4	0	-	0
		CNF SRD bRed1	6.64ms	137%	-	11.55ms	51/72	21	4	-	0	0
105	10/20/30	CNF SRD fbRed1	13.94ms	288%	40.49ms	13.21ms	51/72	21	12	0	0	0
		CNF SRD inclRed2	8.72ms	180%	-	-	51/72	21	4	-	-	0
		CNF SRD fRed2	6.74ms	139%	43.74ms	-	51/72	21	4	0	-	0
		CNF SRD bRed2	6.79ms	140%	-	11.23ms	51/72	21	4	-	0	0
		CNF SRD fbRed2	8.73ms	181%	40.35ms	12.96ms	51/72	21	4	0	0	0
		CNF SRD inclRed4	7.77ms	161%	-	-	51/72	21	0	-	-	0
		CNF SRD fRed4	6.75ms	140%	38.54ms	-	51/72	21	0	0	-	0
		CNF SRD bRed4	6.94ms	144%	-	11.17ms	51/72	21	0	-	0	0
		CNF SRD fbRed4	7.10ms	147%	37.58ms	11.36ms	51/72	21	0	0	0	0
		CNF Baseline	-	-	-	-	0/0	0	0	-	-	1
		CNF SRD inclusion	5.76s	-	-	-	82/120	38	0	-	-	0
		CNF SRD forward	5.63s	-	8.88ms	-	82/120	38	0	0	-	0
		CNF SRD backward	3.53s	-	-	6.77ms	82/120	38	0	-	0	0
		CNF SRD fb	3.66s	-	5.09ms	4.80ms	82/120	38	0	0	0	0
		CNF SRD inclRed1	4.82s	-	-	-	82/120	38	25	-	-	0
		CNF SRD fRed1	3.97s	-	5.93ms	-	82/120	38	25	0	-	0
		CNF SRD bRed1	4.10s	-	-	4.69ms	82/120	38	25	-	0	0
108	10/20/30	CNF SRD fbRed1	4.97s	-	5.47ms	5.12ms	82/120	38	43	0	0	0
		CNF SRD inclRed2	4.12s	-	-	-	82/120	38	25	-	-	0
		CNF SRD fRed2	4.38s	-	5.29ms	-	82/120	38	25	0	-	0
		CNF SRD bRed2	4.34s	-	-	4.23ms	82/120	38	25	-	0	0
		CNF SRD fbRed2	4.64s	-	6.95ms	6.00ms	82/120	38	25	0	0	0
		CNF SRD inclRed4	3.98s	-	-	-	82/120	38	14	-	-	0
		CNF SRD fRed4	4.05s	-	5.81ms	-	82/120	38	14	0	-	0
		CNF SRD bRed4	4.25s	-	-	4.66ms	82/120	38	14	-	0	0
		CNF SRD fbRed4	4.67s	-	4.58ms	4.17ms	82/120	38	14	0	0	0

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc	to
		CNF Baseline	3.08ms	100%	-	-	13/17	4	0	-	-	0
		CNF SRD inclusion	3.70ms	120%	-	-	13/17	4	0	-	-	0
		CNF SRD forward	5.48ms	178%	5.22s	-	13/17	4	0	670	-	0
		CNF SRD backward	4.55ms	148%	-	424.39ms	13/17	4	0	-	0	0
		CNF SRD fb	4.60ms	149%	5.29s	412.70ms	13/17	4	0	670	0	0
		CNF SRD inclRed1	3.09ms	100%	-	-	13/17	4	8	-	-	0
		CNF SRD fRed1	3.40ms	111%	3.69s	-	13/17	4	8	670	-	0
		CNF SRD bRed1	5.13ms	167%	-	406.70ms	13/17	4	8	-	0	0
126	30/20/10	CNF SRD fbRed1	4.97ms	161%	5.45s	440.73ms	13/17	4	12	670	0	0
		CNF SRD inclRed2	3.65ms	118%	-	-	13/17	4	8	-	-	0
		CNF SRD fRed2	4.86ms	158%	5.19s	-	13/17	4	8	670	-	0
		CNF SRD bRed2	3.99ms	130%	-	420.26ms	13/17	4	8	-	0	0

