A Finite Exact Representation of Register Automata Configurations

Yu-Fang Chen
Academia Sinica, Taiwan

Bow-Yaw Wang
Academia Sinica, Taiwan

Di-De Yen
Academia Sinica, Taiwan

A register automaton is a finite automaton with finitely many registers ranging from an infinite alphabet. Since the valuations of registers are infinite, there are infinitely many configurations. We describe a technique to classify infinite register automata configurations into finitely many *exact* representative configurations. Using the finitary representation, we give an algorithm solving the reachability problem for register automata. We moreover define a computation tree logic for register automata and solve its model checking problem.

1 Introduction

Register automata are generalizations of finite automata to process strings over infinite alphabets [9]. In addition to a finite set of states, a register automaton has finitely many registers ranging from an infinite alphabet. When a register automaton reads a data symbol with parameters from the infinite alphabet, it compares values of registers and parameters and finite constants, updates registers, and moves to a new location. Since register automata allow infinitely many values in registers and parameters, they have been used to model systems with unbounded data. For instance, a formalization of user registration and account management in the XMPP protocol is given in [1, 2]. Since user identifiers are not fixed *a priori*, models in register automata are more realistic for the protocol.

Analyzing register automata nonetheless is not apparent. Since there are infinitely many valuations of registers, the number of configurations for a register automaton is inherently infinite. Moreover, register automata can update a register with values of registers or parameters in data symbols. The special feature makes register automata more similar to programs than to classical automata. Infinite configurations and register updates increase the expressive power of register automata. They also complicate analysis of the formalism as well.

In this paper, we develop a finitary representation for configurations of register automata. As observed in [6], register automata recognize strings modulo automorphisms on the infinite alphabet. That is, a string is accepted by a register automaton if and only if the image of the string under a one-to-one and onto mapping on the infinite alphabet is accepted by the same automaton. Subsequently, two valuations of registers are indistinguishable by register automata if one is the image of the other under an automorphism on the infinite alphabet. We therefore identify indistinguishable valuations and classify valuations into finitely many representative valuations. Naturally, our finitary representation enables effective analysis on register automata.

The first application of representative valuations is reachability analysis. Based on representative valuations, we define representative configurations. Instead of checking whether a given configuration is reachable in a register automaton, it suffices to check whether its representative configuration belongs to the finite set of reachable representative configurations. We give an algorithm to compute successors of an arbitrary representative configuration. The set of reachable representative configurations is obtained by fixed point computation.

L. Clemente and L. Holik (Eds.): 15th International Workshop on Verification of Infinite-State Systems (INFINITY'13)
EPTCS 140, 2014, pp. 16–34, doi:10.4204/EPTCS.140.2

Our second application is model checking on register automata. We define a computation tree logic (CTL) for register automata. Configurations in a representative configuration are shown to be indistinguishable in our variant of computation tree logic. The CTL model checking problem for register automata thus is solved by the standard algorithm with slight modifications.

As an illustration, we model an algorithm for the Byzantine generals problem under an interesting scenario. In the scenario, two loyal generals are trying to reach a consensus at the presence of a treacherous general. They would like to know how many soldiers should be sent to the front line. Since the total number of soldiers is unbounded,[1] we use natural numbers as the infinite alphabet and model the algorithm in a register automaton. By the CTL model checking algorithm, we compute the initial configurations leading to a consensus eventually.

