Extensions and Applications of Antichain Algorithms

Kei Shirakizawa

March 15, 2017

Contents

1 Introduction

This is the contribution to the study of automata-theoretic thorem proving. Automata-theoretic theorem proving checks the satisfiability / validity of a first-order formula under a fixed interpretation. The elements in the universe are encoded into words and every predicate has a corresponding automaton which accepts the support of the formula. A famous result is the Büchi's theorem for WS1S and S1S [18]. WS1S(resp. S1S) has the set of predicates $\{\subseteq, Sing, = \{0\}\}$ and the set of function symbols $\{Succ, 0\}$. Set variables in a WS1S formula are only instantiated to

a finite set, while S1S allows an infinite set. First-order terms are interpreted on $\mathbb{N}$ as a standard manner. Here $Sing(X)$ means X is a singleton set and $X = \{0\}$ means X is the constant $\{0\}$. Satisfiability / validity are decidable for WS1S formula ϕ, i.e., subset of $(\{0,1\}^n)^*$ is recognizable iff it is definable in WS1S. Left to right direction of the statement is proved by constructing the automaton for each atomic predicate of ϕ and for each logical connectives, we conduct language operations. From closure properties of these operations, the resulting automaton recognizes the support of the formula ϕ.

There are several existing tools for automata-theoretic theorem proving. MONA [6] translates formulas in weak monadic second-order logic(WSkS) to finite tree automata. Recently FORT [15] is implemented for First Order Theory of term rewriting. It is based on tree automata and GTT for left-liner right-ground term rewriting systems. They determine not only whether a formula is valid, but also generate counter-examples from the automata if the formula is not valid.

Difficulty of the automata construction comes from the state explosion problem. WS1S requires tower of computation task corresponding to the depth of quantifiers in the input formula. The determinization of automata causes state explosion, which is necessary to translate the complementation.

Antichain Algorithms The commonly used optimization to tackle the state explosion is the on-the-fly state space generation [10]. Antichain algorithm, another technique originally developed in the model checking, combines the on-the-fly determinization and minimization [19]. Abdulla, et al. [1] combined antichains and a simulation technique and further reduced the state space of the universality/inclusion checking. These techniques are expanded to

1. tree automata [3],
2. Büchi automata on ω language (implemented as ALASKA [20]) and
3. visibly pushdown automata [14, 16, 17, 12].

A number of mitigation techniques have been devised; MONA adopts BDD and path compression. MONA has been improved by antichain algorithm [8]. The work extends the antichain algorithm to handle the nested structure of the prenex normal form. Recently, FORT started to introduce antichain algrorithms. However, antichain algorithms are mostly adopted on a prernex normal forms An interesting empirical observation of FORT is that the flattening of a formula into a prenex normal form triggers further state explosion, which motivated our work. This paper investigates a generalized antichain algorithms without flattening. We focus on monadic first-order logic which has neither set variables (as MONA) nor transitive closure (as FORT), as the most simple case study.

As an optimization, we further introduce

- conversion rules of composition terms which preserve the accepted language and
- distributive laws of emptiness checking into a composition terms.

Our major targets in experiments are Presburger formulas We perform experiments on randomly generated 3000 Presburger formulas. Generalized antichain algorithm improves the performance for sufficiently large and complex problems. Due to the overhead of calculating orderings, it does not work for small problems. In the most cases, conversion of the composition term leads to performance improvement. It also implies that normalization not dedicated to the regular language operations, namely prenex normal form, could affect the performance in the automata construction step.

2 Preliminaries

2.1 Finite Automata and closure property

Definition 2.1. Let Σ be a finite alphabet. A finite automaton (FA) $\mathcal{M}$ on finte words over Σ is a tuple $\langle Q, \Sigma, \delta, I, F \rangle$, where Q is a finite set of states, $\delta \subseteq Q \times \Sigma \times Q$ a transition relation, and $I, F \subseteq Q$ is initial states and final states respectively.

We use subscript to refer each component of the automaton $\mathcal{A}$, e.g., the set of states of $\mathcal{A}$ is referred as $Q_{\mathcal{A}}$.

We impose several assumptions on finite automata.

- A standard ϵ-elimination procedure ensures that $\delta_{\mathcal{M}}$ has no $\epsilon - transition$.
- All states have an incoming transition edge, and thus reachable from the initial states. In case there exist such unreachable states, we can delete them without changing its language so that this assumption holds.

$$\forall\ q' \in Q_{\mathcal{M}}.\ \exists\ c \in \Sigma.\ \exists\ q \in Q_{\mathcal{M}}.\ (q, c, q') \in \delta$$

- By adding the garbage state, all states have outgoing transition edges.

$$\forall\ q \in Q_{\mathcal{M}}.\ \forall\ c \in \Sigma.\ \exists\ q' \in Q_{\mathcal{M}}.\ (q, c, q') \in \delta$$

- Among automata $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, .., they share the same alphabet Σ.
- For any 2 automata $\mathcal{A}$ and $\mathcal{B}$, the state sets are mutually disjoint, i.e., $Q_{\mathcal{A}} \cap Q_{\mathcal{B}} = \varnothing$.

Definition 2.2. $\delta_{\mathcal{M}}$is *deterministic* if

$$\forall\ q \in Q_{\mathcal{M}}.\ \forall\ c \in \Sigma.\ |\{q' \in Q_{\mathcal{M}} \mid (q, c, q') \in \delta_{\mathcal{M}}\}| = 1$$

For a non-deterministic automaton $\mathcal{M}$, when reading one alphabet on a state there are multiple states in the transition relation $\delta_{\mathcal{M}}$. A transition function takes an alphabet and a state and maps to set of states in the transition relation. We denote the transition function $\Delta_{\mathcal{M}}$ as follows;

$$\Delta_{\mathcal{M}} : Q_{\mathcal{M}} \times \Sigma \to 2^{Q_{\mathcal{M}}}$$
$$\Delta_{\mathcal{M}}(q, c) := \{q' \mid (q, c, q') \in \delta_{\mathcal{M}}\}$$

$\Delta_{\mathcal{M}}$ is extended to read a word on a set of states, inductively defined on a word length as follows;

$$\hat{\Delta}_{\mathcal{M}} : 2^{Q_{\mathcal{M}}} \times \Sigma^* \to 2^{Q_{\mathcal{M}}}$$
$$\hat{\Delta}_{\mathcal{M}}(s, \epsilon) := s$$
$$\hat{\Delta}_{\mathcal{M}}(s, cx) := \hat{\Delta}_{\mathcal{M}}(\bigcup_{q \in s} \Delta_{\mathcal{M}}(q, c), x)$$

Note that set operator $\bigcup_i$ commutes; $\bigcup_i \hat{\Delta}(A_i, x) = \hat{\Delta}(\bigcup_i A_i, x)$. Especially $\bigcup_{p \in s} \hat{\Delta}(\{p\}, x) = \hat{\Delta}(s, x)$ holds. For $\hat{\Delta}$ we prepare another inductive definition in terms of a word length.

Proposition 2.1. *Let* $s \subseteq Q_{\mathcal{A}}$, $x \in \Sigma^*$, *and* $c \in \Sigma$. $\hat{\Delta}_{\mathcal{A}}(s, xc) = \bigcup_{q \in \hat{\Delta}_{\mathcal{A}}(s,x)} \Delta_{\mathcal{A}}(q, c)$

We say a string x is accepted by $\mathcal{M}$ if $\exists q \in F_{\mathcal{M}}.\ q \in \hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x)$. For $c \in \Sigma$, we denote $q \xrightarrow{c} p$ if $p \in \Delta_{\mathcal{A}}(q, c)$. For a word $x \in \Sigma^*$, we write $q \xrightarrow{x} p$ if $p \in \hat{\Delta}_{\mathcal{A}}(\{q\}, x)$. By fixing the word x, sometimes we regard $q \xrightarrow{x} p$ as a binary relation on $Q_{\mathcal{A}}$. We say the set of words $\mathcal{A}$ accepts the language of $\mathcal{A}$ and denote it as $L(\mathcal{A})$.

Definition 2.3.

$$post_{\mathcal{A}}(c, s) := \{q' \in Q_{\mathcal{A}} \mid \exists\ q \in s.\ (q, c, q') \in \delta_{\mathcal{A}}\}$$
$$Post_{\mathcal{A}}(s) := \bigcup_{c \in \Sigma} \bigcup_{q \in s} \Delta_{\mathcal{A}}(q, c)$$

Note that $post_{\mathcal{A}}(c, s) = \bigcup_{q \in s} \Delta_{\mathcal{A}}(q, c)$ and that $Post_{\mathcal{A}}(s) = \bigcup_{c \in \Sigma} post_{\mathcal{A}}(c, s)$

Definition 2.4. Let $\mathcal{U}$ be a set and a function $f : 2^{\mathcal{U}} \to 2^{\mathcal{U}}$ be a set operator on $\mathcal{U}$. f is monotone if $s \subseteq t \Rightarrow f(s) \subseteq f(t)$ and f is finitary if $f(A)$ consists of finite subsets of A, i.e., $f(A) = \bigcup_{B \subseteq A} f(B)$, where B is finite. Let X be a set variable. $\mu X.\ f(X)$ is the least fixpoint of f, the point where $X = f(X)$ holds.

$\mu X.\ f(X)$ does not necessarily exist for arbitrary f. Given that f is *finitary*, then $\mu X.\ f(X)$ exists and equals to $\varnothing \cup f(\varnothing) \cup f^2(\varnothing) \cup \ldots \cup f^n(\varnothing)$ for some n [9].

Lemma 2.1. *Let $\mathcal{A}$ be an NFA. $Post_{\mathcal{A}}$ is monotone.*

Definition 2.5. Given a partial order $X, \sqsubseteq$, *antichain* is a subset $Y \sqsubseteq X$ containing only incomparable elements, i.e., $\forall s, s' \in Y.\ s \not\sqsubseteq s'$

Definition 2.6. $s \in S$ is *minimal*.w.r.t $\sqsubseteq$ if $\forall s' \in S.\ s' \not\sqsubset s$. Let $min_{\sqsubseteq}(S) := \{s \in S \mid s \text{ is } minimal \text{ w.r.t } \sqsubseteq \text{ in } S\}$

Theorem 2.2. *The class of the regular language is closed under union, intersection and complement operations.*

2.2 Automata-theoretic theorem proving

Automata-theoretic theorem proving checks the satisfiability / validity of a formula under a fixed interpretation. The elements in the universe are encoded into words and every predicate has a corresponding automaton which accepts the support of the formula. A famous result is the Büchi's theorem for WS1S and S1S [18]. WS1S(resp. S1S) has the set of predicates $\{\subseteq, Sing, = \{0\}\}$ and the set of function symbols $\{Succ, 0\}$. Set variables in the WS1S formula are only instantiated to a finite set, while S1S allows an infinite set. First-order terms are interpreted on $\mathbb{N}$ in a standard manner. Here $Sing(X)$ means X is a singleton set and $X = \{0\}$ means X is the constant $\{0\}$. Satisfiability / validity are decidable for WS1S formula ϕ, i.e., subset of $(\{0,1\}^n)^*$ is recognizable iff it is definable in WS1S. Left to right direction of the statement is proved by constructing the automaton for each atomic predicate of ϕ and for each logical connectives, we conduct language operations. From closure properties of these operations, the resulting automaton recognizes the support of the formula ϕ. In this section we explain our research aim and specify the target problem. Deciding Presburger arithmetic only requires regular language operations on finite automata. For simplicity, we focus on Presburger arithmetic. Following the notation in [8], we write $\begin{smallmatrix} x_1 : a \\ x_2 : b \end{smallmatrix}$ to denote the substitution for each variable x_1, x_2 There are several existing tools for automata-theoretic theorem proving. MONA [6] translates formulas in weak monadic second-order logic(WSkS) to finite tree automata. Recently FORT [15] is implemented for First Order Theory of term rewriting. It is based on tree automata and GTT

for left-liner right-ground term rewriting systems. They determine not only whether a formula is valid, but also generate counter-examples from the automata if the formula is not valid.

