修士論文

Extensions and Applications of Antichain Algorithms

1510025	白木澤 啓

北陸先端科学技術大学院大学

情報科学研究科

平成 29 年 2 月

Contents

Chapter 1

Introduction

This thesis is the contribution to the study of automata-theoretic theorem proving. Automata-theoretic theorem proving checks the satisfiability / validity of a first-order formula under a fixed interpretation. 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 Büchi's theorem for Weak monadic second-order logic of one successor (WS1S) and S1S [18]. WS1S (resp. S1S) has the set of predicates $\{\subseteq, \mathit{Sing}, = \{0\}\}$ and the set of function symbols $\{\mathit{Succ}, 0\}$. 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 $\mathbb{N}$ in a standard manner. Here $\mathit{Sing}(X)$ means X is a singleton and $X = \{0\}$ means X is the constant $\{0\}$. Satisfiability / validity are decidable for a WS1S formula. Suppose that ϕ has n free variables, then a set of strings over n-tuple of $\{0, 1\}$ 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 connective, we conduct language operations. From closure properties of these operations, the resulting automaton recognizes the support of the formula ϕ. The same story holds for WSkS and SkS, by extending finite automata to tree automata.

There are several existing tools for automata-theoretic theorem proving. MONA [6] translates formulas in weak monadic second-order logic of k successor (WSkS) to finite tree automata. Recently FORT [15] is implemented for the first-order theory of term rewriting. It is based on tree automata and ground tree transducers (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 height 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, an additional 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 the 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 prenex 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.

We evaluate them 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.

Chapter 2

Preliminaries

We introduce notations on automata. We assume basic knowledge of set theory and automata theory.

2.1 Finite automata and closure property

Definition 2.1.1. Let Σ be a finite alphabet. A finite automaton (FA) $\mathcal{M}$ on finite 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 $I, F \subseteq Q$ are the sets of initial and final states, respectively.

FAs are denoted by calligraphic symbols $\mathcal{M}, \mathcal{A}, ...$ We use the subscript to refer each component of the automaton $\mathcal{M}$, e.g., the set of states of $\mathcal{M}$ is referred to as $Q_{\mathcal{M}}$. We use these letters a, b, c, τ to denote elements of Σ and x a string in Σ^*. We write xc for the concatenation of $x \in \Sigma^*$ and $c \in \Sigma$. Small letters $p, q, ..$ denote states and large letters $S, T, U, V, ..$ denote sets of states. We impose several assumptions on finite automata.

- A standard ϵ-elimination procedure ensures that $\delta_{\mathcal{M}}$ has no ϵ -transition.
- Except for initial states, all states have an incoming transition edge.
- By adding the black-hole state, all states have outgoing transition edges.
- Among automata $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, .., they share the same input alphabet Σ.
- For any different automata $\mathcal{A}$ and $\mathcal{B}$, the state sets are mutually disjoint, i.e., $Q_{\mathcal{A}} \cap Q_{\mathcal{B}} = \varnothing$.

Definition 2.1.2. A transition relation $\delta_{\mathcal{M}}$ 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 (NFA) $\mathcal{M}$, reading one character on a state, it may have multiple destination states in the transition relation $\delta_{\mathcal{M}}$. A transition function for NFA takes a character and a state and maps to set of states in the transition relation. We define the transition function $\Delta_{\mathcal{M}}$ as follows:

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

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

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

Note that set operator $\bigcup_i$ commutes: $\bigcup_i \hat{\Delta}(S_i, x) = \hat{\Delta}(\bigcup_i S_i, x)$. Further, $\bigcup_{p \in S} \hat{\Delta}(\{p\}, x) = \hat{\Delta}(S, x)$ holds. For $\hat{\Delta}$, we prepare another inductive definition with respect to the word length.

Proposition 2.1.1. *Let $S \subseteq Q_{\mathcal{M}}$, $x \in \Sigma^*$, and $c \in \Sigma$. $\hat{\Delta}_{\mathcal{M}}(S, xc) = \bigcup_{q \in \hat{\Delta}_{\mathcal{M}}(S,x)} \Delta_{\mathcal{M}}(q, c)$.*

We say that 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 $p \in \Delta_{\mathcal{M}}(q, c)$. For $x \in \Sigma^*$, we denote $q \xrightarrow{x} p$ if $p \in \hat{\Delta}_{\mathcal{M}}(\{q\}, x)$. The language of $\mathcal{M}$ is the set of words accepted by $\mathcal{M}$ and is denoted by $L(\mathcal{M})$.

In order to collect a set of reachable states from S, we define the following functions.

$$\begin{aligned} post_{\mathcal{M}}(c, S) &:= \{q' \in Q_{\mathcal{M}} \mid \exists\ q \in S.\ (q, c, q') \in \delta_{\mathcal{M}}\} \\ Post_{\mathcal{M}}(S) &:= \bigcup_{c \in \Sigma} \bigcup_{q \in S} \Delta_{\mathcal{M}}(q, c) \end{aligned}$$

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

Definition 2.1.3. Let $\mathcal{U}$ be a set and $f\ : 2^{\mathcal{U}} \to 2^{\mathcal{U}}$ a set operator on $\mathcal{U}$. We say f is *monotone* if $S \subseteq T \Rightarrow f(S) \subseteq f(T)$ and f is *finitary* if $f(S)$ consists of finite subsets of S, i.e., $f(S) = \bigcup_{T \underset{fin}{\subseteq} S} f(T)$, where $T \underset{fin}{\subseteq} S$ denotes that T is a finite subset of S. Let X be a set variable. The notation $\mu X.\ f(X)$ stands for the least fixpoint of f, the point where $X = f(X)$ holds.

The fixpoint $\mu X.\ f(X)$ 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 2.1.1. *Let $\mathcal{M}$ be an NFA. $Post_{\mathcal{M}}$ is monotone.*

Definition 2.1.4. Given a partially ordered set $\langle \mathcal{U}, \sqsubseteq \rangle$, *antichain* is a subset $S \subseteq \mathcal{U}$ containing only incomparable elements, i.e., $\forall s, s' \in S.\ s \not\sqsubseteq s'$.

Definition 2.1.5. An element $s \in S$ is *minimal* with respect to $\sqsubseteq$ if $\forall s' \in S.\ s' \not\sqsubseteq s$. Let $min_{\sqsubseteq}(S) := \{s \in S \mid s$ is *minimal* with respect to $\sqsubseteq$ in $S\}$

Theorem 2.1.2. *The class of regular languages is closed under union, intersection and complement operations.*

2.2 Automata-theoretic theorem proving

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, among automata theoretic theorem proving, 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 variables x_1, x_2. There are several existing tools for automata-theoretic theorem proving. MONA [6] translates formulas in weak monadic second-order logic of k successor (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 height of quantifiers in the input formula. Thus, in theory, its satisfiability / validity checking problem of is nonelementary [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 $\overline{\Sigma}$ denote $\Sigma \times \ldots \times \Sigma$, n -tuple of Σ. The n-tuple $\begin{smallmatrix} c_1 \\ \vdots \\ c_n \end{smallmatrix}$ is the element of $\overline{\Sigma}$ and we denote it by $\overline{c}$. $\pi_i(\overline{a}, c)$ substitutes the i-th element of $\overline{a}$ to c. We denote $\pi_i(\overline{a}, c) := \begin{smallmatrix} \vdots \\ a_{i-1} \\ c \\ a_{i+1} \\ \vdots \end{smallmatrix}$.

Let $\mathcal{A}$ be a FA with the alphabet $\overline{\Sigma}$. 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.

Related work 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 prenex 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 experiments are performed on Presburger arithmetic. Presburger arithmetic is a First-Order theory, whose structure consists of constant symbols $\{0, 1\}$, a function symbol $\{+\}$, and a predicate symbol$\{=\}$. Its interpretation is fixed on the domain $\mathbb{N}$ with the standard addition and the equality of numbers. For each atomic formula in Presburger arithmetic, we construct an automaton as described in [4]. For example, 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 $\overline{\Sigma}$. For instance, $x_0 = 9$, $x_1 = 4$ and $x_2 = 5$ is one of the solutions of ϕ. We use binary representation of base 2 for natural numbers. One of $\{0, 1\}$ is assigned for each power of $2^0, 2^1, ..., 2^i$ from left to right e.g., 9 is 10010. The assignment $\begin{smallmatrix} x_0 : 9 \\ x_1 : 4 \\ x_2 : 5 \end{smallmatrix}$ is represented by $\begin{smallmatrix} 1 \\ 0 \\ 1 \end{smallmatrix}, \begin{smallmatrix} 0 \\ 0 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 0 \\ 1 \\ 1 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 0 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 0 \\ 0 \\ 0 \end{smallmatrix}$ as the string in the $\overline{\Sigma}^*$. We can confirm that the automaton $\mathcal{R}$ accepts the string.