Our formulation of register automata follows those in [1, 2]. It is easy to show that the expressive power of register automata with constant symbols is no difference from those versions without constants. A canonical representation theorem similar to Myhill-Nerode theorem for deterministic register automata is developed in [1]. In [2], a learning algorithm for register automata is proposed. Finite-memory automata is another generalization of finite automata to infinite alphabets [6]. Finite-memory automata and register automata have the same expressive power. In [6], we know that the emptiness problem for finite-memory automata is decidable. Therefore, the reachability problem for register automata is also decidable. In [4], it has been shown that the emptiness for register automata is in *PSPACE*. This is done by reducing an emptiness checking problem for register automata to an emptiness problem of a finite transition system over the so called "abstract states", which is very similar to the "equivalence classes" defined in this paper. However, in their reduction, they did not provide any algorithm to move from one abstract state to another abstract state, which is in fact non-trivial. In contrast, we provide an algorithm in Section 4. It has been shown in [5] that register automata together with a total order over the *alphabet* are equivalent to timed automata. In fact, the register automata model defined in this paper can be easily extended to support arbitrary order among alphabet symbols (the order can be partial) and hence is more general then the one defined in [5]. This because the finite representation of configurations defined in this paper can be extended to describe any finite relations between alphabet symbols by adding more possible values in the matrix. That is, instead of just 0 and 1 used in the current paper, we can add more possible values such as $\leq, <, >, \ldots$ to describe a richer relation between alphabet symbols. A survey on expressive power of various finite automata with infinite alphabets is given in [9]. We model the algorithm for the Byzantine generals problem [8] presented in [10].

The paper is organized as follows. We briefly review register automata in Section 2. Section 3 presents an exact finitary representation for configurations. It is followed by the reachability algorithm for register automata (Section 4). A computation tree logic for register automata and its model checking algorithm are given in Section 5. We discuss the Byzantine generals problem as an example (Section 6). Finally, we conclude the presentation in Section 7.

2 Preliminaries

Let S, S', and S'' be sets. An *automorphism* on S is a one-to-one and onto mapping from S to S. Given a subset T of S, an automorphism σ on S is invariant on T if $\sigma(x) = x$ for every $x \in T$. If f is an onto mapping from S to S' and h is a mappings from S' to S'', $(h \circ f)$ is a mapping from S to S'' that $(h \circ f)(a) = h(f(a))$ for $a \in S$. We write $S_{n\times n}$ for the set of square matrices of size n with entries in S.

[1] This is certainly an ideal simplification. The number of soldiers of course is bounded by the population of the empire.

Let Σ be an infinite *alphabet*. A set of constants, denoted by C, is a finite subset of Σ. Let A be a finite set of *actions*. Each action has a finite *arity*. A *data symbol* $\alpha(\overline{d_n})$ consists of an action $\alpha \in A$ and $\overline{d_n} = d_1 d_2 \cdots d_n \in \Sigma^n$ when α is of arity n. A *string* is a sequence of data symbols.

Fix a finite set X of *registers*. Define $X' = \{x' | x \in X\}$. A *valuation* v is a mapping from X to Σ. Since X is finite, we represent a valuation by a string of $\Sigma^{|X|}$. We write $V_{(X,\Sigma)}$ for the set of valuations from X to Σ.

Let $P = \{p_1, p_2, \ldots\}$ be an infinite set of *formal parameters* and $P_n = \{p_1, p_2, \ldots, p_n\} \subseteq P$. A *parameter valuation* $v_{\overline{d_n}}$ is a mapping from P_n to Σ such that $v_{\overline{d_n}}(p_i) = d_i$ for every $1 \leq i \leq n$. We write $V_{(P,\Sigma)}$ for the set of parameter valuations. Obviously, each finite sequence $\overline{d_n} \in \Sigma^n$ corresponds to a parameter valuation $v_{\overline{d_n}} \in V_{(P,\Sigma)}$.

Given a valuation v, a parameter valuation $v_{\overline{d_n}}$, and $e \in X \cup P_n \cup C$, define

$$[\![e]\!]_{v,v_{\overline{d_n}}} = \begin{cases} v(e) & \text{if } e \in X \\ v_{\overline{d_n}}(e) & \text{if } e \in P_n \\ e & \text{if } e \in C \end{cases}$$

Thus $[\![e]\!]_{v,v_{\overline{d_n}}}$ is the value of e on the valuation v, parameter valuation $v_{\overline{d_n}}$, or constant e.