Difficulty of the automata construction comes from the state explosion problem. WS1S requires tower of computation task corresponding to the depth of quantifiers in the input formula. In theory, its satisfiability / validity checking problem of is nonerementary [5]. The determinization of automata causes state explosion, which is necessary to translate the complementation. We augment the regular operation with *projection*, which corresponds to an existential quantifier $\exists$. Let $\bar{\Sigma}$ denote $\Sigma \times \ldots \times \Sigma$, n -tuple of Σ. While $c \in \Sigma$, $\begin{smallmatrix} c_1 \\ \vdots \\ c_n \end{smallmatrix}$ is the element of $\bar{\Sigma}$ and we denote it as $\bar{c}$. $\pi_i(\bar{a}, c)$ substitutes the i-th element of $\bar{a}$ to c. We denote $\pi_i(\bar{a}, c) := \begin{smallmatrix} \cdot \\ a_{i-1} \\ c \\ a_{i+1} \\ \cdot \end{smallmatrix}$.

Let $\mathcal{A}$ be a FA with the alphabet $\bar{\Sigma}$. Let $\delta'_{\mathcal{A}}$ be $\bigcup_{(q,\tau,q') \in \delta_{\mathcal{A}}} \bigcup_{c \in \Sigma} \{(q, \pi_i(\tau, c), q')\}$. Projection is an automata operation which replaces $\delta_{\mathcal{A}}$ with $\delta'_{\mathcal{A}}$. Even though $\delta_{\mathcal{A}}$ is deterministic, $\delta'_{\mathcal{A}}$ often becomes non-deterministic.

Antichain Algorithms The commonly used optimization to tackle the state explosion is the on-the-fly state space generation [10]. Antichain algorithm, another technique originally developed in the model checking, combines the on-the-fly determinization and minimization [19]. [11] Abdulla, et al. [1] combined antichains and a simulation technique and further reduced the state space of the universality/inclusion checking. These techniques are expanded to

1. tree automata [3],
2. automata on ω language (implemented as ALASKA [20]) and
3. visibly pushdown automata [14, 16, 17, 12].

A number of mitigation techniques have been devised; MONA adopts BDD and path compression. MONA has been improved by antichain algorithm [8]. The work extends the antichain algorithm to handle the nested structure of the prenex normal form. Recently, FORT started to introduce antichain algorithms. However antichain algorithms are mostly adopted on a prernex normal forms. An interesting empirical observation of FORT is that the flattening of a formula into a prenex normal form triggers further state explosion This observation motivated us to directly handle the formulas of the nested structure without flattening. We focus on monadic first-order logic which has neither set variables (as MONA) nor transitive closure (as FORT), as the most simple case study. Instead, we aim to directly handle a nested formula with an antichain algorithm (i.e., without flattening).

Our major example in the experiments are taken from Presburger arithmetic. Presburger arithmetic is a First-Order theory, whose structure consists of constant symbols $\{0, 1\}$, a function symbol $\{+\}$, a predicate symbol$\{=\}$ and the interpretation is fixed on the domain of $\mathbb{N}$ in a usual addition and equality of numbers. For each atomic formula in Presburger arithmetic, we construct an automaton as described in [4]. For instance, an equation $\phi : x_0 + 2x_1 - 3x_2 = 2$ is recognized by the automaton $\mathcal{R}$ (Fig 1), where the set of positive solutions of ϕ are represented in words over $\bar{\Sigma}$.

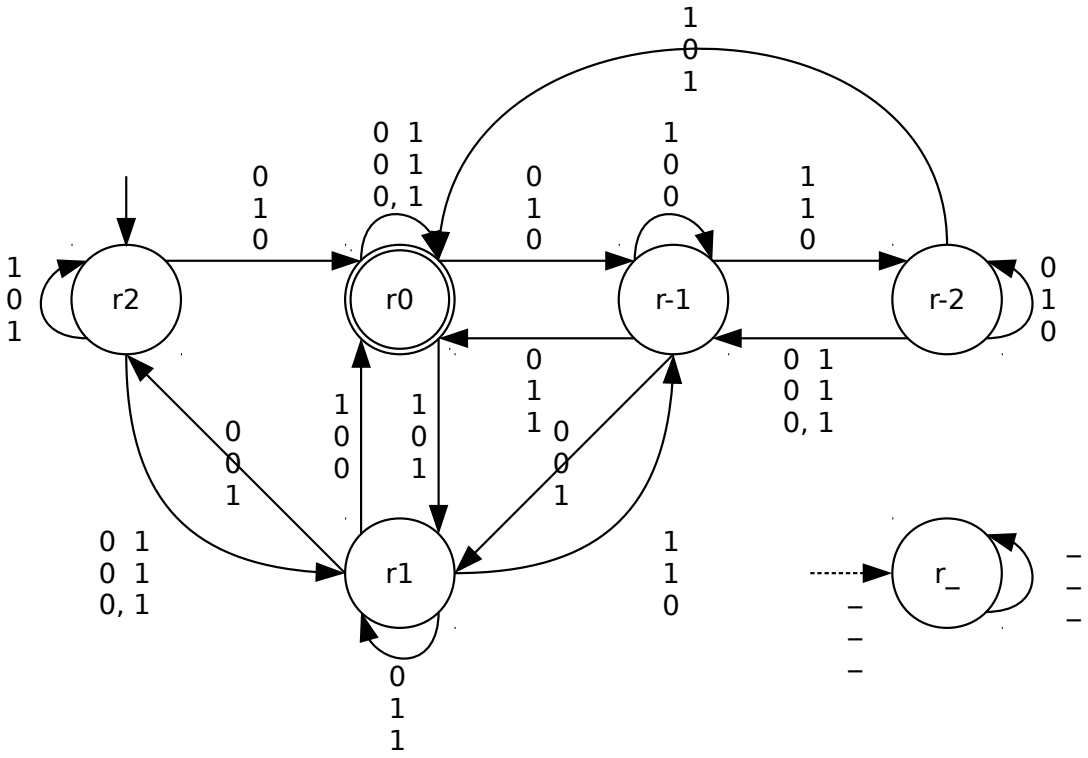

Figure 1: The automaton $\mathcal{R}$ which recognizes the solutions of $x_0 + 2x_1 - 3x_2 = 2$

3 Conventional Antichain Algorithm

In this section, we recall and reformulate the antichain algorithm [13]. We introduce deduction rules of the emptiness checking for later generalization of antichain algorithms.

3.1 Emptiness Checking

Definition 3.1. *Emptiness Checking* is the problem that given an input automaton $\mathcal{A}$, answer whether its language is empty, i.e., $L(\mathcal{A}) = \varnothing$.

The result of *Emptiness Checking* is denoted as $\{\texttt{Empty}, \texttt{NonEmpty}\}$ as boolean values $\{true, false\}$ respectively. We denote the function $succ_{\mathcal{A}} : 2^{Q_{\mathcal{A}}} \to 2^{Q_{\mathcal{A}}}$ for $succ_{\mathcal{A}}(s) := I_{\mathcal{A}} \cup Post_{\mathcal{A}}(s)$.

On-the-fly algorithm of *Emptiness Checking* computes $\mu X.\ (succ_{\mathcal{A}}(X))$ by Kleene ascending chain. If $\mu X.\ (succ_{\mathcal{A}}(X)) \cap F_{\mathcal{A}} = \varnothing$ then return `Empty` else `NonEmpty`. Since $succ_{\mathcal{A}}$ is finitary, the least fixpoint $\mu X.\ (succ_{\mathcal{A}}(X))$ exists.

Lemma 3.1. *Let $x \in \Sigma^*$ and n be the length of x. For all $s \in 2^{Q_{\mathcal{A}}}$, $\hat{\Delta}_{\mathcal{A}}(s, x) \subseteq Post^n_{\mathcal{A}}(s)$.*

Proof. By induction of the length n.

Base case.

$x = \epsilon$, $\hat{\Delta}_{\mathcal{A}}(s, \epsilon) = s \subseteq Post^0_{\mathcal{A}}(s) = s$

Inductive step.

$x = x'c$, let $k+1$ be the length of $x'c$. We have the I.H. $\hat{\Delta}_{\mathcal{A}}(s, x') \subseteq Post^k_{\mathcal{A}}(s)$. By definition, $\hat{\Delta}_{\mathcal{A}}(s, x'c) = \bigcup_{q \in \hat{\Delta}_{\mathcal{A}}(s,x')} \Delta_{\mathcal{A}}(q, c)$. On the other hand,

$$Post^{k+1}_{\mathcal{A}}(s) = Post_{\mathcal{A}}(Post^k_{\mathcal{A}}(s)) = \bigcup_{c \in \Sigma} \bigcup_{q \in Post^k_{\mathcal{A}}(s)} \Delta_{\mathcal{A}}(q, c)$$

$$Post^{k+1}_{\mathcal{A}}(s) \supseteq \bigcup_{q \in Post^k_{\mathcal{A}}(s)} \Delta_{\mathcal{A}}(q, c)$$

By I.H. we have, $\bigcup_{q \in \hat{\Delta}_{\mathcal{A}}(s,x')} \Delta_{\mathcal{A}}(q, c) \subseteq \bigcup_{q \in Post^k_{\mathcal{A}}(s)} \Delta_{\mathcal{A}}(q, c)$. Therefore we have $\hat{\Delta}_{\mathcal{A}}(s, x'c) \subseteq Post^{k+1}_{\mathcal{A}}(s)$. □

Lemma 3.2. $\mu X.\ (succ_{\mathcal{A}}(X)) \cap F_{\mathcal{A}} = \varnothing \Leftrightarrow L(\mathcal{A}) = \varnothing$

Proof. By contradiction. We suppose $\mu X.\ (succ_{\mathcal{A}}(X)) \cap F_{\mathcal{A}} = \varnothing$ and assume $L(\mathcal{A}) \neq \varnothing$. Then we have $x \in L(\mathcal{A})$, i.e., we have an accepting state q such that $q \in \hat{\Delta}_{\mathcal{A}}(I_{\mathcal{A}}, x) \cap F_{\mathcal{A}}$. On the other hand, from Lemma 3.1, we have $\hat{\Delta}_{\mathcal{A}}(I_{\mathcal{A}}, x) \subseteq (Post_{\mathcal{A}})_n(I_{\mathcal{A}})$, where n is the length of x. Since $(Post_{\mathcal{A}})_n(I_{\mathcal{A}}) \subseteq \mu X.\ (succ_{\mathcal{A}}(X))$, we have $q \in \mu X.\ (succ_{\mathcal{A}}(X))$. This is a contradiction. □

3.2 Forward Antichain algorithms for universality/ inclusion

Given an input NFA $\mathcal{A}$, the universality problem is to decide whether $L(\mathcal{A}) = \Sigma^*$. Typically, we first determinize $\mathcal{A}$ and then alternate final states to obtain the complement automaton. If the resulting automaton $\mathcal{A}'$ is empty, then the original is universal. An antichain algorithm performs on-the-fly determinization. Note that a subset of $Q_{\mathcal{A}}$ becomes the new state. The antichain algorithm exploits an ordering over such state sets. Subsets of $Q_{\mathcal{A}}$ are ordered with set inclusion. The idea of the antichain algorithm is to minimize the search space by removing the redundant branches which are bigger w.r.t. '$\subseteq$', and by focusing only on the antichain. [13] *Universality Checking* computes $\mu X.\ (min_{\subseteq}(\{I_{\mathcal{A}}\} \cup \bigcup_{s \in X} \bigcup_{c \in \Sigma} \{post_{\mathcal{A}}(c, s)\}))$. If the least fixpoint contains s such that $s \cap F_{\mathcal{A}} = \varnothing$ then $L(\mathcal{A})$ is not universal.

Given an input NFA $\mathcal{A}$ and $\mathcal{B}$, the inclusion problem is to decide whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$. By closure property of regular languages, typically the problem is reduced to check $L(A) \cap L(\overline{B}) = \varnothing$. In addition to take complement of the automaton, we also conduct product construction. This time a tuple of a state in $Q_{\mathcal{A}}$ and a subset of $Q_{\mathcal{B}}$ becomes the new state. The following ordering over tuples are used to minimize the search space; $=\subseteq \ := \{((q, s), (p, t)) \mid q = p \wedge s \subseteq t\}$. Antichain algorithm for inclusion checking computes $\mu X.\ (min_{=\subseteq}(\bigcup_{q \in I_{\mathcal{A}}} \{(q, I_{\mathcal{B}})\} \cup \bigcup_{(q,s) \in X} \bigcup_{c \in \Sigma} \bigcup_{p \in \Delta_{\mathcal{A}}(q,c)} \{(p, post_{\mathcal{A}}(c, s))\}))$. If the least fixpoint contains (q, s) such that $q \in F_{\mathcal{A}}$ and $s \cap F_{\mathcal{B}} = \varnothing$ then $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$.