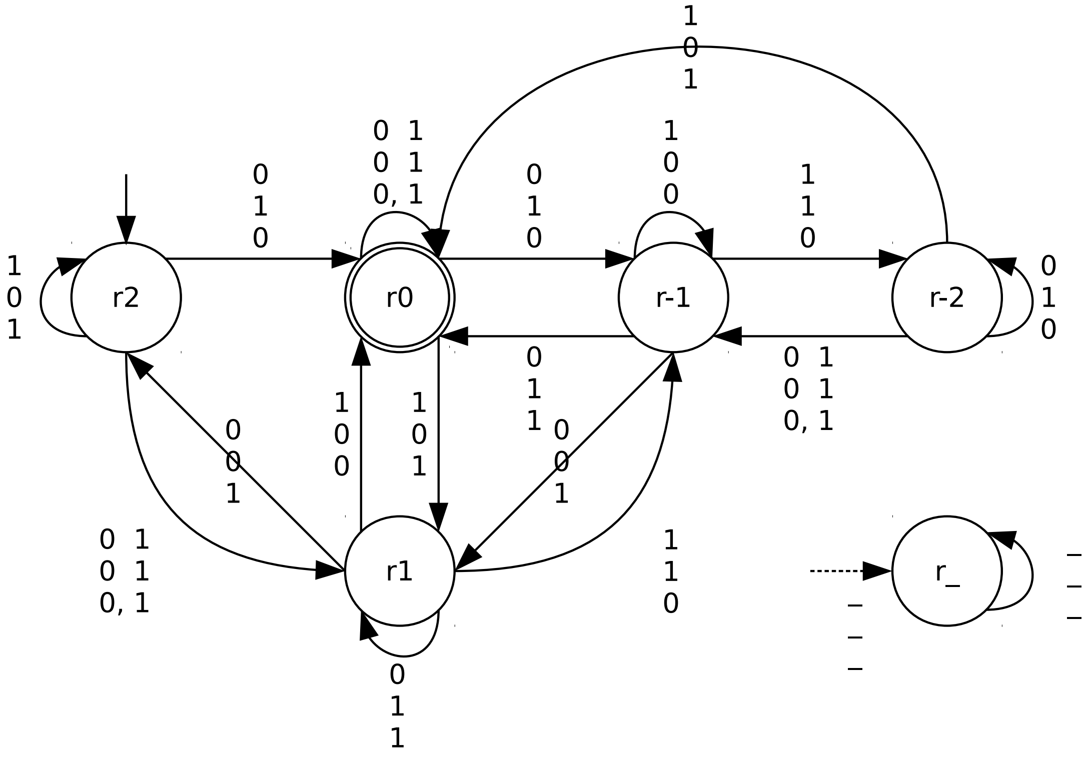


Figure 2.1: The automaton $\mathcal{R}$ which recognizes the solutions of $x_0 + 2x_1 - 3x_2 = 2$

Chapter 3

Conventional Antichain Algorithm

In this section, we recall and reformulate the antichain algorithms for the universality and the inclusion problems of FAs [13]. Antichain algorithms reduce both of the problems to an emptiness problem. Thus we first explain the emptiness problem and the on-the-fly emptiness checking algorithm in detail. We introduce deduction rules of the emptiness checking for later generalization of antichain algorithms.

3.1 Emptiness Checking

The emptiness problem is given an input automaton $\mathcal{A}$, answer whether its language is empty, i.e., $L(\mathcal{A}) = \varnothing$. Let the function $succ_{\mathcal{A}} : 2^{Q_{\mathcal{A}}} \to 2^{Q_{\mathcal{A}}}$ be $succ_{\mathcal{A}}(s) := I_{\mathcal{A}} \cup Post_{\mathcal{A}}(s)$.

Definition 3.1.1. The on-the-fly algorithm of *Emptiness Checking* computes $\mu X.\ (succ_{\mathcal{A}}(X))$. If $\mu X.\ (succ_{\mathcal{A}}(X)) \cap F_{\mathcal{A}} = \varnothing$, then return "empty"; "non-empty", otherwise. Since $succ_{\mathcal{A}}$ is finitary, the least fixpoint $\mu X.\ (succ_{\mathcal{A}}(X))$ exists.

We show that the computation of the fixpoint actually solves the emptiness problem.

Lemma 3.1.1. *Let $x \in \Sigma^*$ and let 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 on 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 3.1.2. $\mu X.\ (succ_{\mathcal{A}}(X)) \cap F_{\mathcal{A}} = \varnothing \Leftrightarrow L(\mathcal{A}) = \varnothing$.

Proof. We only show the $\Rightarrow$ direction. We prove 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 3.1.1, 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 $\mathcal{A}$ is universal. An antichain algorithm performs on-the-fly determinization. Note that subsets of $Q_{\mathcal{A}}$ become the new state. The antichain algorithm minimizes the state sets with exploiting an ordering. Subsets of $Q_{\mathcal{A}}$ are ordered with the subset relation. 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 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 input NFAs $\mathcal{A}$ 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 $L(A) \cap \overline{L(B)} = \varnothing$. We first conduct complementation of the automaton $\mathcal{B}$ to obtain $\mathcal{B}'$, and then perform product construction of $\mathcal{A}$ and $\mathcal{B}'$. Let $\mathcal{C}$ denote the resulting automaton. A state of $\mathcal{C}$ has a form of a tuple: the left element is a state in $Q_{\mathcal{A}}$ and the right one is a subset of $Q_{\mathcal{B}}$. The following ordering over tuples are used to minimize the search space: $(=, \subseteq) := \{((q, S), (p, T)) \mid q = p \wedge S \subseteq T\}$. The antichain algorithm of *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 observe that those procedures construct reachable states, and that for each construction they check certain properties of the state. In *Emptiness Checking*, the procedure visits the reachable states and checks the state is accepting or not. If an accepting state is found, then the procedure terminates with the answer "non-empty". In *Universality Checking*, on the other hand, performs the subset construction. Then the procedure checks whether none of the member in the set of states is accepting. If such a state set is found, then the procedure outputs "non-universal". Based on the observation above, we extract the common part of these algorithms, namely the construction of the reachable states and property checking at each extension of transition steps. We prepare the basic definitions in the deduction style. Later in the paper, we extend the algorithm to make more general and efficient by replacing rules with new ones.

Each time we collect the set of states S of a given automaton $\mathcal{M}$, we check the property of the members in S. The property varies among problems. We define the predicate $SAT, UNSAT$ for any subset S of $Q_{\mathcal{M}}$ to denote the property.

Definition 3.3.1. Let $\mathcal{M}$ be a NFA. $S \subseteq Q_{\mathcal{M}}$.

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

We evaluate the predicate $SAT(S)$ to true if some element of S is also a member of $F_{\mathcal{M}}$. Note that given S, $SAT(S)$ iff $\neg UNSAT(S)$. In the context of *Emptiness Checking*, if we have a set of reachable states T, then $SAT(T)$ allows us to conclude the language is "non-empty" and to abort the search. Let us introduce a set of rules as a method to derive these predicates.

Definition 3.3.2. The *OTF* is the deduction system consisting of the following rules.

$$\frac{}{UNSAT(succ^1(\varnothing))}\ I \cap F = \varnothing \qquad \frac{}{SAT(succ^1(\varnothing))}\ I \cap F \neq \varnothing$$

$$\frac{UNSAT(succ^n(\varnothing))}{UNSAT(succ^{n+1}(\varnothing))}\ Post(succ^n(\varnothing)) \cap F = \varnothing$$

$$\frac{UNSAT(succ^n(\varnothing))}{SAT(succ^{n+1}(\varnothing))}\ Post(succ^n(\varnothing)) \cap F \neq \varnothing$$

$$\frac{UNSAT(succ^n(\varnothing))}{UNSAT(\mu X.\ (succ(X)))}\ succ^n(\varnothing) = succ^{n+1}(\varnothing) \qquad \frac{SAT(succ^n(\varnothing))}{SAT(\mu X.\ (succ(X)))}$$

Definition 3.3.3. Given a NFA $\mathcal{M}$, a *derivation* for $\mathcal{M}$ is a sequence of rules $r_0, r_1, ..., r_n$ such that for each r_i the side condition of r_i instantiated to $\mathcal{M}$ is true.