An *assignment* π is of the form

$$(x_{k_1} x_{k_2} \ldots x_{k_n}) \mapsto (e_{l_1} e_{l_2} \ldots e_{l_n})$$

where $x_{k_i} \in X$, $e_{l_i} \in X \cup P_n \cup C$, and $x_{k_i} \neq x_{k_j}$ whenever $i \neq j$. Let Π denote the set of assignments. For valuation v and parameter valuation $v_{\overline{d_n}}$, define

$$[\![\pi]\!]_{v,v_{\overline{d_n}}} \triangleq \{v' | v'(x_{k_i}) = [\![e_{l_i}]\!]_{v,v_{\overline{d_n}}} \text{ for every } 1 \leq i \leq n\}.$$

That is, $[\![\pi]\!]_{v,v_{\overline{d_n}}}$ contains the valuations obtained by executing the assignment under the valuation v and parameter valuation $v_{\overline{d_n}}$.

An *atomic guard* is of the form $e = f$ or its negation $\neg(e = f)$ (written $e \neq f$) where $e, f \in X \cup P_n \cup C$. A *guard* is a conjunction of atomic guards. We write Γ for the set of guards. For any valuation v and parameter valuation $v_{\overline{d_n}}$, define

$$\begin{array}{ll} v, v_{\overline{d_n}} \models e = f & \text{if } [\![e]\!]_{v,v_{\overline{d_n}}} = [\![f]\!]_{v,v_{\overline{d_n}}} \\ v, v_{\overline{d_n}} \models e \neq f & \text{if } [\![e]\!]_{v,v_{\overline{d_n}}} \neq [\![f]\!]_{v,v_{\overline{d_n}}} \\ v, v_{\overline{d_n}} \models g_1 \wedge g_2 \wedge \cdots \wedge g_k & \text{if } v, v_{\overline{d_n}} \models g_i \text{ for every } 1 \leq i \leq k \end{array}$$

Definition 1. *A* register automaton *is a tuple* $(\Sigma, A, X, L, l_0, \Delta)$ *where*

- *A is a finite set of actions;*
- *L is a finite set of* locations*;*
- *$l_0 \in L$ is the* initial location*;*
- *X is a finite set of registers.*
- *$\Delta \subseteq L \times A \times \Gamma \times \Pi \times L$ is a finite set of* transitions.

A *configuration* $\langle l, v\rangle$ of a register automaton $(\Sigma, A, X, L, l_0, \Delta)$ consists of a location $l \in L$ and a valuation $v \in V_{(X,\Sigma)}$. For configurations $\langle l, v\rangle$ and $\langle l', v'\rangle$, we say $\langle l, v\rangle$ *transits* to $\langle l', v'\rangle$ on $\alpha(\overline{d_n})$ (written $\langle l, v\rangle \xrightarrow{\alpha(\overline{d_n})} \langle l', v'\rangle$) if there is a transition $(l, \alpha, g, \pi, l') \in \Delta$ such that $v, v_{\overline{d_n}} \models g$ and $v' \in [\![\pi]\!]_{v,v_{\overline{d_n}}}$.

A *run* of a register automaton $(\Sigma, A, X, L, l_0, \Delta)$ on a string $\alpha_0(\overline{d^0_{n_0}})\ \alpha_1(\overline{d^1_{n_1}}) \cdots\ \alpha_{k-1}(\overline{d^{k-1}_{n_{k-1}}})$ is a sequence of configurations $\langle l_0, v_0\rangle\ \langle l_1, v_1\rangle \cdots\ \langle l_k, v_k\rangle$ such that $\langle l_i, v_i\rangle \xrightarrow{\alpha(\overline{d^i_{n_i}})} \langle l_{i+1}, v_{i+1}\rangle$ for every $0 \leq i < k$.

Example 1. *Let $\mathbb{N}$ denote the set of natural numbers, $\Sigma = \mathbb{N}$, $A = \{\alpha, \beta\}$, $L = \{l_0, l_1\}$, $C = \{2\}$, and $X = \{x_1, x_2\}$ where α and β have arities 2 and 1 respectively. Consider the register automaton in Figure 1. In the figure, $\dfrac{\alpha|g}{\pi}$ denotes a transition with action α, guard g, and assignment π. Here is a run of the automaton:*

$$\langle l_0, 77\rangle \xrightarrow{\alpha(1,3)} \langle l_1, 13\rangle \xrightarrow{\beta(1)} \langle l_1, 13\rangle \xrightarrow{\beta(2)} \langle l_1, 23\rangle \xrightarrow{\beta(1)} \langle l_0, 69\rangle$$