3.3 Deduction rules for Emptiness Checking

From comparison of the 3 problems above, we can extract the common part of these algorithm, namely calculation of least fixpoint and emptiness checking. Let us prepare the basic definition in the deduction style. Later in this paper we extend the algorithm to more general and efficient one by adding and replacing rules.

We define the binary relation between X, a set of states of $\mathcal{A}$ and $\{\texttt{Empty}, \texttt{NonEmpty}\}$.

Definition 3.2. Let $\mathcal{M}$ be an NFA. $s \subseteq Q_{\mathcal{M}}$.

$$UNSAT(s) := s \cap F_{\mathcal{M}} = \varnothing$$
$$SAT(s) := s \cap F_{\mathcal{M}} \neq \varnothing$$

Definition 3.3.

$$\frac{}{UNSAT(succ^1_{\mathcal{M}}(\varnothing))}\, I_{\mathcal{M}} \cap F_{\mathcal{M}} = \varnothing \quad \frac{}{SAT(succ^1_{\mathcal{M}}(\varnothing))}\, I_{\mathcal{M}} \cap F_{\mathcal{M}} \neq \varnothing$$

$$\frac{UNSAT(succ^n_{\mathcal{M}}(\varnothing))}{UNSAT(succ^{n+1}_{\mathcal{M}}(\varnothing))}\, Post_{\mathcal{M}}(succ^n_{\mathcal{M}}(\varnothing)) \cap F_{\mathcal{M}} = \varnothing$$

$$\frac{UNSAT(succ^n_{\mathcal{M}}(\varnothing))}{SAT(succ^{n+1}_{\mathcal{M}}(\varnothing))}\, Post_{\mathcal{M}}(succ^n_{\mathcal{M}}(\varnothing)) \cap F_{\mathcal{M}} \neq \varnothing$$

$$\frac{UNSAT(succ^n_{\mathcal{M}}(\varnothing))}{UNSAT(\mu X.\ (succ_{\mathcal{M}}(X)))}\, succ^n_{\mathcal{M}}(\varnothing) = succ^{n+1}_{\mathcal{M}}(\varnothing) \quad \frac{SAT(succ^n_{\mathcal{M}}(\varnothing))}{SAT(\mu X.\ (succ_{\mathcal{M}}(X)))}$$

Note that the i-th iteration of $succ^i_{\mathcal{A}}(\varnothing)$ corresponds to the i-th step reachable states. The proof tree begins from the initial states and explores i-th reachable states of $\mathcal{M}$ for each i until it reaches the least fixpoint, that is $\mu X.\ (succ_{\mathcal{M}}(X))$. It is stressed that the construction of states is incremental; if the non-empty witness is found before fixpoint, then the construction is aborted and the algorithm returns $SAT(\mu X.\ (succ_{\mathcal{M}}(X)))$.

4 Generalized antichain algorithm for composition term

In this section we introduce an inductive definition of the regular operations. Recall that the regular language is closed under complementation, union, intersection. We further add projection. We denote by $\mathcal{A} \otimes \mathcal{B}$ the product automaton of $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A} \oplus \mathcal{B}$ the sum automaton. We decompose the complementation operation into determinization $\mathcal{A}.d$ and alternation of the final states $\mathcal{A}.c$. Projection to the i-th element of an input is denoted $\mathcal{A}.p_i$. For $\mathcal{A}$ or $\mathcal{B}$, if it corresponds to a monadic atomic predicate that has a regular set of supports, we refer an atomic automaton by $\mathcal{A}_0$.

Definition 4.1. Composition term ::=| $\mathcal{A}_0 \mid \mathcal{A}.d \mid \mathcal{A}.c \mid \mathcal{A}.p_i \mid \mathcal{A} \oplus \mathcal{B} \mid \mathcal{A} \otimes \mathcal{B}$

Each symbol in a logical formula is translated into corresponding operator in compostion term. An automaton operation for negation is not direct. When an automaton $\mathcal{A}$ representing φ is non-deterministic, we should apply c to the determinized automaton $\mathcal{A}.d$ to express $\neg\varphi$. We say a composition term is *well formed* if every occurrence of 'c' follows a deterministic automaton. We assume composition term is *well formed*.

Definition 4.2. [2] A set of positions of s is a set of strings over the alphabet $\{1, 2\}$, where;
$\mathcal{P}os(s) \subseteq \{1, 2\}^* :=$

$$\begin{aligned}
\mathcal{P}os(\mathcal{A}) &:= \{\epsilon\} \\
\mathcal{P}os(\mathcal{A}.u) &:= \{\epsilon\} \cup \{1p \mid p \in \mathcal{P}os(\mathcal{A})\} && u \in \{d, c, p_i\} \\
\mathcal{P}os(\mathcal{A}, \mathcal{B}).b &:= \{\epsilon\} \cup \{1p \mid p \in \mathcal{P}os(\mathcal{A})\} \cup \{2p \mid p \in \mathcal{P}os(\mathcal{B})\} && b \in \{\otimes, \oplus\}
\end{aligned}$$

Definition 4.3. A subterm of s at position p , denoted by $s|_p$ is;

$$\begin{aligned}
s|_\epsilon &:= s \\
s_1.u|_{1p'} &:= s_1|_{p'} && u \in \{d, c, p_i\} \\
(s_1, s_2).b|_{ip'} &:= s_i|_{p'} && b \in \{\otimes, \oplus\}
\end{aligned}$$

A subterm relation over composition terms s, t , denoted by $s \leq t$ is;
$s \leq t := \exists p \in \mathcal{P}os(t).\ t|_p = s$

4.1 Interpretation of composition terms

Definition 4.4.

$$\Delta : Q_{\mathcal{M}} \times c \to 2^{Q_{\mathcal{M}}}$$

Case $\mathcal{M} \equiv \mathcal{A}_0$	$\Delta_{\mathcal{A}_0} := \Delta_{\mathcal{A}_0}$
Case $\mathcal{M} \equiv \mathcal{A} \otimes \mathcal{B}$	$\Delta_{\mathcal{A}\otimes\mathcal{B}}((q_l, q_r), c) := \bigcup_{q'_l \in \Delta_{\mathcal{A}}(q_l,c)} \bigcup_{q'_r \in \Delta_{\mathcal{B}}(q_r,c)} \{(q'_l, q'_r)\}$
Case $\mathcal{M} \equiv \mathcal{A} \oplus \mathcal{B}$	$\Delta_{\mathcal{A}\oplus\mathcal{B}} := \Delta_{\mathcal{A}} \cup \Delta_{\mathcal{B}}$
Case $\mathcal{M} \equiv \mathcal{A}.d$	$\Delta_{\mathcal{A}.d}(s, c) := \left\{ \bigcup_{q \in s} \Delta_{\mathcal{A}}(q, c) \right\}$
Case $\mathcal{M} \equiv \mathcal{A}.p_i$	$\Delta_{\mathcal{A}.p_i}(q, \bar{a}) := \bigcup_{c \in \Sigma} \Delta_{\mathcal{A}}(q, \pi_i(\bar{a}, c))$
Case $\mathcal{M} \equiv \mathcal{A}.c$	$\Delta_{\mathcal{A}.c} := \Delta_{\mathcal{A}}$

The transition function $\hat{\Delta}$ for $\mathcal{A}_0$ is also defined for $\mathcal{M}$ without changes.

Definition 4.5.

	$F_{\mathcal{M}} \subseteq Q_{\mathcal{M}}$	$I_{\mathcal{M}} \subseteq Q_{\mathcal{M}}$
Case $\mathcal{M} \equiv \mathcal{A}_0$	$F_{\mathcal{A}_0} := F_{\mathcal{A}_0}$	$I_{\mathcal{A}_0} := I_{\mathcal{A}_0}$
Case $\mathcal{M} \equiv \mathcal{A} \otimes \mathcal{B}$	$F_{A\otimes B} := F_{\mathcal{A}} \times F_{\mathcal{B}}$	$I_{A\otimes B} := I_{\mathcal{A}} \times I_{\mathcal{B}}$
Case $\mathcal{M} \equiv \mathcal{A} \oplus \mathcal{B}$	$F_{A\oplus B} := F_{\mathcal{A}} \cup F_{\mathcal{B}}$	$I_{A\oplus B} := I_{\mathcal{A}} \cup I_{\mathcal{B}}$
Case $\mathcal{M} \equiv \mathcal{A}.d$	$F_{\mathcal{A}.d} := \{s \mid \exists\ q \in s.\ q \in F_{\mathcal{A}}\}$	$I_{\mathcal{A}.d} := \{I_{\mathcal{A}}\}$
Case $\mathcal{M} \equiv \mathcal{A}.p_i$	$F_{\mathcal{A}.p_i} := F_{\mathcal{A}}$	$I_{\mathcal{A}.p_i} := I_{\mathcal{A}}$
Case $\mathcal{M} \equiv \mathcal{A}.c$	$F_{\mathcal{A}.c} := \{q \mid q \notin F_{\mathcal{A}}\}$	$I_{\mathcal{A}.c} := I_{\mathcal{A}}$

4.2 Ordering on states

Definition 4.6.

$$\sqsubseteq_{\mathcal{M}} \subseteq Q_{\mathcal{M}} \times Q_{\mathcal{M}}$$

Case $\mathcal{M} \equiv \mathcal{A}_0$	$\sqsubseteq_{\mathcal{A}_0} := \{(q, q) \mid q \in Q_{\mathcal{A}_0}\}$
Case $\mathcal{M} \equiv \mathcal{A} \otimes \mathcal{B}$	$\sqsubseteq_{\mathcal{A}\otimes\mathcal{B}} := \{((q_a, q_b), (p_a, p_b)) \mid q_a \sqsubseteq_{\mathcal{A}} p_a \wedge q_b \sqsubseteq_{\mathcal{B}} p_b\}$
Case $\mathcal{M} \equiv \mathcal{A} \oplus \mathcal{B}$	$\sqsubseteq_{\mathcal{A}\oplus\mathcal{B}} := \sqsubseteq_{\mathcal{A}} \cup \sqsubseteq_{\mathcal{B}}$
Case $\mathcal{M} \equiv \mathcal{A}.d$	$\sqsubseteq_{\mathcal{A}.d} := \{(U, V) \mid \forall v \in V.\ \exists u \in U.\ u \sqsubseteq_{\mathcal{A}} v\}$
Case $\mathcal{M} \equiv \mathcal{A}.c$	$\sqsubseteq_{\mathcal{A}.c} := \{(p, q) \mid q \sqsubseteq_{\mathcal{A}} p\}$
Case $\mathcal{M} \equiv \mathcal{A}.p_i$	$\sqsubseteq_{\mathcal{A}.p_i} := \sqsubseteq_{\mathcal{A}}$

4.3 Generalized antichain algorithm for composition term

A satisfiability checking algorithm consists of two steps.

1. automata construction described by a composition term t.
2. emptiness checking `EC` described by a deduction rules.

It can be understood as $\texttt{EC}(t)$ when we apply the on-the-fly algorithm, it becomes $(\texttt{EC}(t))_{OTF} = (\texttt{EC})_{OTF}(t_{OTF})$, when $(\texttt{EC}(t))_{OTF}$4.7 and are performed step-by-step on the length of words. A generalized antichain algorithm is described as $(\texttt{EC}(t))_{GAC} = \texttt{EC}_{GAC}(t_{OTF})$ in which the automaton construction is amalgamated with the minimization following to inductively defined orderings. Further optimizations

- conversion of the compositional term to a minimally quantified form(preserving language equivalence)
- distributive laws of the emptiness checking (preserving equisatisfiability)

are presented in Section 6.