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

Theorem 3.3.1. *Let* $r_0, r_1, ..., r_n$ *be a derivation for* $\mathcal{M}$.

$$r_n \text{ deduces } UNSAT(\mu X.\ (succ(X))) \Leftrightarrow L(\mathcal{M}) = \varnothing.$$

Proof. We only show $\Rightarrow$ direction. Suppose we have such a derivation $r_0, r_1, ..., r_n$. First, we have 3 observations:

1. $UNSAT(\mu X.\ (succ(X)))$ is deduced only when the condition $succ^n(\varnothing) = succ^{n+1}(\varnothing)$ is met for some n.

2. Since *succ* is defined in terms of *Post*, there exists such n because of the property of *Post*.

3. The deduction rules correspond to the definition of *succ*. Apparently they compute the n-th iteration of $succ_{\mathcal{M}}(\varnothing)$ once instantiated.

From the above observation, we have that $UNSAT(\mu X.\ (succ(X)))$ is deduced when $\mu X.\ (succ_{\mathcal{M}}(X)) \cap F_{\mathcal{M}} = \varnothing$. By Lemma 3.1.2, we conclude that $L(\mathcal{M}) = \varnothing$. □

Chapter 4

Generalized Antichain Algorithm for Composition Term

Similar to the *Emptiness Checking*, the forward antichain algorithms *Universality Checking* and *Inclusion Checking* try to construct the set of reachable states. The differences of the forward antichain algorithms from the *Emptiness Checking* are as follows:

1. States are obtained via the closure operations (the complementation for Universality problem, and the complementation and intersection for Inclusion problem).

2. Each time the antichain algorithm collects the state set, it minimizes the set with respect to a certain ordering and keeps the search space minimal.

The idea is that the transition from a larger state (w.r.t. the specific ordering) is always simulated by those from the smaller state. Thus we can safely minimize the search space, eliminating the larger ones. In this section, we introduce an inductive definition of the regular operations and the ordering. Recall that the regular language is closed under the complementation, the union, and the intersection. We further add the projection. We denote the product automaton of $\mathcal{A}$ and $\mathcal{B}$ by $\mathcal{A} \otimes \mathcal{B}$, and the sum automaton by $\mathcal{A} \oplus \mathcal{B}$. We decompose the complementation operation into the determinization $\mathcal{A}.d$ and the alternation of the final states $\mathcal{A}.c$. Projection to the i-th element of an input is denoted by $\mathcal{A}.p_i$. For a monadic atomic predicate that has a regular set of supports, we refer it as an atomic automaton $\mathcal{A}_0$.

Definition 4.0.1. Composition term ::= $\mathcal{A}_0 \mid \mathcal{A}.d \mid \mathcal{A}.c \mid \mathcal{A}.p_i \mid \mathcal{A} \oplus \mathcal{B} \mid \mathcal{A} \otimes \mathcal{B}$

Since the correspondence of composition terms to FAs is straightforward, we take over the same notion as FAs $\mathcal{M}, \mathcal{A}, \mathcal{B}, ..$ to denote composition terms. Each symbol in a logical formula is translated into a corresponding operator in a composition term. An automaton operation for negation is not direct. When an automaton $\mathcal{A}$ representing φ is non-deterministic, we first apply the determinization d before applying c to express $\neg\varphi$. We say a composition term is *well formed* if every occurrence of 'c' is associated to a deterministic automaton.

Definition 4.0.2. [2] Let $\mathcal{M}$ be a composition term. A set of positions of $\mathcal{M}$ is a set of strings over the alphabet $\{1, 2\}$. A set of position of $\mathcal{M}$ is denoted $\mathcal{P}os(\mathcal{M}) \subseteq \{1, 2\}^*$ and defined as follows:

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

Definition 4.0.3. A subterm of $\mathcal{M}$ at position p , denoted by $\mathcal{M}|_p$ is:

$$\begin{aligned} \mathcal{M}|_\epsilon &:= \mathcal{M} && \\ \mathcal{A}.u|_{1p'} &:= \mathcal{A}|_{p'} && u \in \{d, c, p_i\} \\ (\mathcal{A}_1\ b\ \mathcal{A}_2)|_{ip'} &:= \mathcal{A}_i|_{p'} && b \in \{\otimes, \oplus\} \end{aligned}$$

The subterm relation over composition terms $\mathcal{A}, \mathcal{B}$, denoted by $\mathcal{A} \leq \mathcal{B}$, is defined by $\exists p \in \mathcal{P}os(\mathcal{B}).\ \mathcal{B}|_p = \mathcal{A}$.

4.1 Interpretation of composition terms

We design composition terms to express the each closure operation. Each component of a FA (a transition function, a set of initial and final states) is defined inductively along the structure of the composition terms.

Definition 4.1.1.

$$\begin{array}{ll} & \Delta : Q_{\mathcal{M}} \times c \to 2^{Q_{\mathcal{M}}} \\ \text{Case } \mathcal{M} \equiv \mathcal{A}_0 & \Delta_{\mathcal{A}_0} := \Delta_{\mathcal{A}_0} \\ \text{Case } \mathcal{M} \equiv \mathcal{A} \otimes \mathcal{B} & \Delta_{\mathcal{A}\otimes\mathcal{B}}((q_l, q_r), c) := \bigcup_{q'_l \in \Delta_{\mathcal{A}}(q_l, c)} \bigcup_{q'_r \in \Delta_{\mathcal{B}}(q_r, c)} \{(q'_l, q'_r)\} \\ \text{Case } \mathcal{M} \equiv \mathcal{A} \oplus \mathcal{B} & \Delta_{\mathcal{A}\oplus\mathcal{B}} := \Delta_{\mathcal{A}} \cup \Delta_{\mathcal{B}} \\ \text{Case } \mathcal{M} \equiv \mathcal{A}.d & \Delta_{\mathcal{A}.d}(s, c) := \left\{ \bigcup_{q \in s} \Delta_{\mathcal{A}}(q, c) \right\} \\ \text{Case } \mathcal{M} \equiv \mathcal{A}.p_i & \Delta_{\mathcal{A}.p_i}(q, \overline{a}) := \bigcup_{c \in \Sigma} \Delta_{\mathcal{A}}(q, \pi_i(\overline{a}, c)) \\ \text{Case } \mathcal{M} \equiv \mathcal{A}.c & \Delta_{\mathcal{A}.c} := \Delta_{\mathcal{A}} \end{array}$$

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

Definition 4.1.2.

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

4.2 Ordering on states

The composition term give rise to states with a complex structure, e.g., $S \in Q_{(\mathcal{A}\otimes\mathcal{B}).d}$ consists of a states in $Q_{\mathcal{A}\otimes\mathcal{B}}$, whereas a member of $Q_{\mathcal{A}\otimes\mathcal{B}}$ is a tuple whose left element is a state in $Q_{\mathcal{A}}$ and whose right element is a state in $Q_{\mathcal{B}}$. In the previous section, we see that the key part of antichain algorithms is minimization with respect to the certain orderings. We introduce an inductive definition of ordering on states for the composition terms.