		CNF SRD fbRed2	5.07ms	165%	5.13s	404.21ms	13/17	4	8	670	0	0
		CNF SRD inclRed4	4.90ms	159%	-	-	13/17	4	5	-	-	0
		CNF SRD fRed4	4.73ms	154%	5.41s	-	13/17	4	5	670	-	0
		CNF SRD bRed4	4.96ms	161%	-	456.60ms	13/17	4	5	-	0	0
		CNF SRD fbRed4	5.12ms	166%	5.21s	439.08ms	13/17	4	5	670	0	0
129	30/20/10	CNF Baseline	31.79ms	100%	-	-	28/36	8	0	-	-	0
		CNF SRD inclusion	10.13ms	32%	-	-	17/21	4	0	-	-	0
		CNF SRD forward	12.89ms	41%	2.19s	-	17/21	4	0	588	-	0
		CNF SRD backward	15.49ms	49%	-	186.96ms	17/21	4	0	-	0	0
		CNF SRD fb	15.65ms	49%	2.29s	183.40ms	17/21	4	0	588	0	0
		CNF SRD inclRed1	15.73ms	49%	-	-	17/21	4	5	-	-	0
		CNF SRD fRed1	14.56ms	46%	2.40s	-	17/21	4	5	588	-	0
		CNF SRD bRed1	14.10ms	44%	-	214.34ms	17/21	4	5	-	0	0
		CNF SRD fbRed1	8.94ms	28%	2.48s	195.71ms	17/21	4	8	588	0	0
		CNF SRD inclRed2	11.46ms	36%	-	-	17/21	4	5	-	-	0
		CNF SRD fRed2	17.73ms	56%	2.29s	-	17/21	4	5	588	-	0
		CNF SRD bRed2	15.70ms	49%	-	171.73ms	17/21	4	5	-	0	0
		CNF SRD fbRed2	14.44ms	45%	2.28s	174.67ms	17/21	4	5	588	0	0
		CNF SRD inclRed4	10.31ms	32%	-	-	17/21	4	3	-	-	0
		CNF SRD fRed4	17.91ms	56%	2.78s	-	17/21	4	3	588	-	0
		CNF SRD bRed4	15.56ms	49%	-	217.99ms	17/21	4	3	-	0	0
		CNF SRD fbRed4	17.77ms	56%	2.85s	211.67ms	17/21	4	3	588	0	0

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	fc	bc	to
143	30/20/30	CNF Baseline	11.41ms	100%	-	-	42/50	8	0	-	-	0
		CNF SRD inclusion	12.83ms	112%	-	-	42/50	8	0	-	-	0
		CNF SRD forward	11.42ms	100%	3.19s	-	42/50	8	0	654	-	0
		CNF SRD backward	12.28ms	108%	-	265.66ms	42/50	8	0	-	0	0
		CNF SRD fb	11.66ms	102%	3.09s	275.66ms	42/50	8	0	654	0	0
		CNF SRD inclRed1	11.65ms	102%	-	-	42/50	8	3	-	-	0
		CNF SRD fRed1	13.75ms	121%	3.27s	-	42/50	8	3	654	-	0
		CNF SRD bRed1	11.12ms	97%	-	285.61ms	42/50	8	3	-	0	0
		CNF SRD fbRed1	11.71ms	103%	3.20s	270.41ms	42/50	8	6	654	0	0
		CNF SRD inclRed2	8.87ms	78%	-	-	42/50	8	3	-	-	0
		CNF SRD fRed2	14.51ms	127%	3.56s	-	42/50	8	3	654	-	0
		CNF SRD bRed2	13.30ms	117%	-	303.33ms	42/50	8	3	-	0	0
		CNF SRD fbRed2	11.15ms	98%	3.46s	269.85ms	42/50	8	3	654	0	0
		CNF SRD inclRed4	9.95ms	87%	-	-	42/50	8	1	-	-	0
		CNF SRD fRed4	10.91ms	96%	2.95s	-	42/50	8	1	654	-	0
		CNF SRD bRed4	10.62ms	93%	-	251.10ms	42/50	8	1	-	0	0
		CNF SRD fbRed4	10.05ms	88%	2.82s	235.57ms	42/50	8	1	654	0	0