Definition 4.7.

$$\frac{}{UNSAT((succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^1(\varnothing))}\, I_{\mathcal{M}} \cap F_{\mathcal{M}} = \varnothing \quad \frac{}{SAT((succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^1(\varnothing))}\, I_{\mathcal{M}} \cap F_{\mathcal{M}} \neq \varnothing$$

$$\frac{UNSAT((succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^n(\varnothing))}{UNSAT((succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^{n+1}(\varnothing))}\, Post_{\mathcal{M}}((succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^n(\varnothing)) \cap F_{\mathcal{M}} = \varnothing$$

$$\frac{UNSAT((succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^n(\varnothing))}{SAT((succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^{n+1}(\varnothing))}\, Post_{\mathcal{M}}((succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^n(\varnothing)) \cap F_{\mathcal{M}} \neq \varnothing$$

$$\frac{UNSAT((succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^n(\varnothing))}{UNSAT(\mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(X)))}\, (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^n(\varnothing) = (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^{n+1}(\varnothing)$$

$$\frac{SAT((succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}})^n(\varnothing))}{SAT(\mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(X)))}$$

Example Let us demonstrate the construction for composition terms through an example. Let ψ be a Presburger formula $\neg(\exists x_0.\ \exists x_2.\ x_0 + 2x_1 - 3x_2 = 2) \wedge \exists x_0.\ 3x_0 + x_1 + 2x_2 = 1$. The automaton $\mathcal{R}$ in the previous chapter accepts the set of solutions for $x_0 + 2x_1 - 3x_2 = 2$. We give another automaton $\mathcal{G}$ for $3x_0 + x_1 + 2x_2 = 1$. A composition term ct which corresponds to ψ is $(\mathcal{R}.p_2.p_0.d.c \otimes \mathcal{G}.p_0)$. The initial state of ct is computed as follows; $I_{ct} = I_{\mathcal{R}.p_2.p_0.d.c \otimes \mathcal{G}.p_0} = \{I_{\mathcal{R}.p_2.p_0.d.c} \times I_{\mathcal{G}.p_0}\} = \ldots = \{\{I_{\mathcal{R}}\} \times I_{\mathcal{G}}\} = \{(\{r_2\}, g_1)\}$. For instance, $\Delta_{ct}\begin{pmatrix}0\\(\{r_2\}, g_1), 1\\0\end{pmatrix} = \{\Delta_{\mathcal{R}.p_2.p_0.d.c}\begin{pmatrix}0\\\{r_2\}, 1\\0\end{pmatrix} \times \Delta_{\mathcal{G}.p_0}\begin{pmatrix}0\\(g_1), 1\\0\end{pmatrix}\} = \ldots = \{(\{r_0, r_1, r\_\}, g_0), (\{r_0, r_1, r\_\}, g_{-1})\}$. $Post_{ct}, succ_{ct}$ are computed based on the Δ_{ct}. We can also find the ordering between the states of ct. According to the definition of $\sqsubseteq_{ct}$, the ordering is subset relation for the left hand side of the tuple and equality for the right hand side. The figure 3 depicts the difference between $ct \rightarrow_{\texttt{EC}} \texttt{Empty}$ and $ct \rightarrow_{\texttt{EC}_{\sqsubseteq}} \texttt{Empty}$. The initial state $(\{r_2\}, g_1)$ is located in the left of the picture. $ct \rightarrow_{\texttt{EC}} \texttt{Empty}$ computes all reachable states within i step. As i proceeds, the new states are added to right direction in the picture. This algorithm generates 31 states and reaches the fixpoint. On the other hand, the generalized antichain algorithm only maintains minimal states w.r.t. $\sqsubseteq_{ct}$. The generated states are circled in the yellow line in the picture. Eventually the antichain algorithm constructs just 15 states and reaches the fixpoint.

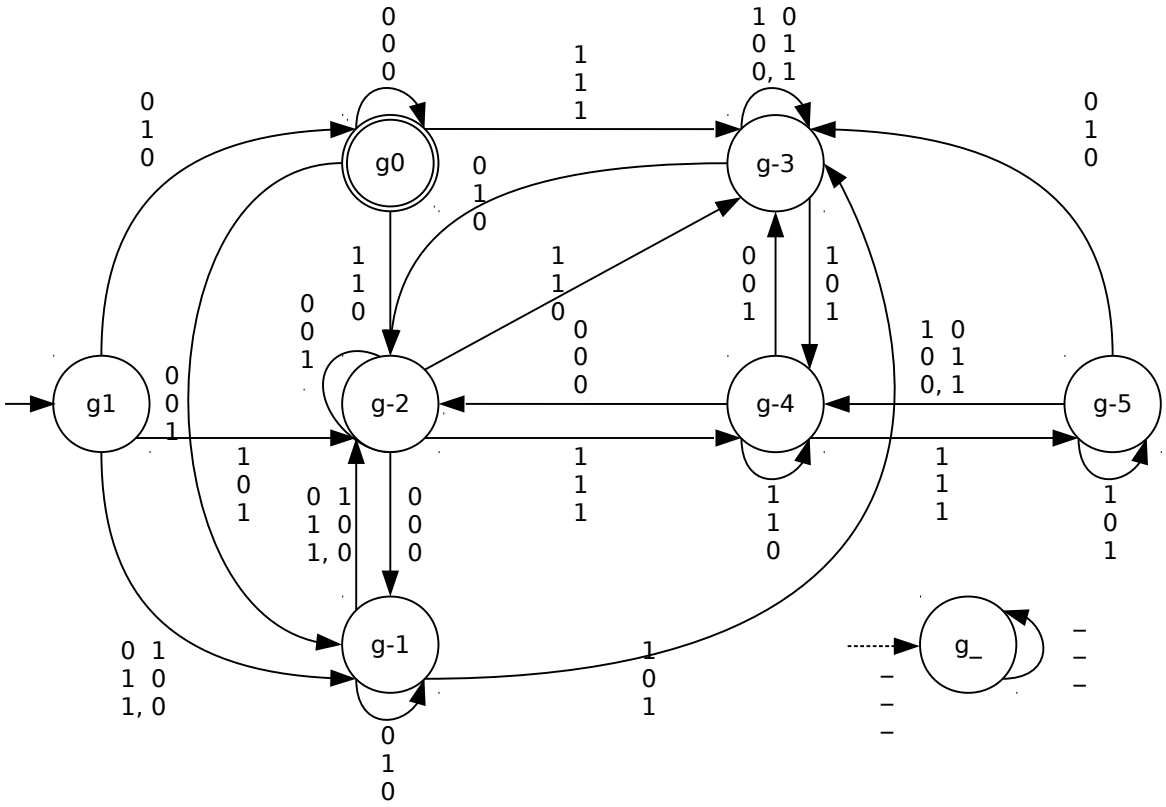

Figure 2: The automaton $\mathcal{G}$ which recognizes the solusions of $3x_0 + x_1 + 2x_2 = 1$

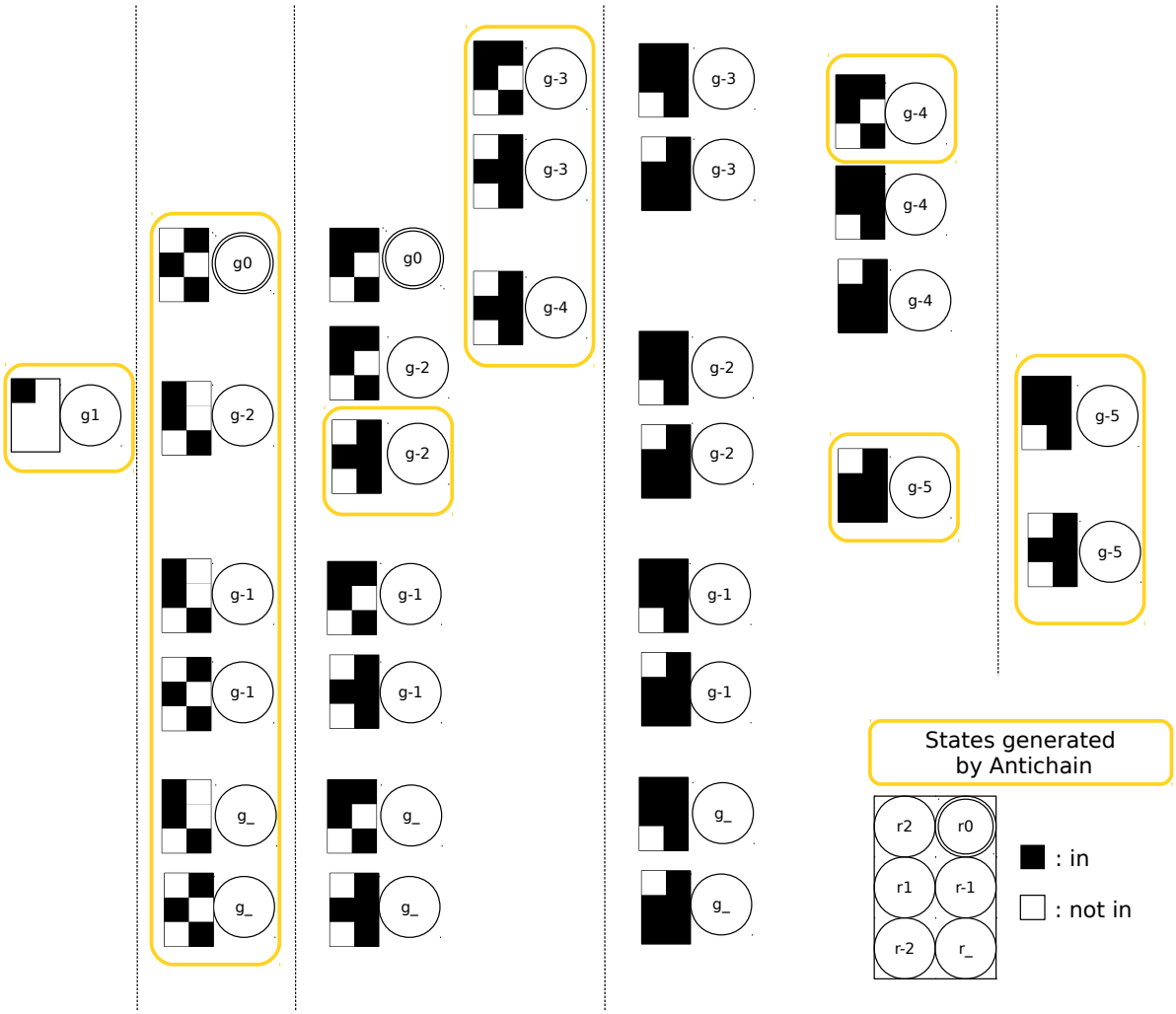

Figure 3: Comparison between the on-the-fly state construction and the antichain algorithm

5 Composition term to solve problems

In this section, we prove that the generalized antichain algorithm is complete and sound. For universality and inclusion, our approach is identical to the conventional antichain algorithms. We observe closely on the universality, the inclusion and a general case, step by step. The key is an inductive construction of orderings following to the structure of composition terms.

5.1 Universality Problem

Lemma 5.1. *Let $\mathcal{A}.d$ be a composition term and $s, t \in Q_{\mathcal{A}.d}$. Let $c \in \Sigma$.*

$$\forall u \in \Delta_{\mathcal{A}.d}(s, c), v \in \Delta_{\mathcal{A}.d}(t, c).\ s \subseteq t \Rightarrow u \subseteq v$$

Proof. Let $m = t \setminus s$. By definition4.4, $u \in \left\{\bigcup_{p \in s} \Delta_{\mathcal{A}}(p, c)\right\}$. We have $u = \bigcup_{p \in s} \Delta_{\mathcal{A}}(p, c)$. Similarly,$v = \bigcup_{p \in t} \Delta_{\mathcal{A}}(p, c)$. Then we have $v = \bigcup_{p \in m} \Delta_{\mathcal{A}}(p, c) \cup \bigcup_{p \in s} \Delta_{\mathcal{A}}(p, c)$. Therefore $u \subseteq v$. □

Lemma 5.2. *Let $x \in \Sigma^*$. $\forall u \in \hat{\Delta}_{\mathcal{A}.d}(\{s\}, x), v \in \hat{\Delta}_{\mathcal{A}.d}(\{t\}, x).\ s \subseteq t \Rightarrow u \subseteq v$*

Proof. By induction on the length of the word x.

Base case.

$x = \epsilon$, $\hat{\Delta}_{\mathcal{A}.d}(\{s\}, \epsilon) = \{s\}$. We have $u = s$. Similarly $v = t$. Thus $u \subseteq v$.