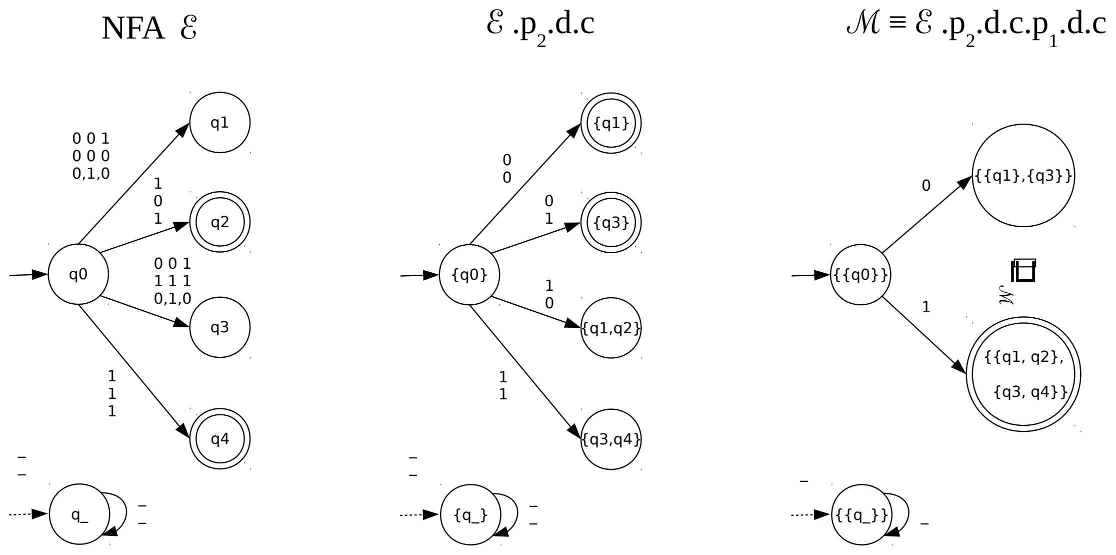


Figure 4.1: The NFA $\mathcal{E}$ and composition term $\mathcal{E}.p_2.d.c.p_1.d.c$

Definition 4.2.1.

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

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

The satisfiability checking algorithm consists of two steps.

1. The automata construction described by a composition term t.

2. The emptiness checking `EC` described by 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 $(\texttt{EC}(t))_{OTF}$ and are performed step-by-step on the length of words. The 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,

- the conversion of the compositional term to a minimally quantified form(preserving language equivalence)

- the distributive laws of the emptiness checking (preserving equisatisfiability)

are presented in Chapter 6.

Definition 4.3.1.

$$\frac{}{UNSAT((succ \circ min_{\sqsubseteq})^1(\varnothing))}\ I \cap F = \varnothing \qquad \frac{}{SAT((succ \circ min_{\sqsubseteq})^1(\varnothing))}\ I \cap F \neq \varnothing$$

$$\frac{UNSAT((succ \circ min_{\sqsubseteq})^n(\varnothing))}{UNSAT((succ \circ min_{\sqsubseteq})^{n+1}(\varnothing))}\ Post((succ \circ min_{\sqsubseteq})^n(\varnothing)) \cap F = \varnothing$$

$$\frac{UNSAT((succ \circ min_{\sqsubseteq})^n(\varnothing))}{SAT((succ \circ min_{\sqsubseteq})^{n+1}(\varnothing))}\ Post((succ \circ min_{\sqsubseteq})^n(\varnothing)) \cap F \neq \varnothing$$

$$\frac{UNSAT((succ \circ min_{\sqsubseteq})^n(\varnothing))}{UNSAT(\mu X.\ (succ \circ min_{\sqsubseteq}(X)))}\ (succ \circ min_{\sqsubseteq})^n(\varnothing) = (succ \circ min_{\sqsubseteq})^{n+1}(\varnothing)$$

$$\frac{SAT((succ \circ min_{\sqsubseteq})^n(\varnothing))}{SAT(\mu X.\ (succ \circ min_{\sqsubseteq}(X)))}$$

Example 4.3.1. 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 which corresponds to ψ is $(\mathcal{R}.p_2.p_0.d.c \otimes \mathcal{G}.p_0)$. Let $\mathcal{E}$ denote the term. The initial state of $\mathcal{E}$ is computed as follows:

$$\begin{aligned} I_{\mathcal{E}} &= I_{\mathcal{R}.p_2.p_0.d.c \otimes \mathcal{G}.p_0} = \{I_{\mathcal{R}.p_2.p_0.d.c} \times I_{\mathcal{G}.p_0}\} = \\ &\dots \\ &= \{\{I_{\mathcal{R}}\} \times I_{\mathcal{G}}\} = \{(\{r_2\}, g_1)\} \end{aligned}$$

For instance,

$$\begin{aligned} \Delta_{\mathcal{E}}\left((\{r_2\}, g_1), \begin{smallmatrix}0\\1\\0\end{smallmatrix}\right) &= \{\Delta_{\mathcal{R}.p_2.p_0.d.c}\left(\{r_2\}, \begin{smallmatrix}0\\1\\0\end{smallmatrix}\right) \times \Delta_{\mathcal{G}.p_0}\left((g_1), \begin{smallmatrix}0\\1\\0\end{smallmatrix}\right)\} = \\ &\dots \\ &= \{(\{r_0, r_1, r\_\}, g_0), (\{r_0, r_1, r\_\}, g_{-1})\} \end{aligned}$$

$Post_{\mathcal{E}}, succ_{\mathcal{E}}$ are computed based on the $\Delta_{\mathcal{E}}$. We can also find the ordering between the states of $\mathcal{E}$. According to the definition of $\sqsubseteq_{\mathcal{E}}$, the ordering is the subset relation for the left hand side of the tuple and the equality for the right hand side.

The figure 3 depicts the difference between $\mathcal{E} \rightarrow_{\texttt{EC}}$ `Empty` and $\mathcal{E} \rightarrow_{\texttt{EC}_{\sqsubseteq}}$ `Empty`. The initial state $(\{r_2\}, g_1)$ is located in the left of the picture. $\mathcal{E} \rightarrow_{\texttt{EC}}$ `Empty` computes all reachable states within i step. As i proceeds, the new states are added to the right direction in the picture. This algorithm generates 31 states until reaching the fixpoint. On the other hand, the generalized antichain algorithm only maintains minimal states w.r.t. $\sqsubseteq_{\mathcal{E}}$. The generated states are circled in the yellow line in the picture. The antichain algorithm constructs just 15 states until reaching the fixpoint.

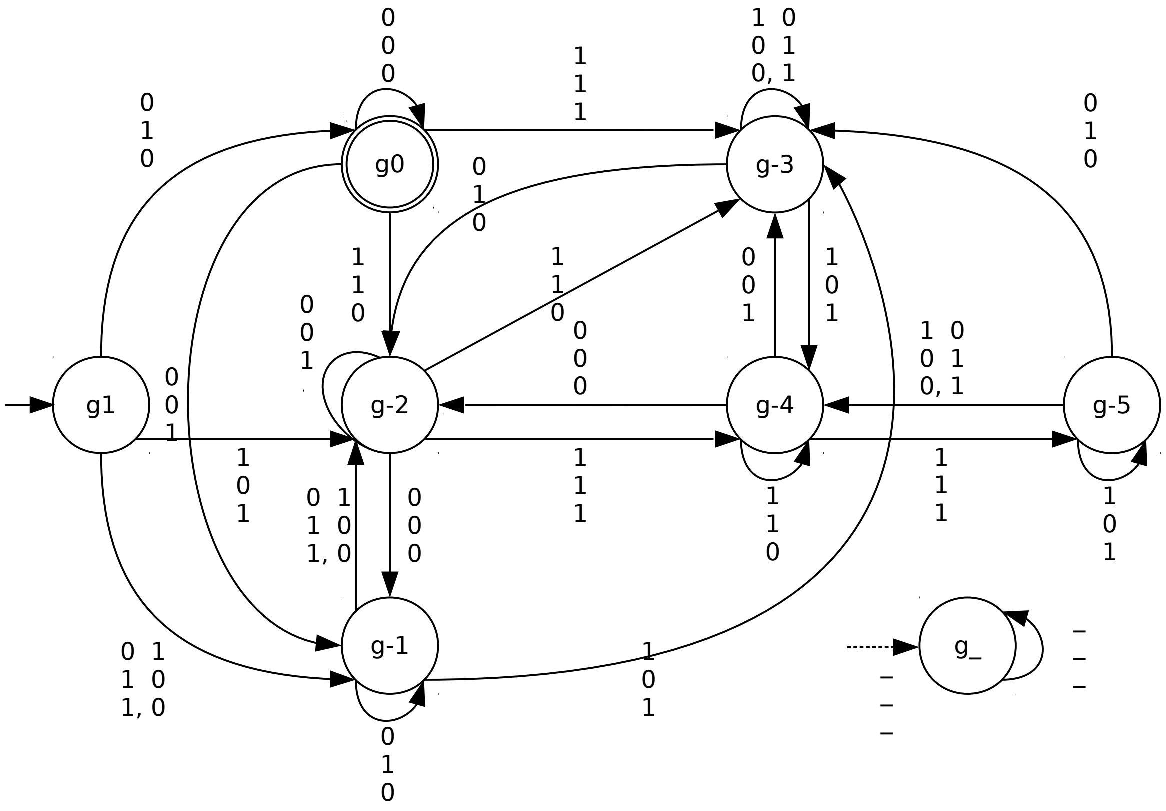


Figure 4.2: The automaton $\mathcal{G}$ which recognizes the solutions of $3x_0 + x_1 + 2x_2 = 1$