145	30/20/30	CNF Baseline	8.13ms	100%	-	-	30/37	7	0	-	-	0
		CNF SRD inclusion	12.27ms	151%	-	-	30/37	7	0	-	-	0
		CNF SRD forward	12.84ms	158%	543.81ms	-	30/37	7	0	0	-	0
		CNF SRD backward	12.88ms	158%	-	194.52ms	30/37	7	0	-	0	0
		CNF SRD fb	13.29ms	163%	592.21ms	218.82ms	30/37	7	0	0	0	0
		CNF SRD inclRed1	12.11ms	149%	-	-	30/37	7	4	-	-	0
		CNF SRD fRed1	12.85ms	158%	592.35ms	-	30/37	7	4	0	-	0
		CNF SRD bRed1	12.10ms	149%	-	210.08ms	30/37	7	4	-	0	0
		CNF SRD fbRed1	12.19ms	150%	574.34ms	196.13ms	30/37	7	5	0	0	0
		CNF SRD inclRed2	11.89ms	146%	-	-	30/37	7	4	-	-	0
		CNF SRD fRed2	12.20ms	150%	554.51ms	-	30/37	7	4	0	-	0
		CNF SRD bRed2	11.17ms	137%	-	192.91ms	30/37	7	4	-	0	0
		CNF SRD fbRed2	13.29ms	163%	553.26ms	192.94ms	30/37	7	4	0	0	0
		CNF SRD inclRed4	12.65ms	155%	-	-	30/37	7	3	-	-	0
		CNF SRD fRed4	11.79ms	145%	553.04ms	-	30/37	7	3	0	-	0
		CNF SRD bRed4	11.05ms	136%	-	186.97ms	30/37	7	3	-	0	0
		CNF SRD fbRed4	12.65ms	155%	536.77ms	187.19ms	30/37	7	3	0	0	0

Appendix B

Table of Aggregates

The following table sums up the results for every parameter combination.

Table B.1: Aggregated experiment results.

No	Q/Σ/N	Solver	time	time%	ftime	btime	ic	fu	fr	to
1-20	10/2/10	CNF Baseline	9.02ms	100%	-	-	377/ 500	123	0	0
		CNF SRD inclusion	7.40ms	82%	-	-	349/ 455	106	0	0
		CNF SRD forward	10.00ms	111%	6.81ms	-	342/ 442	100	0	0
		CNF SRD backward	8.06ms	89%	-	4.91ms	343/ 445	102	0	0
		CNF SRD fb	6.73ms	75%	6.48ms	5.14ms	342/ 442	100	0	0
		CNF SRD inclRed1	4.67ms	52%	-	-	349/ 455	106	55	0
		CNF SRD fRed1	3.98ms	44%	6.47ms	-	342/ 442	100	50	0
		CNF SRD bRed1	3.90ms	43%	-	4.83ms	343/ 445	102	49	0
		CNF SRD fbRed1	4.47ms	50%	6.55ms	5.01ms	342/ 442	100	101	0
		CNF SRD inclRed2	4.67ms	52%	-	-	349/ 455	106	55	0
		CNF SRD fRed2	4.20ms	47%	6.58ms	-	342/ 442	100	50	0
		CNF SRD bRed2	4.52ms	50%	-	5.11ms	343/ 445	102	49	0
		CNF SRD fbRed2	4.52ms	50%	6.73ms	4.92ms	342/ 442	100	50	0
		CNF SRD inclRed4	5.47ms	61%	-	-	349/ 455	106	34	0
		CNF SRD fRed4	4.29ms	48%	7.00ms	-	342/ 442	100	30	0
		CNF SRD bRed4	4.73ms	52%	-	5.77ms	343/ 445	102	31	0
		CNF SRD fbRed4	4.96ms	55%	7.23ms	5.34ms	342/ 442	100	30	0
21-40	10/2/30	CNF Baseline	5.37ms	100%	-	-	860/1007	147	0	0
		CNF SRD inclusion	4.91ms	91%	-	-	838/ 975	137	0	0