Inductive step.

$x = x'c$, we have the I.H.

$$\forall u \in \hat{\Delta}_{\mathcal{A}.d}(\{s\}, x'), v \in \hat{\Delta}_{\mathcal{A}.d}(\{t\}, x').\ s \subseteq t \Rightarrow u \subseteq v$$

$$\begin{aligned} \text{We also have} \hat{\Delta}_{\mathcal{A}.d}(\{s\}, cx') &= \bigcup_{s' \in \hat{\Delta}_{\mathcal{A}.d}(\{s\}, x')} \Delta_{\mathcal{A}.d}(s', c) \\ \text{Similarly we have } \hat{\Delta}_{\mathcal{A}.d}(\{t\}, cx') &= \bigcup_{t' \in \hat{\Delta}_{\mathcal{A}.d}(\{s\}, x')} \Delta_{\mathcal{A}.d}(t', c) \end{aligned}$$

By I.H., $\forall s' \in \hat{\Delta}_{\mathcal{A}.d}(\{s\}, x'), t' \in \hat{\Delta}_{\mathcal{A}.d}(\{t\}, x').\ s \subseteq t \Rightarrow s' \subseteq' t$, which implies, by Lemma 5.1, $\forall u \in \Delta_{\mathcal{A}.d}(s', c)\ \forall v \in \Delta_{\mathcal{A}.d}(t', c).\ u \subseteq v$. Finally we have

$$\forall u \in \bigcup_{s' \in \hat{\Delta}_{\mathcal{A}.d}(\{s\}, x')} \Delta_{\mathcal{A}.d}(s', c)\ \forall v \in \bigcup_{t' \in \hat{\Delta}_{\mathcal{A}.d}(\{s\}, x')} \Delta_{\mathcal{A}.d}(t', c).\ u \subseteq v$$

□

Lemma 5.3. $\forall u, v \in Q_{\mathcal{A}.d.c}.\ u \subseteq v \Rightarrow (u \in F_{\mathcal{A}.d.c} \Leftarrow v \in F_{\mathcal{A}.d.c})$

Proof. Suppose $v \in F_{\mathcal{A}.d.c}$. By definition4.1 of composition terms, it is equivalent to $\neg(v \in F_{\mathcal{A}.d})$, and to $\neg(\exists q \in v.\ q \in F_{\mathcal{A}})$. Equivalently, $\forall q \in v.\ q \notin F_{\mathcal{A}}$. Since $u \subseteq v$, $\forall q \in u.\ q \notin F_{\mathcal{A}}$. We conclude $u \in F_{\mathcal{A}.d.c}$. □

Theorem 5.4. $\forall u \in \hat{\Delta}_{\mathcal{A}.d.c}(\{s\}, x), v \in \hat{\Delta}_{\mathcal{A}.d.c}(\{t\}, x).\ s \subseteq t \Rightarrow (u \in F_{\mathcal{A}.d.c} \Leftarrow v \in F_{\mathcal{A}.d.c})$

Proof. Note that $\hat{\Delta}_{\mathcal{A}.d} = \hat{\Delta}_{\mathcal{A}.d.c}$. From Lemma 5.2, $s \subseteq t \Rightarrow u \subseteq v$. From Lemma 5.3, $u \in F_{\mathcal{A}.d.c} \Leftarrow v \in F_{\mathcal{A}.d.c}$ □

5.2 Inclusion Problem

Lemma 5.5. *Let $\mathcal{M} = \mathcal{A} \otimes \mathcal{B}.d.c$, a composition term and $(s_l, s_r), (t_l, t_r) \in Q_{\mathcal{M}}$. Let $c \in \Sigma$.*

$$\forall (v_l, v_r) \in \Delta_{\mathcal{M}}((t_l, t_r), c).\ \exists (u_l, u_r) \in \Delta_{\mathcal{M}}((s_l, s_r), c).\ s_l = t_l \wedge s_r \subseteq t_r \Rightarrow (u_l = v_l \wedge u_r \subseteq v_r)$$

Proof. Let $(v_l, v_r) \in \bigcup_{q_l \in \Delta_{\mathcal{A}}(t_l, c)} \bigcup_{q_r \in \Delta_{\mathcal{B}.d.c}(t_r, c)} \{(q_l, q_r)\}$. Let $u_r \in \Delta_{\mathcal{B}.d.c}(s_r, c)$. Since $s_r \subseteq t_r$, from Lemma 5.1, $u_r \subseteq v_r$. We have $v_l = v_l \wedge u_r \subseteq v_r$. Since $s_l = t_l$, $v_l \in \Delta_{\mathcal{A}}(s_l, c)$, we conclude that $(v_l, u_r) \in \Delta_{\mathcal{M}}((s_l, s_r), c)$. □

Lemma 5.6. *Let $\mathcal{M} = \mathcal{A} \otimes \mathcal{B}.d.c$, and $(s_l, s_r), (t_l, t_r) \in Q_{\mathcal{M}}$. Let $x \in \Sigma^*$.*

$$\forall (v_l, v_r) \in \hat{\Delta}_{\mathcal{M}}(\{(t_l, t_r)\}, x).\ \exists (u_l, u_r) \in \hat{\Delta}_{\mathcal{M}}(\{(s_l, s_r)\}, x).\ s_l = t_l \wedge s_r \subseteq t_r \Rightarrow (u_l = v_l \wedge u_r \subseteq v_r)$$

Proof. By induction on the length of x.

Base case.

$x = \epsilon$, $\hat{\Delta}_{\mathcal{M}}(\{(s_l, s_r)\}, \epsilon) = \{(s_l, s_r)\}$ and $\hat{\Delta}_{\mathcal{M}}(\{(t_l, t_r)\}, \epsilon) = \{(t_l, t_r)\}$. (s_l, s_r) and (t_l, t_r) satisfy the condition.

Inductive step.

$x = x'c$, We have I.H.

$$\forall (v_l, v_r) \in \hat{\Delta}_{\mathcal{M}}(\{(t_l, t_r)\}, x').\ \exists (u_l, u_r) \in \hat{\Delta}_{\mathcal{M}}(\{(s_l, s_r)\}, x').\ s_l = t_l \wedge s_r \subseteq t_r \Rightarrow (u_l = v_l \wedge u_r \subseteq v_r)$$

Let $(v_l, v_r) \in \hat{\Delta}_{\mathcal{M}}(\{(t_l, t_r)\}, cx') = \bigcup_{q \in \hat{\Delta}_{\mathcal{M}}(\{(t_l, t_r)\}, x')} \Delta_{\mathcal{M}}(q, c)$. We suppose $(v_l, v_r) \in \Delta_{\mathcal{M}}((t'_l, t'_r), c)$, for arbitrary $(t'_l, t'_r) \in \hat{\Delta}_{\mathcal{M}}(\{(t_l, t_r)\}, x')$. By I.H., we have $(s'_l, s'_r) \in \hat{\Delta}_{\mathcal{M}}(\{(s_l, s_r)\}, x')$ such that $s'_l = t'_l \wedge s'_r \subseteq t'_r$. By Lemma 5.5, there exists $(u_l, u_r) \in \Delta_{\mathcal{M}}((s'_l, s'_r), c)$ and $(u_l = v_l \wedge u_r \subseteq v_r)$ holds. Since $(u_l, u_r) \in \hat{\Delta}_{\mathcal{M}}(\{(s_l, s_r)\}, cx')$, the statement holds for the inductive case. □

Lemma 5.7.

$$\forall (u_l, u_r), (v_l, v_r) \in Q_{\mathcal{A} \otimes \mathcal{B}.d.c}.\ u_l = v_l \wedge u_r \subseteq v_r \Rightarrow ((u_l, u_r) \in F_{\mathcal{A} \otimes \mathcal{B}.d.c} \Leftarrow (v_r, v_l) \in F_{\mathcal{A} \otimes \mathcal{B}.d.c})$$

Proof. Suppose $(v_l, v_r) \in F_{\mathcal{A} \otimes \mathcal{B}.d.c}$. By definition of composition terms, it is equivalent to $v_l \in F_{\mathcal{A}} \wedge v_r \in F_{\mathcal{B}.d.c}$. Since $u_l = v_l \wedge u_r \subseteq v_r$, by using Lemma 5.3, $u_l \in F_{\mathcal{A}} \wedge u_r \in F_{\mathcal{B}.d.c}$. We conclude $(u_l, u_r) \in F_{\mathcal{A}.d.c}$. □

Theorem 5.8. $\forall (v_l, v_r) \in \hat{\Delta}_{\mathcal{M}}(\{(t_l, t_r)\}, x).\ \exists (u_l, u_r) \in \hat{\Delta}_{\mathcal{M}}(\{(s_l, s_r)\}, x).\ s_l = t_l \wedge s_r \subseteq t_r \Rightarrow ((u_l, u_r) \in F_{\mathcal{A} \otimes \mathcal{B}.d.c} \Leftarrow (v_r, v_l) \in F_{\mathcal{A} \otimes \mathcal{B}.d.c})$

Proof. Suppose $(v_l, v_r) \in \hat{\Delta}_{\mathcal{M}}(\{(t_l, t_r)\}, x)$ and $(v_l, v_r) \in F_{\mathcal{A} \otimes \mathcal{B}.d.c}$. Since we have $s_l = t_l \wedge s_r \subseteq t_r$ and from Lemma 5.6, there exists $(u_l, u_r) \in \hat{\Delta}_{\mathcal{M}}(\{(s_l, s_r)\}, x)$ such that $u_l = v_l \wedge u_r \subseteq v_r$. From Lemma 5.7, we have $(u_l, u_r) \in F_{\mathcal{A} \otimes \mathcal{B}.d.c}$. □

5.3 Generalized antichain algorithm for emptiness checking

Lemma 5.9. $\forall q, p \in Q_{\mathcal{M}}. q \sqsubseteq_{\mathcal{M}} p \Rightarrow (q \in F_{\mathcal{M}} \Leftarrow p \in F_{\mathcal{M}})$

Proof.

Base case. $\mathcal{M} \equiv \mathcal{A}_0$

We have $q = p$ by $q \sqsubseteq_{\mathcal{A}_0} p$. Assume $p \in F_{\mathcal{A}_0}$ and we have $q \in F_{\mathcal{A}_0}$.

Case. $\mathcal{M} \equiv \mathcal{A} \otimes \mathcal{B}$

Suppose $(q_a, q_b), (p_a, p_b) \in Q_{\mathcal{A}\otimes\mathcal{B}}$ with $(q_a, q_b) \sqsubseteq_{\mathcal{A}\otimes\mathcal{B}} (p_a, p_b)$. The I.H. is as follows:

- $\forall q_a, p_a \in Q_{\mathcal{A}}.q_a \sqsubseteq_{\mathcal{A}} p_a \Rightarrow (q_a \in F_{\mathcal{A}} \Leftarrow p_a \in F_{\mathcal{A}})$
- $\forall q_b, p_b \in Q_{\mathcal{B}}.q_b \sqsubseteq_{\mathcal{B}} p_a \Rightarrow (q_a \in F_{\mathcal{B}} \Leftarrow p_b \in F_{\mathcal{B}})$

We have $q_a \sqsubseteq_{\mathcal{A}} p_a$ and $q_b \sqsubseteq_{\mathcal{B}} p_b$ by the definition of $\sqsubseteq_{\mathcal{A}\otimes\mathcal{B}}$. Assume $(p_a, p_b) \in F_{\mathcal{A}\otimes\mathcal{B}}$, i.e., $p_a \in F_{\mathcal{A}}$ and $p_b \in F_{\mathcal{B}}$. Then by $q_a \sqsubseteq_{\mathcal{A}} p_a$, $q_b \sqsubseteq_{\mathcal{B}} p_b$ and I.H., we have $q_a \in F_{\mathcal{A}}$ and $q_b \in F_{\mathcal{A}}$, i.e., $(q_a, q_b) \in F_{\mathcal{A}\otimes\mathcal{B}}$.

Case. $\mathcal{M} \equiv \mathcal{A} \oplus \mathcal{B}$

If $q, p \in Q_{\mathcal{A}}$, then by I.H., the statement holds. If $q \in Q_{\mathcal{A}}$ and $p \in Q_{\mathcal{B}}$, since $(q, p) \notin \sqsubseteq_{\mathcal{A}\oplus\mathcal{B}}$, the statement trivially holds.