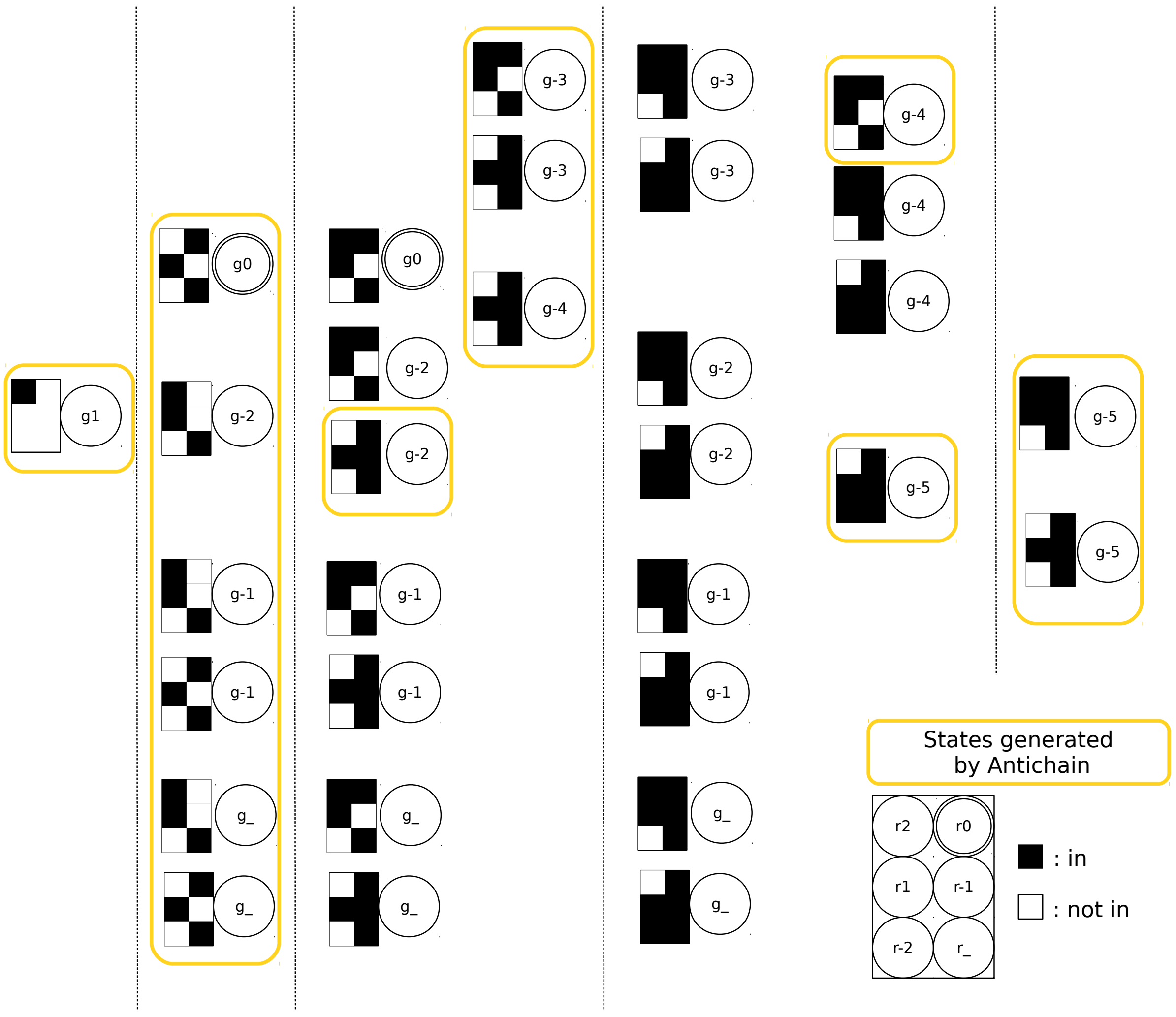


Figure 4.3: Comparison between the on-the-fly state construction and the antichain algorithm

Chapter 5

Composition Terms to Solve Problems

In this section, we prove that the generalized antichain algorithm is complete and sound. For the universality and the 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 the orderings following to the structure of composition terms.

5.1 Universality problem

Lemma 5.1.1. *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 definition 4.1.1, $U \in \left\{\bigcup_{p\in S} \Delta_{\mathcal{A}}(p, c)\right\}$. 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 5.1.2. *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.

$$\forall U \in \hat{\Delta}_{\mathcal{A}.d}(\{S\}, x'), V \in \hat{\Delta}_{\mathcal{A}.d}(\{T\}, x').\ S \subseteq T \Rightarrow U \subseteq V.$$

$$\text{We also have } \hat{\Delta}_{\mathcal{A}.d}(\{S\}, cx') = \bigcup_{S'\in\hat{\Delta}_{\mathcal{A}.d}(\{S\},x')} \Delta_{\mathcal{A}.d}(S', c)$$

$$\text{Similarly we have } \hat{\Delta}_{\mathcal{A}.d}(\{T\}, cx') = \bigcup_{T'\in\hat{\Delta}_{\mathcal{A}.d}(\{S\},x')} \Delta_{\mathcal{A}.d}(T', c)$$

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 5.1.1, $\forall U \in \Delta_{\mathcal{A}.d}(S', c)\ \forall V \in \Delta_{\mathcal{A}.d}(T', c)$. $U \subseteq V$. 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 5.1.3. $\forall U, V \in Q_{\mathcal{A}.d.c}.\ U \subseteq V \Rightarrow (U \in F_{\mathcal{A}.d.c} \Leftarrow V \in F_{\mathcal{A}.d.c})$

Proof. Suppose $V \in F_{\mathcal{A}.d.c}$. By definition 4.0.1 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 5.1.4. $\forall U \in \hat{\Delta}_{\mathcal{A}.d.c}(\{S\}, x), V \in \hat{\Delta}_{\mathcal{A}.d.c}(\{T\}, x).\ S \subseteq T \Rightarrow (U \in F_{\mathcal{A}.d.c} \Leftarrow V \in F_{\mathcal{A}.d.c})$

Proof. Note that $\hat{\Delta}_{\mathcal{A}.d} = \hat{\Delta}_{\mathcal{A}.d.c}$. From Lemma 5.1.2, $S \subseteq T \Rightarrow U \subseteq V$. From Lemma 5.1.3, $U \in F_{\mathcal{A}.d.c} \Leftarrow V \in F_{\mathcal{A}.d.c}$ □

5.2 Inclusion problem

Lemma 5.2.1. *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 5.1.1, $u_r \subseteq v_r$. 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 5.2.2. *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 5.2.1, 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 5.2.3.

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

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 5.1.3, $u_l \in F_{\mathcal{A}} \wedge u_r \in F_{\mathcal{B}.d.c}$. We conclude $(u_l, u_r) \in F_{\mathcal{A}.d.c}$. □

Theorem 5.2.4. $\forall (v_l, v_r) \in \hat{\Delta}_{\mathcal{M}}(\{(t_l, t_r)\}, x).\ \exists (u_l, u_r) \in \hat{\Delta}_{\mathcal{M}}(\{(s_l, s_r)\}, x).\ s_l = t_l \wedge s_r \subseteq t_r \Rightarrow ((u_l, u_r) \in F_{\mathcal{A} \otimes \mathcal{B}.d.c} \Leftarrow (v_r, v_l) \in F_{\mathcal{A} \otimes \mathcal{B}.d.c})$.

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 5.2.2, 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 5.2.3, we have $(u_l, u_r) \in F_{\mathcal{A} \otimes \mathcal{B}.d.c}$. □

5.3 Generalized antichain algorithm for emptiness checking

Lemma 5.3.1. $\forall q, p \in Q_{\mathcal{M}}. q \sqsubseteq_{\mathcal{M}} p \Rightarrow (q \in F_{\mathcal{M}} \Leftarrow p \in F_{\mathcal{M}})$.