		CNF SRD forward	5.28ms	98%	6.60ms	-	836/ 972	136	0	0
		CNF SRD backward	4.86ms	90%	-	4.82ms	836/ 972	136	0	0
		CNF SRD fb	5.05ms	94%	6.67ms	4.72ms	836/ 972	136	0	0
		CNF SRD inclRed1	4.58ms	85%	-	-	838/ 975	137	47	0
		CNF SRD fRed1	4.50ms	84%	6.58ms	-	836/ 972	136	52	0
		CNF SRD bRed1	4.76ms	89%	-	4.56ms	836/ 972	136	51	0
		CNF SRD fbRed1	4.99ms	93%	6.56ms	5.03ms	836/ 972	136	99	0
		CNF SRD inclRed2	4.94ms	92%	-	-	838/ 975	137	47	0
		CNF SRD fRed2	4.91ms	91%	6.75ms	-	836/ 972	136	52	0
		CNF SRD bRed2	4.95ms	92%	-	5.04ms	836/ 972	136	51	0
		CNF SRD fbRed2	5.02ms	94%	7.17ms	5.04ms	836/ 972	136	52	0
		CNF SRD inclRed4	4.73ms	88%	-	-	838/ 975	137	25	0
		CNF SRD fRed4	4.73ms	88%	6.63ms	-	836/ 972	136	26	0
		CNF SRD bRed4	4.78ms	89%	-	4.83ms	836/ 972	136	25	0
		CNF SRD fbRed4	4.65ms	87%	6.83ms	4.80ms	836/ 972	136	26	0

		CNF Baseline	86.74ms	100%	-	-	408/ 572	164	0	0
		CNF SRD inclusion	56.31ms	65%	-	-	377/ 511	134	0	0
		CNF SRD forward	38.87ms	45%	217.54ms	-	363/ 483	120	0	0
		CNF SRD backward	52.79ms	61%	-	159.64ms	371/ 503	132	0	0
		CNF SRD fb	39.56ms	46%	204.62ms	150.15ms	363/ 483	120	0	0
		CNF SRD inclRed1	52.48ms	60%	-	-	377/ 511	134	66	0
		CNF SRD fRed1	39.90ms	46%	205.99ms	-	363/ 483	120	62	0
		CNF SRD bRed1	55.97ms	65%	-	152.17ms	371/ 503	132	63	0
41-60	30/2/10	CNF SRD fbRed1	43.10ms	50%	207.25ms	156.31ms	363/ 483	120	115	0
		CNF SRD inclRed2	54.71ms	63%	-	-	377/ 511	134	66	0
		CNF SRD fRed2	39.83ms	46%	204.72ms	-	363/ 483	120	62	0
		CNF SRD bRed2	54.65ms	63%	-	155.46ms	371/ 503	132	63	0
		CNF SRD fbRed2	41.16ms	47%	206.42ms	153.17ms	363/ 483	120	62	0
		CNF SRD inclRed4	55.15ms	64%	-	-	377/ 511	134	35	0
		CNF SRD fRed4	42.79ms	49%	212.27ms	-	363/ 483	120	33	0
		CNF SRD bRed4	53.52ms	62%	-	151.70ms	371/ 503	132	34	0
		CNF SRD fbRed4	41.50ms	48%	205.60ms	155.59ms	363/ 483	120	33	0
		CNF Baseline	17.54ms	100%	-	-	244/ 399	155	0	2
		CNF SRD inclusion	9.17ms	52%	-	-	236/ 372	136	0	1
		CNF SRD forward	11.21ms	64%	38.60ms	-	236/ 370	134	0	1
		CNF SRD backward	10.06ms	57%	-	12.26ms	236/ 372	136	0	1
		CNF SRD fb	12.95ms	74%	41.45ms	13.18ms	236/ 370	134	0	1
		CNF SRD inclRed1	10.68ms	61%	-	-	236/ 372	136	73	1
		CNF SRD fRed1	10.24ms	58%	41.62ms	-	236/ 370	134	75	1
		CNF SRD bRed1	9.18ms	52%	-	12.77ms	236/ 372	136	73	1