Case. $\mathcal{M} \equiv \mathcal{A}.d$

I.H. is as follows: $\forall q, p \in Q_{\mathcal{A}}.q \sqsubseteq_{\mathcal{A}} p \Rightarrow (q \in F_{\mathcal{A}} \Leftarrow p \in F_{\mathcal{A}})$. Suppose $U, V \in Q_{\mathcal{A}.d}$ and $U \sqsubseteq_{\mathcal{A}.d} V$. Assume $V \in F_{\mathcal{A}.d}$. By definition of $F_{\mathcal{A}.d}$, we have $p \in V$ such that $p \in F_{\mathcal{A}.d}$. By definition of $\sqsubseteq_{\mathcal{A}.d}$, we have $q \in U$ such that $q \sqsubseteq_{\mathcal{A}} p$. Since $p \in F_{\mathcal{A}}$, by I.H., q is also in $F_{\mathcal{A}}$. We have $U \in F_{\mathcal{A}.d}$.

Case. $\mathcal{M} \equiv \mathcal{A}.c$

Suppose $q, p \in Q_{\mathcal{A}.c}$ with $q \sqsubseteq_{\mathcal{A}.c} p$. Then we have $p \sqsubseteq_{\mathcal{A}} q$. Suppose $p \in F_{\mathcal{A}.c}$, then $p \notin F_{\mathcal{A}}$. By I.H., $p \in F_{\mathcal{A}} \Leftarrow q \in F_{\mathcal{A}}$, and by taking contraposition, $q \notin F_{\mathcal{A}.c} \Leftarrow p \notin F_{\mathcal{A}.c}$ also holds. We have $q \in F_{\mathcal{A}.c}$.

□

Lemma 5.10. *Let $\mathcal{M}$ be a composition term and $q, p \in Q_{\mathcal{M}}$ Let $c \in \bar{\Sigma}$.*

$$\forall p' \in \Delta_{\mathcal{M}}(p, c).\ \exists q' \in \Delta_{\mathcal{M}}(q, c).\ q \sqsubseteq_{\mathcal{M}} p \Rightarrow q' \sqsubseteq_{\mathcal{M}} p'$$

Proof. By induction on the structure of $\mathcal{M}$.

Base case. $\mathcal{M} \equiv \mathcal{A}_0$

Assume $q \sqsubseteq_{\mathcal{A}_0} p$ and let $p' \in Q_{\mathcal{A}_0}$ with $p' \in \Delta_{\mathcal{A}_0}(p, c)$. By definition, $q = p$. Hence we have $p' \in \Delta_{\mathcal{A}_0}(q, c)$ and $p' \sqsubseteq_{\mathcal{A}_0} p'$.

Inductive step.

Case. $\mathcal{M} \equiv \mathcal{A} \otimes \mathcal{B}$

Let $(q_a, q_b), (p_a, p_b) \in Q_{\mathcal{A}\otimes\mathcal{B}}$ and we assume $(q_a, q_b) \sqsubseteq_{\mathcal{A}\otimes\mathcal{B}} (p_a, p_b)$. We have the I.H.

- $\forall p'_a \in \Delta_{\mathcal{A}}(p, c).\ \exists q'_a \in \Delta_{\mathcal{A}}(q, c).\ q \sqsubseteq_{\mathcal{A}} p \Rightarrow q' \sqsubseteq_{\mathcal{A}} p'$
- $\forall p'_b \in \Delta_{\mathcal{B}}(p, c).\ \exists q'_b \in \Delta_{\mathcal{B}}(q, c).\ q \sqsubseteq_{\mathcal{B}} p \Rightarrow q' \sqsubseteq_{\mathcal{B}} p'$

Let $(p'_a, p'_b) \in \Delta_{\mathcal{A}\otimes\mathcal{B}}((p_a, p_b), c)$. By definition of $\sqsubseteq_{\mathcal{A}\otimes\mathcal{B}}$, we have $q_a \sqsubseteq_{\mathcal{A}} p_a$ and $q_b \sqsubseteq_{\mathcal{B}} p_b$. By definition of $\Delta_{\mathcal{A}\otimes\mathcal{B}}$, we also have $p'_a \in \Delta_{\mathcal{A}}(p_a, c)$ and $p'_b \in \Delta_{\mathcal{B}}(p_b, c)$. By I.H., we have $q'_a \in \Delta_{\mathcal{A}}(q_a, c)$ such that $q'_a \sqsubseteq_{\mathcal{A}} p'_a$. Also by I.H., we have $q'_b \in \Delta_{\mathcal{B}}(q_b, c)$ such that $q'_b \sqsubseteq_{\mathcal{B}} p'_b$. By definition of $\Delta_{\mathcal{A}\otimes\mathcal{B}}$, $(q'_a, q'_b) \in \Delta_{\mathcal{A}\otimes\mathcal{B}}((q_a, q_b), c)$, and by definition of $\sqsubseteq_{\mathcal{A}\otimes\mathcal{B}}$, $(q'_a, q'_b) \sqsubseteq_{\mathcal{A}\otimes\mathcal{B}} (p'_a, p'_b)$.

Case. $\mathcal{M} \equiv \mathcal{A} \oplus \mathcal{B}$

Let $q, p \in Q_{\mathcal{A}\oplus\mathcal{B}}$. There are 2 cases: 1. $q \in Q_{\mathcal{A}}$ and $q \in Q_{\mathcal{B}}$ (or $q \in Q_{\mathcal{B}}$ and $q \in Q_{\mathcal{A}}$). 2. $q \in Q_{\mathcal{A}}$ and $q \in Q_{\mathcal{A}}$ (or $q \in Q_{\mathcal{B}}$ and $q \in Q_{\mathcal{B}}$). If case 1, then $(q, p) \notin \sqsubseteq_{\mathcal{A}\oplus\mathcal{B}}$. The statement trivially holds. If case 2, the I.H. satisfies the statement.

Case. $\mathcal{M} \equiv \mathcal{A}.d$

Let $U, V \in Q_{\mathcal{A}.d}$. We assume $U \sqsubseteq_{\mathcal{A}.d} V$. Let $V' \in \Delta_{\mathcal{A}.d}(V, c)$ and $U' \in \Delta_{\mathcal{A}.d}(U, c)$. Since $V' \in \{ \bigcup_{p \in V} \Delta_{\mathcal{A}}(p, c)\}$, $V' = \bigcup_{p \in V} \Delta_{\mathcal{A}}(p, c)$. Similarly $U' = \bigcup_{q \in U} \Delta_{\mathcal{A}}(q, c)$. We show $U' \sqsubseteq_{\mathcal{A}.d} V'$. Let $p' \in V'$. Then we have $p \in V$ such that $p \xrightarrow{c} p'$ since $V' = \bigcup_{p \in V} \Delta_{\mathcal{A}}(p, c)$. By definition of $\sqsubseteq_{\mathcal{A}.d}$, $\forall p \in V.\ \exists q \in U.\ q \sqsubseteq_{\mathcal{A}} p$. Hence we can take $q \in U$ such that $q \sqsubseteq_{\mathcal{A}} p$. Since $q \in U$, we have $q' \in \Delta_{\mathcal{A}}(q, c)$ with $q' \in U'$. By I.H., we have $q' \sqsubseteq_{\mathcal{A}} p'$. For arbitrary $p' \in V'$, we can find the element $q' \in U'$ such that $q' \sqsubseteq_{\mathcal{A}} p'$. We conclude $U' \sqsubseteq_{\mathcal{A}.d} V'$.

Case. $\mathcal{M} \equiv \mathcal{A}.c$

We have 2 cases: 1. $\mathcal{A}$ is *non-deterministic*. 2. $\mathcal{A}$ is *deterministic*. If case 1, then $\mathcal{A}.c$ is not *well formed*. The statement trivially holds since the assumption is not satisfied, If case 2, $\Delta_{\mathcal{A}.c}$ returns singleton of a state. We have single $p' \in \Delta_{\mathcal{A}.c}(p, c)$. Assume $q \sqsubseteq_{\mathcal{A}.c} p$. By definition of $\sqsubseteq_{\mathcal{A}.c}$, we have $p \sqsubseteq_{\mathcal{A}} q$. Let $q' \in \Delta_{\mathcal{A}.c}(q, c)$. Since p' is taken from the singleton $\Delta_{\mathcal{A}.c}(p, c)$, by I.H., $p' \sqsubseteq_{\mathcal{A}} q'$. Then we have $q' \sqsubseteq_{\mathcal{A}.c} p'$.

□

Lemma 5.11. *Let $x \in \Sigma^*$, $\mathcal{M}$ be a composition term. $\forall q \in \hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x)$. q is minimal w.r.t. $\sqsubseteq_{\mathcal{M}}$ in $\hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x) \Rightarrow \exists q' \in \mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(X))$. $q' \sqsubseteq_{\mathcal{M}} q$*

Proof. By induction on the length of x.

Base case. $x = \epsilon$

Suppose $q \in \hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, \epsilon) = I_{\mathcal{M}}$ and minimal w.r.t. $\sqsubseteq_{\mathcal{M}}$ in $I_{\mathcal{M}}$. If $q \in \mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(X))$, the statement holds for the base case. Otherwise we have $min_{\sqsubseteq_{\mathcal{M}}}(I_{\mathcal{M}}) \subseteq \mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq}\mathcal{M}(X))$ and there exists $q' \in min_{\sqsubseteq_{\mathcal{M}}}(I_{\mathcal{M}})$ such that $q' \sqsubseteq_{\mathcal{M}} q$.

Inductive step. $x = x'c$

We have I.H., $\forall q \in \hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x')$. q is minimal w.r.t. $\sqsubseteq_{\mathcal{M}}$ in $\hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x') \Rightarrow \exists q' \in \mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(X))$. $q' \sqsubseteq_{\mathcal{M}} q$. Let $q \in \hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x'c)$ and suppose q is minimal w.r.t. $\sqsubseteq_{\mathcal{M}}$ in $\hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x'c)$. Since $q \in \bigcup_{p \in \hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x')} \Delta_{\mathcal{M}}(p, c)$, let $p \in \hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x')$ such that $q \in \Delta_{\mathcal{M}}(p, c)$. Then we have p is minimal w.r.t. $\sqsubseteq_{\mathcal{M}}$ in $\hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x')$. It is because, otherwise there exists $p' \in \hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x')$ with $p' \sqsubseteq_{\mathcal{M}} p$. By Lemma 5.10 there exists $q' \in \Delta_{\mathcal{M}}(p', c)$ such that $q' \subseteq_{\mathcal{M}} q$. This contradicts q being minimal in $\hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x'c)$. By I.H., we have $p' \in \mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(X))$ with $p' \sqsubseteq_{\mathcal{M}} p$. By Lemma 5.10 there exists $q' \in \Delta_{\mathcal{M}}(p', c)$ such that $q' \subseteq_{\mathcal{M}} q$.

$$
\begin{aligned}
& q' \in Post_{\mathcal{M}}(\{p'\}) \subseteq I_{\mathcal{M}} \cup Post_{\mathcal{M}}(\mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(X)) \\
\Leftrightarrow\ & q' \in succ_{\mathcal{M}}(\mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(X))
\end{aligned}
$$

If $q' \in succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(\mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(X)))$, then $q' \in \mu X.\ (succ_{\mathcal{M}} \circ min_{\sqsubseteq_{\mathcal{M}}}(X))$. Otherwise we have $q'' \in min_{\mathcal{M}}(succ_{\mathcal{M}}(\mu X.\ (min_{\mathcal{M}}(succ_{\mathcal{M}}(X)))))$ such that $q'' \sqsubseteq q' \sqsubseteq q$ and $q'' \in \mu X.\ (min_{\mathcal{M}}(succ_{\mathcal{M}}(X)))$. □

Lemma 5.12. $\mu X.\ (succ_{\mathcal{M}} \circ min_{\mathcal{M}}(X)) \cap F_{\mathcal{M}} = \varnothing \Leftrightarrow L(\mathcal{M}) = \varnothing$

Theorem 5.13. $\texttt{EC}_{\sqsubseteq}$ *is sound and complete.*

6 Conversion rules

Before constructing the automaton for a given input formula, there are chances to decompose a problem into the smaller subproblems. It is natural to apply distributive laws of $\wedge, \vee$, and $\neg$. Those conversions preserve the logical equivalence of the formula. On the other hand, since it suffices to maintain the satisfiability of the formula, we can further convert the composition terms, i.e., equisatisfiable conversions. The use of "language terms" and the rewriting rules over the terms are proposed and applied to WS1S [7]. They observed that

1. Anti-prenexing moves an existential quantifier down in the AST of a formula. This technique is most effective among others.
2. $\neg$ is pushed down to bottom of the AST. Since the size of an automaton is smaller in the bottom part than upper part, determinization costs less.