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

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

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 5.3.3. *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 5.3.2 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 5.3.2 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 $q'' \in \mu X.\ (min_{\mathcal{M}}(succ_{\mathcal{M}}(X)))$. $\square$

Lemma 5.3.4. $\mu X.\ (succ_{\mathcal{M}} \circ min_{\mathcal{M}}(X)) \cap F_{\mathcal{M}} = \varnothing \Leftrightarrow L(\mathcal{M}) = \varnothing$.

Proof. We only show the $\Rightarrow$ direction. We show by contradiction. We suppose $\mu X.\ (succ_{\mathcal{M}} \circ min_{\mathcal{M}}(X)) \cap F_{\mathcal{M}} = \varnothing$ and assume $L(\mathcal{M}) \neq \varnothing$. Then we have $x \in L(\mathcal{M})$, i.e., we have an accepting state q such that $q \in \hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x) \cap F_{\mathcal{M}}$. We have 2 cases based on the minimality of q in $\hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x)$.

Case. *q is minimal with respect to $\sqsubseteq_{\mathcal{M}}$ in $\hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x)$.*

From Lemma 5.3.3, we have q' in $\mu X.\ (succ_{\mathcal{M}} \circ min_{\mathcal{M}}(X))$ such that $q' \sqsubseteq_{\mathcal{M}} q$ holds. Since $q \in F_{\mathcal{M}}$, from Lemma 5.3.1, we have $q' \in F_{\mathcal{M}}$. $\mu X.\ (succ_{\mathcal{M}} \circ min_{\mathcal{M}}(X))$ contains an accepting state. This is a contradiction.

Case. *q is not minimal.*

We can find another element q' that is minimal w.r.t. $\sqsubseteq_{\mathcal{M}}$ in $\hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x)$, and that $q' \sqsubseteq_{\mathcal{M}} q$ holds. By Lemma 5.3.1, we have $q' \in F_{\mathcal{M}}$. The rest of the discussion is similar to that of the above case. Since q' is the minimal element in $\hat{\Delta}_{\mathcal{M}}(I_{\mathcal{M}}, x)$, by Lemma 5.3.3, we have q'' in $\mu X.\ (succ_{\mathcal{M}} \circ min_{\mathcal{M}}(X))$ such that $q'' \sqsubseteq_{\mathcal{M}} q'$ holds. By Lemma 5.3.1, q'' is accepting, and therefore yields the contradiction. $\square$

In the same way as Theorem 3.3.1, we have the following statement.

Theorem 5.3.5. *Let $r_0, r_1, ..., r_n$ be a derivation for $\mathcal{M}$.*

$$r_n \text{ deduces } UNSAT(\mu X.\ (succ \circ min_{\sqsubseteq}(X))) \Leftrightarrow L(\mathcal{M}) = \varnothing.$$

Chapter 6

Conversion Rules

Before constructing an automaton for a given input formula, there are chances to decompose a problem into smaller sub-problems. It is natural to apply the distributive laws of $\wedge, \vee$, and $\neg$. Those conversions preserve the logical equivalence of the formula. On the other hand, since it is enough 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 the 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, its 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 6.1.1. Logically equivalent rules correspond to distributive law of negation, conjunction and disjunction and quantifiers.

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

In addition, we can derive the rule from VR, $(\mathcal{A} \otimes \mathcal{B}).dc.p_i.dc \overset{VR}{\longrightarrow}{}^{*} \mathcal{A}.dc.p_i.dc \otimes \mathcal{B}.dc.p_i.dc$. The normal form of a composition term $\mathcal{M}$ is called a minimally quantified form. We confirm that the minimally quantified form always exists for any composition terms by Proposition 6.1.1.

Proposition 6.1.1. *VR is terminating.*

Proof. $\xrightarrow{VR}$ is terminating by AC-RPO with the precedence $d, c, p_i, \succ \otimes \succ \oplus$. □

Lemma 6.1.1. *Let $\mathcal{M}$ be a composition term and $\mathcal{M}'$ be a normal form of VR, i.e., $\mathcal{M} \xrightarrow{VR}{}^* \mathcal{M}'$.*

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

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

Lemma 6.1.2. $\forall p, p' \in \mathcal{P}os(\mathcal{M}').\ p \leq p' \wedge \mathcal{M}'|_{p'} = (\_ \oplus \_) \Rightarrow \mathcal{M}'|_p = (\_ \oplus \_)$.

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 $\mathcal{M}'|_{p'} = (\_ \oplus \_) \Rightarrow \mathcal{M}'|_p = (\_ \oplus \_)$.

Inductive step.

$n = k + 1$, we have I.H. that $\mathcal{M}'|_{pi_1i_2\ldots i_k} = (\_ \oplus \_) \Rightarrow \mathcal{M}'|_p = (\_ \oplus \_)$. We suppose $\mathcal{M}'|_{pi_1i_2\ldots i_ki_{k+1}} = (\_ \oplus \_)$. Then from Lemma 6.1.1, $\mathcal{M}'|_{pi_1i_2\ldots i_k} = (\_ \oplus \_)$. By applying I.H., we have $\mathcal{M}'|_p = (\_ \oplus \_)$. □

6.2 Equisatisfiable conversions

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

Definition 6.2.1.

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

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

Lemma 6.2.1. $L(\mathcal{A}) = \varnothing \Leftrightarrow L(\mathcal{A}.d) = \varnothing$.

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

Lemma 6.2.2. $L(\mathcal{A}) = \varnothing \Leftrightarrow L(\mathcal{A}.p_i) = \varnothing$.

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,

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

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

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

Example 6.2.1. Let us demonstrate the conversion rules. We take over the first-order formula ψ from the previous chapter. For clarity, we replace the atomic predicates with R, G and B.

$$\psi := \exists x_0.\ \neg\exists x_1.\ R(x_0, x_1, x_2) \wedge \exists x_2.\ (G(x_0, x_1, x_2) \vee B(x_0, x_1, x_2))$$

First, we translate ψ into a composition term $(\mathcal{R}.p_1.d.c \otimes (\mathcal{G}\underline{\oplus}\mathcal{B}).\underline{p_2}).p_0$. In the conversion, we highlighten the redexes by the underline.

$$\begin{aligned}
&(\mathcal{R}.p_1.d.c \otimes (\mathcal{G}\underline{\oplus}\mathcal{B}).\underline{p_2}).p_0\\
\xrightarrow{VR}&(\mathcal{R}.p_1.d.c\underline{\otimes}(\mathcal{G}.p_2\underline{\oplus}\mathcal{B}.p_2)).p_0\\
\xrightarrow{VR}&((\mathcal{R}.p_1.d.c \otimes \mathcal{G}.p_2)\underline{\oplus}(\mathcal{R}.p_1.d.c \otimes \mathcal{B}.p_2)).\underline{p_0}\\
\xrightarrow{VR}&(\mathcal{R}.p_1.d.c \otimes \mathcal{G}.p_2).p_0 \oplus (\mathcal{R}.p_1.d.c \otimes \mathcal{B}.p_2).p_0
\end{aligned}$$

Further the converted composition term is decomposed into smaller problems. Note that the derivation is bottom-up. We put the composition term at the bottom of the derivation and we put the applicable rules on it.

$$CR_3\ \dfrac{CR_2\ \dfrac{\mathcal{R}.p_1.d.c \otimes \mathcal{G}.p_2 \to_{\texttt{EC}} bool}{(\mathcal{R}.p_1.d.c \otimes \mathcal{G}.p_2).\underline{p_0} \to_{\texttt{EC}} bool} \qquad CR_2\ \dfrac{\mathcal{R}.p_1.d.c \otimes \mathcal{B}.p_2 \to_{\texttt{EC}} bool}{(\mathcal{R}.p_1.d.c \otimes \mathcal{B}.p_2).\underline{p_0} \to_{\texttt{EC}} bool}}{(\mathcal{R}.p_1.d.c \otimes \mathcal{G}.p_2).p_0\underline{\oplus}(\mathcal{R}.p_1.d.c \otimes \mathcal{B}.p_2).p_0 \to_{\texttt{EC}} bool}$$

As a result of the conversion, we have smaller 2 problems $\mathcal{R}.p_1.d.c \otimes \mathcal{G}.p_2$ and $\mathcal{R}.p_1.d.c \otimes \mathcal{B}.p_2$.