61-80	10/20/10	CNF SRD fbRed1	11.49ms	66%	47.66ms	17.88ms	236/ 370	134	133	1
		CNF SRD inclRed2	9.53ms	54%	-	-	236/ 372	136	73	1
		CNF SRD fRed2	9.38ms	53%	41.47ms	-	236/ 370	134	75	1
		CNF SRD bRed2	8.86ms	51%	-	11.71ms	236/ 372	136	73	1
		CNF SRD fbRed2	11.31ms	64%	35.92ms	10.58ms	236/ 370	134	75	1
		CNF SRD inclRed4	8.39ms	48%	-	-	236/ 372	136	37	1
		CNF SRD fRed4	8.38ms	48%	34.24ms	-	236/ 370	134	38	1
		CNF SRD bRed4	10.02ms	57%	-	12.23ms	236/ 372	136	37	1
		CNF SRD fbRed4	9.70ms	55%	39.91ms	12.05ms	236/ 370	134	38	1
		CNF Baseline	14.73ms	100%	-	-	836/ 968	132	0	0
		CNF SRD inclusion	12.91ms	88%	-	-	818/ 941	123	0	0
		CNF SRD forward	15.13ms	103%	334.27ms	-	818/ 941	123	0	0
		CNF SRD backward	15.43ms	105%	-	266.15ms	818/ 941	123	0	0
		CNF SRD fb	14.84ms	101%	357.10ms	278.56ms	818/ 941	123	0	0
		CNF SRD inclRed1	14.92ms	101%	-	-	818/ 941	123	43	0
		CNF SRD fRed1	14.70ms	100%	328.04ms	-	818/ 941	123	43	0
		CNF SRD bRed1	14.30ms	97%	-	249.39ms	818/ 941	123	43	0
81-100	30/2/30	CNF SRD fbRed1	14.54ms	99%	320.02ms	242.66ms	818/ 941	123	92	0
		CNF SRD inclRed2	14.04ms	95%	-	-	818/ 941	123	43	0
		CNF SRD fRed2	14.11ms	96%	328.09ms	-	818/ 941	123	43	0
		CNF SRD bRed2	13.56ms	92%	-	252.04ms	818/ 941	123	43	0
		CNF SRD fbRed2	14.16ms	96%	319.22ms	250.31ms	818/ 941	123	43	0
		CNF SRD inclRed4	13.40ms	91%	-	-	818/ 941	123	22	0
		CNF SRD fRed4	14.55ms	99%	340.94ms	-	818/ 941	123	22	0
		CNF SRD bRed4	15.07ms	102%	-	263.20ms	818/ 941	123	22	0
		CNF SRD fbRed4	15.98ms	109%	345.21ms	273.72ms	818/ 941	123	22	0
		CNF Baseline	10.13ms	100%	-	-	905/1176	271	0	1
		CNF SRD inclusion	9.38ms	93%	-	-	853/1096	243	0	0
		CNF SRD forward	11.59ms	114%	52.24ms	-	853/1096	243	0	0
		CNF SRD backward	15.35ms	152%	-	16.28ms	853/1096	243	0	0
		CNF SRD fb	13.39ms	132%	70.07ms	16.50ms	853/1096	243	0	0
		CNF SRD inclRed1	10.64ms	105%	-	-	853/1096	243	121	0
		CNF SRD fRed1	13.76ms	136%	54.90ms	-	853/1096	243	121	0
		CNF SRD bRed1	11.32ms	112%	-	15.93ms	853/1096	243	121	0
101-120	10/20/30	CNF SRD fbRed1	12.13ms	120%	58.39ms	16.25ms	853/1096	243	250	0
		CNF SRD inclRed2	10.67ms	105%	-	-	853/1096	243	121	0
		CNF SRD fRed2	14.36ms	142%	55.00ms	-	853/1096	243	121	0
		CNF SRD bRed2	10.91ms	108%	-	15.18ms	853/1096	243	121	0