Although our motivation is quite similar, the main difference is that our target normal form is not a prenex normal form, but a minimally quantified form, which optimizes the generalized antichain algorithm. We further introduce the distributive laws of the emptiness checking which respect the equisatisfiability.

6.1 Logically equivalent conversions

Since the language remains unchanged, we regard certain sequences of symbols as a single symbol. $\mathcal{A}.d.c$ is regarded as $\mathcal{A}.dc$ and $\mathcal{A}.p_i.p_j.\ldots.p_k$ is regarded as $\mathcal{A}.p_ip_j\ldots p_k$.

Definition 6.1. Logically equivalent rules corresponds to distributive law of negation, conjunction and disjunction and quantifiers.

$$
\begin{array}{lcll}
\mathcal{A}.dc.dc & \overset{VR}{\longrightarrow} & \mathcal{A} & (VR_1) \\
(\mathcal{A} \otimes \mathcal{B}).dc & \overset{VR}{\longrightarrow} & \mathcal{A}.dc \oplus \mathcal{B}.dc & (VR_2) \\
(\mathcal{A} \oplus \mathcal{B}).dc & \overset{VR}{\longrightarrow} & \mathcal{A}.dc \otimes \mathcal{B}.dc & (VR_3) \\
(\mathcal{A} \oplus \mathcal{B}).p_i & \overset{VR}{\longrightarrow} & \mathcal{A}.p_i \oplus \mathcal{B}.p_i & (VR_5) \\
((\mathcal{A} \oplus \mathcal{B}) \otimes \mathcal{G}) & \overset{VR}{\longrightarrow} & (\mathcal{A} \otimes \mathcal{G}) \oplus (\mathcal{B} \otimes \mathcal{G}) & (VR_6)
\end{array}
$$

In addition, we can derive the rule from VR, $(\mathcal{A} \otimes \mathcal{B}).dc.p_i.dc \overset{VR}{\longrightarrow} \mathcal{A}.dc.p_i.dc \otimes \mathcal{B}.dc.p_i.dc$. A normal form of a composition term t is called a minimally quantified form. From Proposition6.1, it is uniquely determined.

Proposition 6.1. *VR is terminating and confluent.*

Proof. $\overset{VR}{\to}$ is terminating by AC-RPO with the precedence $d, c, p_i, \succ \otimes \succ \oplus$. $\overset{VR}{\to}$ is locally confluent, since the critical pairs occur between VR_1 and VR_2, and VR_1 and VR_3 which are joinable. Thus, by Newman's lemma, VR_1 and $\overset{VR}{\to}$ is confluent. □

Lemma 6.1. *Let ct be a composition term and ct' be a normal form of VR, i.e., $ct \overset{VR}{\longrightarrow}_* ct'$.*

$$\forall p \in \mathcal{P}os(ct').\ \forall i \in \{1,2\}.\ ct'|_{pi} = (\_ \oplus \_) \Rightarrow ct'|_p = (\_ \oplus \_)$$

Proof. By contradiction. Suppose $ct'|_{pi} = (\_ \oplus \_)$ and assume $ct'|_p \neq (\_ \oplus \_)$. For $ct'|_p$ we have the following 3 cases; Case $ct'|_p = (\_ \otimes \_)$. We have $ct'|_p = ((\_ \oplus \_) \otimes \_)$ and $ct'|_p$ has a redex, which contradicts the fact that ct' is a normal form of VR. In other cases where $ct'|_p = (\_ \oplus \_).dc$, and $ct'|_p = (\_ \oplus \_).p_i$, we find the redex of VR. We conclude that our assumption is wrong. □

Lemma 6.2. $\forall p, p' \in \mathcal{P}os(ct').\ p \leq p' \wedge ct'|_{p'} = (\_ \oplus \_) \Rightarrow ct'|_p = (\_ \oplus \_)$

Proof. For arbitrary p, let l, l' be length of p, p' respectively. By induction on $n = l' - l$.

Base case.

$n = 0$, we have $p = p'$, and $ct'|_{p'} = (\_ \oplus \_) \Rightarrow ct'|_p = (\_ \oplus \_)$.

Inductive step.

$n = k + 1$, we have I.H. that $ct'|_{pi_1 i_2 \ldots i_k} = (\_ \oplus \_) \Rightarrow ct'|_p = (\_ \oplus \_)$. We suppose $ct'|_{pi_1 i_2 \ldots i_k i_{k+1}} = (\_ \oplus \_)$. Then from Lemma 6.1, $ct'|_{pi_1 i_2 \ldots i_k} = (\_ \oplus \_)$. By applying I.H., we have $ct'|_p = (\_ \oplus \_)$. □

6.2 Equisatisfiable conversions

In addition, we define equisatisfiable rules for the answer of *Emptiness Checking*. The application of these rules may change the language of composition terms, whereas its emptinessis preserved.

Definition 6.2.

$$CR_1\ \frac{\mathcal{A} \rightarrow_{\texttt{EC}} \texttt{Empty}}{\mathcal{A}.d \rightarrow_{\texttt{EC}} \texttt{Empty}} \quad CR_2\ \frac{\mathcal{A} \rightarrow_{\texttt{EC}} \texttt{Empty}}{\mathcal{A}.p_i \rightarrow_{\texttt{EC}} \texttt{Empty}}$$

$$CR_3\ \frac{\mathcal{A} \rightarrow_{\texttt{EC}} \texttt{Empty} \quad \mathcal{B} \rightarrow_{\texttt{EC}} \texttt{Empty}}{(\mathcal{A} \oplus \mathcal{B}) \rightarrow_{\texttt{EC}} \texttt{Empty}}$$

Lemma 6.3. $L(\mathcal{A}) = \varnothing \Leftrightarrow L(\mathcal{A}.d) = \varnothing$

Proof. By $L(\mathcal{A}) = L(\mathcal{A}.d)$ □

Lemma 6.4. $L(\mathcal{A}) = \varnothing \Leftrightarrow L(\mathcal{A}.p_i) = \varnothing$

Proof. Equivalently, we show $\mu X.\ (succ_{\mathcal{A}}(X)) \cap F_{\mathcal{A}} = \varnothing \Leftrightarrow \mu X.\ (succ_{\mathcal{A}.p_i}(X)) \cap F_{\mathcal{A}.p_i} = \varnothing$ By definition, $F_{\mathcal{A}} = F_{\mathcal{A}.p_i}$ and $I_{\mathcal{A}} = I_{\mathcal{A}.p_i}$. Also we have $Post_{\mathcal{A}} = Post_{\mathcal{A}.p_i}$ since,

$$\begin{aligned} Post_{\mathcal{A}.p_i}(s) &= \bigcup_{\bar{c} \in \bar{\Sigma}} \bigcup_{q \in s} \Delta_{\mathcal{A}.p_i}(q, \bar{c}) \\ &= \bigcup_{\bar{c} \in \bar{\Sigma}} \bigcup_{q \in s} \bigcup_{c \in \Sigma} \Delta_{\mathcal{A}.p_i}(q, \pi_i(\bar{c}, c)) \\ &= \bigcup_{\bar{c} \in \bar{\Sigma}} \bigcup_{q \in s} \Delta_{\mathcal{A}}(q, \bar{c}) = Post_{\mathcal{A}}(s) \end{aligned}$$

Thus we have $\mu X.\ (succ_{\mathcal{A}}(X)) = \mu X.\ (succ_{\mathcal{A}.p_i}(X))$. We conclude that $L(\mathcal{A}) = \varnothing \Leftrightarrow L(\mathcal{A}.p_i)$. □

Lemma 6.5. $L(\mathcal{A} \oplus \mathcal{B}) = \varnothing \Leftrightarrow L(\mathcal{A}) = \varnothing \wedge L(\mathcal{B}) = \varnothing$

Proof. By $L(\mathcal{A} \oplus \mathcal{B}) = L(\mathcal{A}) \cup L(\mathcal{B})$ □

Lemma 6.6. *Let $\mathcal{A}$ and $\mathcal{A}'$ be ct s.t. $\mathcal{A} \overset{VR*}{\longrightarrow} \mathcal{A}'$.*

7 Implementation and Experiments

In this section, we explain our tool and its implementation and describe the experimental results. The tool takes First-Order Presburger formula as the input and answers whether there exist satisfiable assignments for the free variables in the formula. We don't restrict the formula to be closed. The syntactical depth of the formula represents repetition of the regular operations on the automata while number of variables represents the size of Σ of the language. For we have 4 approaches, we compare the running time of those 4 cases.

7.1 Tool

We implemented those algorithms in OCaml. A user specifies one of 4 algorithms; On-the-fly construction without any other techniques (`None`), term conversion only (`Ct`), minimization of the antichain algorithm only (`Ac`), and both of these techniques (`CtAc`). The user gives the number of variables in addition to a Presburger formula. Below is an example of the input form;

```
p 3 and(not(exists(x0, exists(x2, eq(r, 1x0 + 2x1  −3x2 = 2)))),
    exists(x0, eq(g, 3x0 + 1x1 + 2x2 = 1)))
```

7.2 Data set and the experiment method

Our experimental data are First-Order Presburger formulas. The formulas are classified under 1. syntactical depth in terms of their parse tree and 2. number of variables occurring in the formula. We obtain the data in the following manner; for atomic formula, 1a. each coefficient a_i is randomly taken from $0 < a_i \leq 20$ and cannot be 0, while right hand side of the equation is also a random constant c with $0 \leq c \leq 100$. 1b. some atomic formula are excluded if the set of the solutions are scarce. In order to make formula out of the set of atomic formulas, we first randomly generate AST with a specified depth, then fill each node with one of $\neg, \exists$ for unary node, and $\wedge, \vee$ for binary node. The variable bound by $\exists$ is also randomly chosen. We prepare 500 problems for each classification by syntactical depth 3 or 4, by number of variables 2, 3 or 4. Timeout is set to 5 minutes. We collect the running time, the size of the generated set of states and the answer `Empty` or `NonEmpty`.

`Ac` requires additional computation of states comparison, which becomes overhead for small problems. For problems with variables 2 or 3, the overhead is not compensated and affect the performance (From figure 4 and 5). A time record of a problem is excluded from Total Time shown in the table 1, right column when at least one of 4 algorithms failed as timeout. We cannot maintain that the generalized antichain algorithm immediately leads to universal performance improvement. At least we can conclude for the problems with depth 4 and 4 variables, especially whose sizes exceed 3000, surely our optimization techniques both `Ac` and `CtAc` outperform `None` and `Ct`.

7.3 Experimental results

Our observation is twofold;

1. Generalized antichain algorithm enjoys the performance improvement for sufficiently large and complex problems. Due to the overhead of calculating orderings, it does not work for small problems. The threshold would be the formulas with depth 4, 4 variables and more than 3000 state size. Regardless of logics (WS1S, Presburger, and so on), we expect the threshold applies.

2. In the most cases, minimally quantified formof the composition term leads to performance improvement. It also implies that normalization not dedicated to the regular language

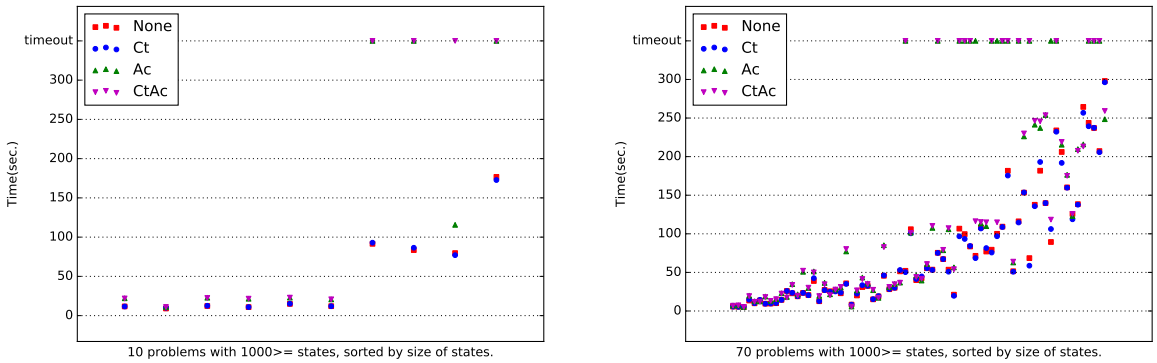

Figure 4: Results for depth 3 and depth 4, with 2 variables.