Chapter 7

Implementation and Experiments

In this section, we explain the implementation and describe the experimental results. The tool takes a First-Order Presburger formula and answers whether there exist satisfiable assignments for the free variables in the formula. We do not restrict the formula to be closed. The syntactical height of the formula represents repetition of the closure operations on the automata, while the number of variables represents the size of Σ. Since we have 4 approaches, we compare the running time of those 4 cases.

7.1 Implementation

We implemented those algorithms in OCaml. A user specifies one of 4 algorithms;

- On-the-fly construction without any other techniques (`OTF`)
- Term conversion only (`Ct`)
- Generalized antichain algorithm only (`Ac`)
- 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)))
```

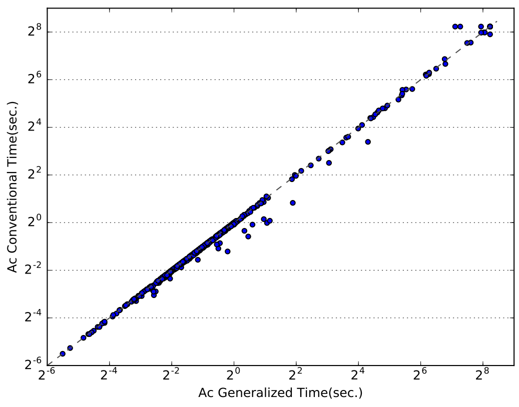
The code size is approximately 1300 lines. Each state of an automaton is indexed by a natural number. A set is implemented by the sorted list. Thus $\subseteq$ is computed by single traversal of lists, whereas $\sqsubseteq$, which appears in $\subseteq_{\mathcal{M}.d}$, requires multiple traversals.

7.2 Data set and the experiment method

Our experimental data are First-Order Presburger formulas. The formulas are classified under 2 characteristics:

1. The syntactical height in terms of their abstract syntax tree (We denote H3 for the formula with the height 3.)
2. The number of distinct variables occurring in the formula (We denote V4 for the formula with the 4 kinds of variables.)

We generate Presburger formulas in the following manner. For an atomic formula:



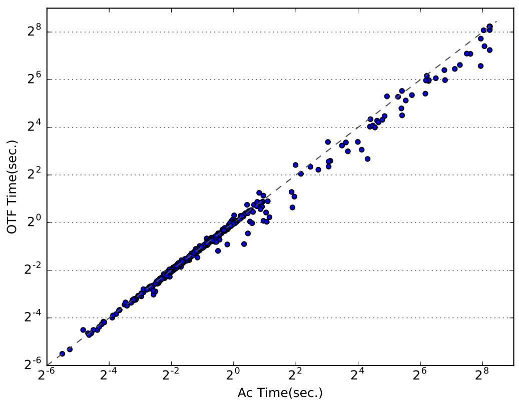


Figure 7.1: Generalized Ac vs Conventional Ac/ Generalized Ac vs OTF, H3V3

1. 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 $0 \leq c \leq 100$.

2. Falsey atomic formulas are excluded.

In order to make a formula out of the set of atomic formulas, we first randomly generate ASTs with the specified height, then fill each node with one of $\neg, \exists$ for unary nodes, and $\wedge, \vee$ for binary nodes. The variable bounded by $\exists$ is also randomly chosen. We prepare 500 problems for each class of the syntactical height 3 or 4, and the 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 result `Empty` or `NonEmpty`. Our experiments are performed on an Intel Xeon-X5690@3.47GHz processor with 15 GiB RAM.

Those 4 algorithms are compared in terms of Success or Timeout as well as the consumed time. The results are shown in Table 7.1. When at least one of the 4 algorithms failed as timeout, the time record of the problem is excluded from "Total Time" shown in the right column of Table 7.1.

We compare our generalized antichain algorithm `Ac` and the conventional antichain algorithm. The conventional antichain algorithm only uses the subset relation as the minimizing ordering. We also compare the generalized antichain algorithm and on-the-fly algorithm `OTF`. The results are shown in Figure 7.1 to Figure 7.4, respectively. In the figures, the generalized antichain algorithm is shown in the x-axis, while the conventional antichain and on-the-fly algorithm is shown in the y-axis. If the point is plotted above the diagonal, we observe that the generalized antichain outperforms the other. Timeout records are plotted at the 300 sec. point in the figure.

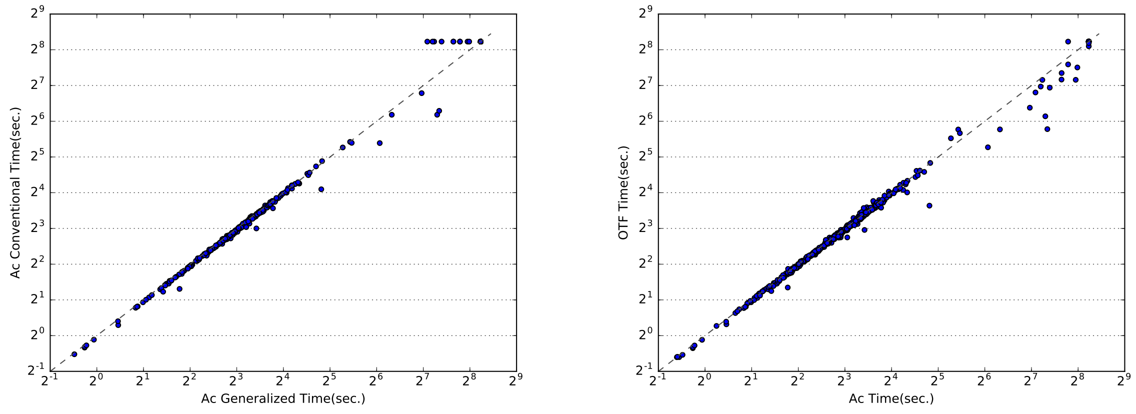


Figure 7.2: Generalized Ac vs Conventional Ac/ Generalized Ac vs OTF, H3V4

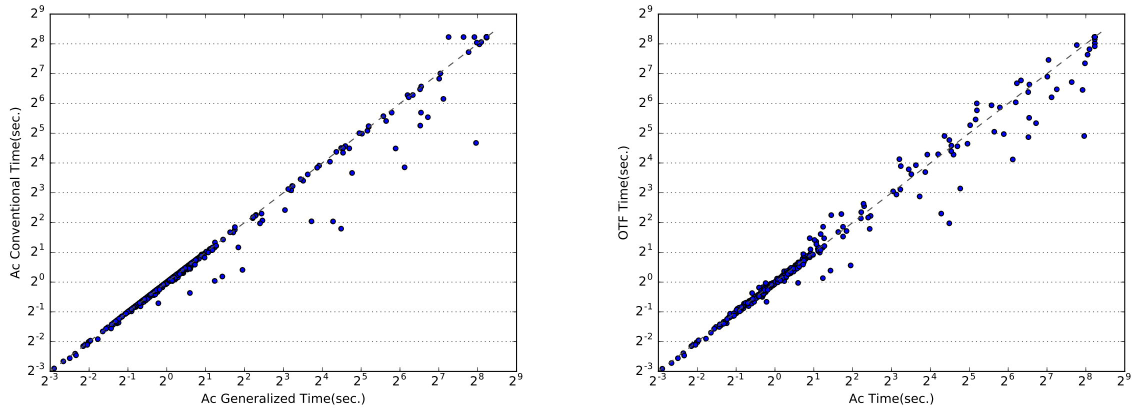


Figure 7.3: Generalized Ac vs Conventional Ac/ Generalized Ac vs OTF, H4V3

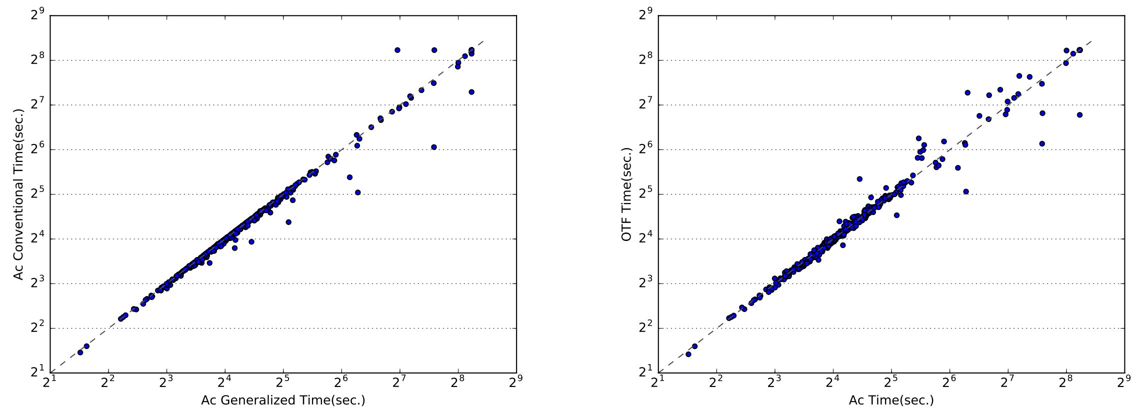


Figure 7.4: Generalized Ac vs Conventional Ac/ Generalized Ac vs OTF, H4V4

Table 7.1: Results in frequency distribution table.

H3V2	Success/ Timeout				Total Time			
Size	OTF	Ct	Ac	CtAc	OTF	Ct	Ac	CtAc
0 -	488/ 0	488/ 0	488/ 0	488/ 0	51.04	51.3	77.32	77.38
1000 -	6/ 0	6/ 0	6/ 0	6/ 0	70.91	72.36	123.37	123.41
2000 -	3/ 0	3/ 0	3/ 0	3/ 0	254.66	256.45	405.6	409.24
3000 -	1/ 0	1/ 0	0/ 1	0/ 1	0	0	0	0
4000 -	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

H3V3	Success/ Timeout				Total Time			
Size	OTF	Ct	Ac	CtAc	OTF	Ct	Ac	CtAc
0 -	438/ 0	438/ 0	438/ 0	438/ 0	242.16	242.98	277.97	277.81
1000 -	27/ 0	27/ 0	27/ 0	27/ 0	585.33	595.97	776.89	782.1
2000 -	8/ 0	8/ 0	8/ 0	8/ 0	628.57	636.62	986.76	996.65
3000 -	8/ 0	8/ 0	6/ 2	6/ 2	715.71	736.69	935.34	946.03
4000 -	2/ 0	2/ 0	1/ 1	1/ 1	269.3	266.09	262.06	263.05
5000 -	0/ 0	0/ 0	0/ 0	0/ 0	0	0	0	0

H3V4	Success/ Timeout				Total Time			
Size	OTF	Ct	Ac	CtAc	OTF	Ct	Ac	CtAc
0 -	422/ 0	422/ 0	422/ 0	422/ 0	3084.79	3087.87	3148.38	3148.04
1000 -	23/ 0	23/ 0	23/ 0	23/ 0	461.22	454.08	604.3	605.52
2000 -	9/ 0	9/ 0	9/ 0	9/ 0	854.78	835.41	1154.07	1149.84
3000 -	5/ 1	5/ 1	5/ 1	5/ 1	642.62	657.05	867.93	848.87