		CNF SRD fbRed2	11.34ms	112%	57.97ms	15.58ms	853/1096	243	121	0
		CNF SRD inclRed4	11.44ms	113%	-	-	853/1096	243	59	0
		CNF SRD fRed4	10.28ms	102%	49.06ms	-	853/1096	243	59	0

		CNF SRD bRed4	11.23ms	111%	-	14.03ms	853/1096	243	59	0
		CNF SRD fbRed4	10.64ms	105%	46.82ms	12.89ms	853/1096	243	59	0
		CNF Baseline	32.19ms	100%	-	-	299/ 432	133	0	1
		CNF SRD inclusion	33.16ms	103%	-	-	280/ 397	117	0	1
		CNF SRD forward	34.19ms	106%	3.08s	-	277/ 390	113	0	1
		CNF SRD backward	34.65ms	108%	-	274.08ms	280/ 397	117	0	1
		CNF SRD fb	35.36ms	110%	3.08s	288.11ms	277/ 390	113	0	1
		CNF SRD inclRed1	34.94ms	109%	-	-	280/ 397	117	66	1
		CNF SRD fRed1	35.26ms	110%	2.93s	-	277/ 390	113	73	1
		CNF SRD bRed1	33.21ms	103%	-	260.25ms	280/ 397	117	66	1
121-140	30/20/10	CNF SRD fbRed1	31.76ms	99%	2.91s	273.02ms	277/ 390	113	147	1
		CNF SRD inclRed2	34.06ms	106%	-	-	280/ 397	117	66	1
		CNF SRD fRed2	32.32ms	100%	2.79s	-	277/ 390	113	73	1
		CNF SRD bRed2	35.42ms	110%	-	258.96ms	280/ 397	117	66	1
		CNF SRD fbRed2	33.81ms	105%	3.06s	277.99ms	277/ 390	113	73	1
		CNF SRD inclRed4	34.39ms	107%	-	-	280/ 397	117	34	1
		CNF SRD fRed4	33.40ms	104%	2.97s	-	277/ 390	113	38	1
		CNF SRD bRed4	33.27ms	103%	-	263.10ms	280/ 397	117	34	1
		CNF SRD fbRed4	34.40ms	107%	2.84s	259.08ms	277/ 390	113	38	1
		CNF Baseline	73.76ms	100%	-	-	903/1186	283	0	0
		CNF SRD inclusion	53.03ms	72%	-	-	842/1087	245	0	0
		CNF SRD forward	54.19ms	73%	1.91s	-	842/1087	245	0	0
		CNF SRD backward	53.89ms	73%	-	207.33ms	842/1087	245	0	0
		CNF SRD fb	54.78ms	74%	1.95s	213.34ms	842/1087	245	0	0
		CNF SRD inclRed1	52.11ms	71%	-	-	842/1087	245	130	0
		CNF SRD fRed1	52.73ms	71%	1.93s	-	842/1087	245	131	0
		CNF SRD bRed1	52.03ms	71%	-	210.80ms	842/1087	245	130	0
141-160	30/20/30	CNF SRD fbRed1	56.67ms	77%	2.01s	216.66ms	842/1087	245	245	0
		CNF SRD inclRed2	54.25ms	74%	-	-	842/1087	245	130	0
		CNF SRD fRed2	53.94ms	73%	1.97s	-	842/1087	245	131	0
		CNF SRD bRed2	53.78ms	73%	-	208.01ms	842/1087	245	130	0
		CNF SRD fbRed2	56.63ms	77%	2.04s	215.27ms	842/1087	245	131	0
		CNF SRD inclRed4	53.23ms	72%	-	-	842/1087	245	60	0
		CNF SRD fRed4	53.18ms	72%	1.99s	-	842/1087	245	60	0
		CNF SRD bRed4	51.00ms	69%	-	207.47ms	842/1087	245	60	0
		CNF SRD fbRed4	54.43ms	74%	1.97s	209.20ms	842/1087	245	60	0