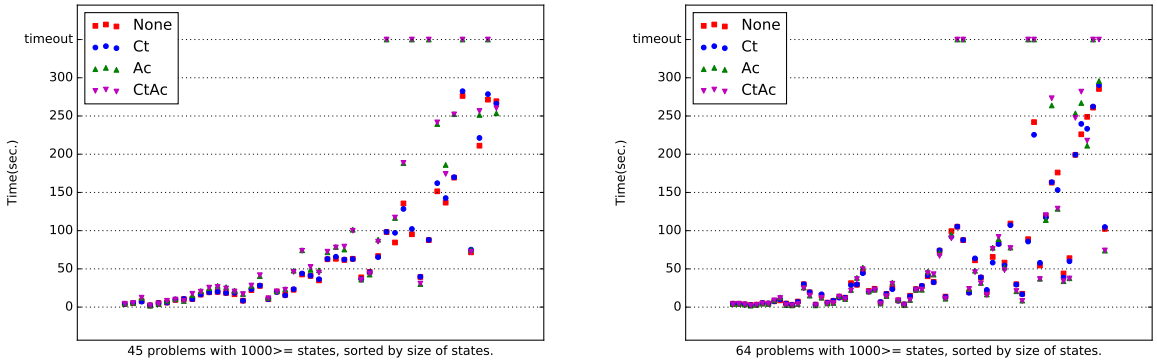

Figure 5: Results for depth 3 and depth 4, with 3 variables.

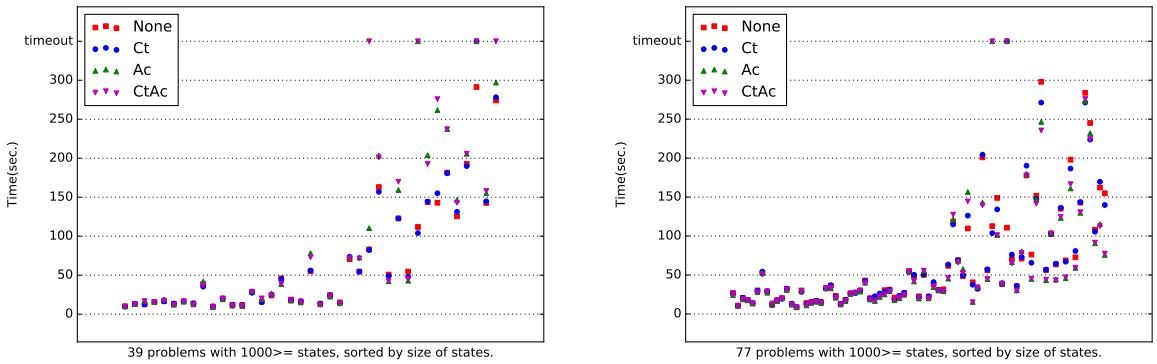

Figure 6: Results for depth 3 and depth 4, with 4 variables.

Table 1: Results in frequency distribution table.

D3V2	Success/ Timeout				Total Time			
Size	None	Ct	Ac	CtAc	None	Ct	Ac	CtAc
0 - 1000	488/ 0	488/ 0	488/ 0	488/ 0	51.04	51.3	74.62	73.79
1000 - 2000	6/ 0	6/ 0	6/ 0	6/ 0	70.91	72.36	121.0	120.79
2000 - 3000	3/ 0	3/ 0	1/ 2	0/ 3	0	0	0	0
3000 - 4000	1/ 0	1/ 0	0/ 1	0/ 1	0	0	0	0
4000 - 5000	0/ 0	0/ 0	0/ 0	0/ 0	0	0	0	0
5000 -	0/ 0	0/ 0	0/ 0	0/ 0	0	0	0	0

D3V3	Success/ Timeout				Total Time			
Size	None	Ct	Ac	CtAc	None	Ct	Ac	CtAc
0 - 1000	438/ 0	438/ 0	438/ 0	438/ 0	242.16	242.98	253.79	253.49
1000 - 2000	27/ 0	27/ 0	27/ 0	27/ 0	585.33	595.97	776.15	777.68
2000 - 3000	8/ 0	8/ 0	6/ 2	6/ 2	435.12	435.94	572.37	570.85
3000 - 4000	8/ 0	8/ 0	6/ 2	6/ 2	779.57	810.81	1033.87	1027.4
4000 - 5000	2/ 0	2/ 0	1/ 1	1/ 1	269.3	266.09	253.52	260.2
5000 -	0/ 0	0/ 0	0/ 0	0/ 0	0	0	0	0

D3V4	Success/ Timeout				Total Time			
Size	None	Ct	Ac	CtAc	None	Ct	Ac	CtAc
0 - 1000	422/ 0	422/ 0	422/ 0	422/ 0	3084.79	3087.87	3093.95	3084.75
1000 - 2000	23/ 0	23/ 0	23/ 0	23/ 0	461.22	454.08	480.28	465.65
2000 - 3000	9/ 0	9/ 0	8/ 1	7/ 2	659.73	649.15	796.76	794.21
3000 - 4000	5/ 1	5/ 1	5/ 1	5/ 1	642.62	657.05	851.35	861.66
4000 - 5000	1/ 0	1/ 0	1/ 0	1/ 0	142.64	144.8	155.25	157.9
5000 -	1/ 0	1/ 0	1/ 0	0/ 1	0	0	0	0

D4V2	Success/ Timeout				Total Time			
Size	None	Ct	Ac	CtAc	None	Ct	Ac	CtAc
0 - 1000	376/ 0	376/ 0	376/ 0	376/ 0	119.07	119.47	133.39	132.27
1000 - 2000	33/ 0	33/ 0	32/ 1	32/ 1	680.0	692.51	897.08	888.12
2000 - 3000	26/ 0	26/ 0	15/ 11	18/ 8	1288.9	1294.83	1841.45	1868.93
3000 - 4000	9/ 0	9/ 0	5/ 4	5/ 4	894.87	864.89	940.43	940.82
4000 - 5000	1/ 0	1/ 0	1/ 0	1/ 0	297.99	296.31	248.49	259.15
5000 -	0/ 0	0/ 0	0/ 0	0/ 0	0	0	0	0

D4V3	Success/ Timeout				Total Time			
Size	None	Ct	Ac	CtAc	None	Ct	Ac	CtAc
0 - 1000	328/ 0	328/ 0	328/ 0	328/ 0	348.25	348.34	343.36	342.04
1000 - 2000	32/ 0	32/ 0	32/ 0	32/ 0	417.92	412.44	396.24	385.84
2000 - 3000	17/ 0	17/ 0	15/ 2	15/ 2	773.21	757.53	724.91	710.1
3000 - 4000	10/ 0	10/ 0	8/ 2	8/ 2	837.52	806.61	876.63	886.04
4000 - 5000	5/ 0	5/ 0	4/ 1	3/ 2	577.3	577.74	551.92	573.84
5000 -	0/ 0	0/ 0	0/ 0	0/ 0	0	0	0	0

D4V4	Success/ Timeout				Total Time			
Size	None	Ct	Ac	CtAc	None	Ct	Ac	CtAc
0 - 1000	302/ 0	302/ 0	302/ 0	302/ 0	4999.64	5009.72	4994.13	4989.78
1000 - 2000	34/ 0	34/ 0	34/ 0	34/ 0	794.41	789.95	745.69	745.23
2000 - 3000	20/ 0	20/ 0	19/ 1	19/ 1	1086.41	1105.43	1015.82	1001.12
3000 - 4000	11/ 0	10/ 1	10/ 1	10/ 1	1123.95	1090.03	979.94	956.93
4000 - 5000	7/ 1	7/ 1	8/ 0	8/ 0	783.53	781.5	667.18	674.66
5000 -	5/ 1	5/ 1	6/ 0	5/ 1	954.08	910.19	786.74	782.53

operations, namely prenex normal form, could affect the performance in the automata construction step.

8 Conclusion

We developed an efficient algorithm for automata theoretic theorem proving expanding antichain algorithms. Our aim is to directly handle a nested formula with an antichain algorithm (i.e., without flattening). We introduced composition terms that represents automata construction so that the generalized antichain algorithm is inductively defined for the structure of composition terms.

1. As an optimization, we further introduced conversion rules of composition terms which preserves the accepted language distributive laws of emptiness checking into a composition terms.

2. We perform experiments on randomly generated 3000 Presburger formulas. Generalized antichain algorithm improved the performance for sufficiently large and complex problems. Due to the overhead of calculating orderings, it did not work for small problems. In the most cases, conversion of the composition term led to performance improvement. It also implies that normalization not dedicated to the regular language operations, namely prenex normal form, could affect the performance in the automata construction step.

References

[1] P. A. Abdulla, Y. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In *Proc. TACAS*, vol. 6015 *Lecture Notes in Computer Science*, pages 158–174. Springer, 2010.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[3] A. Bouajjani, P. Habermehl, L. Holík, T. Touili, and T. Vojnar. Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In *Proc. CIAA*, volume 5148 of *Lecture Notes in Computer Science*, pages 57–67. Springer, 2008.

[4] A. Boudet and H. Comon. Diophantine equations, presburger arithmetic and finite automata. In *Proc. CAAP'96*, volume 1059 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 1996.

[5] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[6] J. Elgaard, N. Klarlund, and A. Møller. MONA 1.x: New techniques for WS1S and WS2S. In *Proc. CAV* , volume 1427 of *Lecture Notes in Computer Science*, pages 516–520. Springer, 1998.

[7] T. Fiedor, L. Holík, P. Janku, O. Lengál, and T. Vojnar. Lazy automata techniques for WS1S. *CoRR*, abs/1701.06282, 2017.

[8] T. Fiedor, L. Holík, O. Lengál, and T. Vojnar. Nested antichains for WS1S. In *Proc. TACAS* , volume 9035 of *Lecture Notes in Computer Science*, pages 658–674. Springer, 2015.

[9] D. Harel, J. Tiuryn, and D. Kozen. *Dynamic Logic.* MIT Press, Cambridge, MA, USA, 2000.

[10] G. Holzmann. On-the-fly model checking. *ACM Comput. Surv.*, 28(4es), Dec. 1996.

[11] L.Doyen and J.-F.Raskin. Improved algorithms for the automata-based approach to model-checking. In *Proc. TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 451–465. Springer-Verlag, 2007.

[12] V. B. Marc and D. O. Gauwin. Visibly pushdown automata: Universality and inclusion via antichains. In *LATA*, volume 7810, pages 190–201. Springer, 2013.

[13] M.D.Wulf, L.Doyen, T.A.Henzinger, and J.-F.Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proc. CAV 2006*, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer-Verlag, 2006.

[14] R.Alur and P.Madhusudan. Visibly pushdown languages. In *Proc. the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 202–211, 2004.

[15] F. Rapp and A. Middeldorp. Automating the first-order theory of rewriting for left-linear right-ground rewrite systems. In *FSCD* , pages 36:1–36:12, 2016.

[16] N. V. Tang. *Pushdown Automata and Inclusion Problems.* PhD thesis, JAIST, 3 2009.

[17] P. Techaveerapong. *Antichain algorithm, its theory and appilcations.* Master thesis, JAIST, 3 2009.

[18] W. Thomas. Handbook of formal languages, vol. 3. chapter Languages, Automata, and Logic, pages 389–455. Springer-Verlag , 1997.

[19] M. D. Wulf, L. Doyen, T. A. Henzinger, and J. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proc. CAV* , volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2006.

[20] M. D. Wulf, L. Doyen, N. Maquet, and J. Raskin. Alaska. In *Proc. ATVA*, volume 5311 of *Lecture Notes in Computer Science*, pages 240–245. Springer, 2008.