4000 -	1/ 0	1/ 0	1/ 0	1/ 0	142.64	144.8	150.5	151.76
5000 -	1/ 0	1/ 0	0/ 1	0/ 1	0	0	0	0

H4V2	Success/ Timeout				Total Time			
Size	OTF	Ct	Ac	CtAc	OTF	Ct	Ac	CtAc
0 -	376/ 0	376/ 0	376/ 0	376/ 0	119.07	119.47	179.59	179.8
1000 -	33/ 0	33/ 0	33/ 0	33/ 0	732.08	742.84	1248.84	1254.29
2000 -	26/ 0	26/ 0	24/ 2	24/ 2	2025.78	2011.47	3333.26	3358.61
3000 -	9/ 0	9/ 0	4/ 5	4/ 5	734.73	705.19	781.75	796.62
4000 -	1/ 0	1/ 0	1/ 0	1/ 0	297.99	296.31	252.98	271.37
5000 -	0/ 0	0/ 0	0/ 0	0/ 0	0	0	0	0

H4V3	Success/ Timeout				Total Time			
Size	OTF	Ct	Ac	CtAc	OTF	Ct	Ac	CtAc
0 -	328/ 0	328/ 0	328/ 0	328/ 0	348.25	348.34	389.68	386.65
1000 -	32/ 0	32/ 0	32/ 0	32/ 0	417.92	412.44	844.14	838.03
2000 -	17/ 0	17/ 0	17/ 0	17/ 0	966.0	950.76	1304.76	1299.88
3000 -	10/ 0	10/ 0	9/ 1	9/ 1	926.39	892.39	1046.68	1047.38
4000 -	5/ 0	5/ 0	3/ 2	3/ 2	577.3	577.74	565.36	562.13
5000 -	0/ 0	0/ 0	0/ 0	0/ 0	0	0	0	0

H4V4	Success/ Timeout				Total Time			
Size	OTF	Ct	Ac	CtAc	OTF	Ct	Ac	CtAc
0 -	302/ 0	302/ 0	302/ 0	302/ 0	4999.64	5009.72	5049.68	5053.81
1000 -	34/ 0	34/ 0	34/ 0	34/ 0	794.41	789.95	832.62	824.57
2000 -	20/ 0	20/ 0	19/ 1	19/ 1	1089.36	1082.96	1116.32	1124.41
3000 -	11/ 0	10/ 1	11/ 0	11/ 0	1123.95	1090.03	1120.65	1132.61
4000 -	7/ 1	7/ 1	7/ 1	7/ 1	783.53	781.5	684.94	681.76
5000 -	5/ 1	5/ 1	5/ 1	5/ 1	954.08	910.19	818.73	821.85

7.3 Experimental results

From Table 7.1, we observe that `Ct` and `OTF` is better than `Ac` or `CtAc`. However, the generalized antichain algorithm `Ac` enjoys the performance improvement for sufficiently large and complex problems (Table 7.1, H4V4, size 4000 -). Due to the overhead of calculating orderings, it does not work for small problems. The threshold would be the formulas with height 4, 4 variables and more than 4000 state size. `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 Table 7.1). We cannot maintain that the generalized antichain algorithm immediately leads to the universal performance improvement.

The same hold for the comparison of the generalized antichain and the conventional antichain algorithm. Through Figure 7.1 to 7.4, most of the points are plotted below the diagonals. That is, the conventional one is better. However, there are more cases where the generalized antichain succeeds while the conventional timeouts than the cases where the generalized timeouts while the conventional succeeds. Except H4V4, the generalized covers the problems that the conventional successfully solves within the timeout. As previously observed, the generalized antichain algorithm could work when the problem structure is complex.

In spite of the efforts to minimize the search space smaller and smaller, we could not observe the performance improvements from the results. Additional experiments are required: the present data set is not sufficiently large nor complex.

Chapter 8

Conclusion

We have presented a generalized approach to solve the problems in the automata theoretic theorem proving. We developed the generalized antichain algorithm expanding the forward antichain algorithms for *Universality* and *Inclusion* problems. One of our aim was to directly handle a nested formula with an antichain algorithm (i.e., to avoid flattening the input formula). The aim was achieved by introducing the composition terms that represent the automata construction and by finding ordering inductively. So that the generalized antichain algorithm is inductively defined for the structure of composition terms. As an optimization, we further introduced

1. the conversion rules of composition terms which preserve the accepted language, and
2. the distributive laws of emptiness checking into the composition terms.

We performed experiments on randomly generated 3000 Presburger formulas. We could not observe that the generalized antichain algorithm improved the performance. Due to the overhead of calculating orderings, it did not work for small problems. In the most cases, conversion of the composition term performed better than the generalized antichain algorithm. Still, there were some cases where the generalized antichain algorithm outperformed other algorithms, when the problem was sufficiently large.

Acknowledgements I thank Prof. Mizuhito Ogawa for the careful support and lively discussion. I am excited to embody his ideas and insights in the proof and implementation. I would like to thank Prof. Nao Hirokawa and Prof. Tachio Terauchi for their valuable comments. Special thanks go to Prof. Aart Middeldorp for the precise comments and suggestion for a pitfall in the proof, and to Ms. Franziska Rapp for helping me revising the thesis. In addition, I owe so much to the members in Ogawa, Hirokawa and Terauchi laboratories for their comments and supports.

Bibliography

[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.