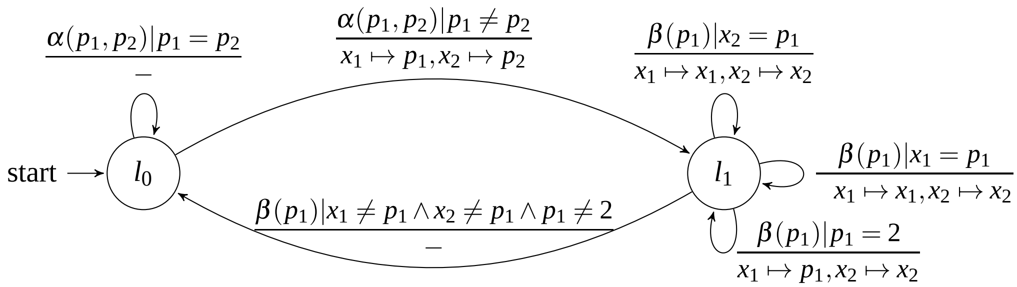


Figure 1: A Register Automaton

Let $(\Sigma, A, X, L, l_0, \Delta)$ be a register automaton. A configuration $\langle l, v\rangle$ is *reachable* if there is a run $\langle l_0, v_0\rangle\langle l_1, v_1\rangle \cdots \langle l_k, v_k\rangle$ of $(\Sigma, A, X, L, l_0, \Delta)$ with $\langle l_k, v_k\rangle = \langle l, v\rangle$. The *reachability* problem for register automata is to decide whether a given configuration is reachable in a given register automaton.

Definition 2 ([7])**.** *An* equality logic formula *is defined as follows.*

$$\begin{array}{rcl} \phi & : & \phi \wedge \phi \mid \neg\phi \mid \phi \implies \phi \mid var = var \\ var & : & x \mid x' \mid p \mid c \end{array}$$

where $x \in X$, $x' \in X'$, $p \in P$, and $c \in C$.

Note that a guard is also an equality logic formula. An equality logic formula ϕ is *valid* if ϕ always evaluates to true by assigning each member of $X \cup X' \cup P$ with an arbitrary element in Σ. We write $\vdash \phi$ when ϕ is valid. The formula ϕ is *consistent* if it is not the case that $\vdash \neg\phi$. Given an equality logic formula ϕ, the *validity* problem for equality logic is to decide whether $\vdash \phi$.

Theorem 1 ([7])**.** *The validity problem for equality logic is coNP-complete.*

3 Representative Configurations

Consider a register automaton $(\Sigma, A, X, L, l_0, \Delta)$. Since Σ is infinite, there are an infinite number of valuations in $V_{(X,\Sigma)}$. A register automaton subsequently has infinitely many configurations. In this section, we show that configurations can be partitioned into finitely many classes. Any two configurations in the same class are indistinguishable by register automata.

Definition 3. *Let $u, v \in V_{(X,\Sigma)}$. u is* equivalent *to v with respect to C (written $u \sim_C v$) if there is an automorphism σ on Σ such that σ is invariant on C and $(\sigma \circ u)(x) = v(x)$ for every $x \in X$.*

For example, let $\Sigma = \mathbb{N}$, $X = \{x_1, x_2, x_3\}$, $C = \{1\}$, $v_1 = 123$, $v_2 = 134$, and $v_3 = 523$. We have $v_1 \sim_C v_2$ but $v_1 \not\sim_C v_3$.

It is easy to see that $\sim_C$ is an equivalence relation on $V_{(X,\Sigma)}$. For any valuation $v \in V_{(X,\Sigma)}$, we write $[v]$ for the equivalence class of v. That is,

$$[v] \triangleq \{u \in V_{(X,\Sigma)} | u \sim_C v\}.$$

The equivalence class $[v]$ is called a *representative valuation*. Note that there are only finitely many representative valuations for X is finite.

Definition 4. *A* representative configuration $\langle l, [v] \rangle$ *is a pair where $l \in L$ and $[v]$ is a representative valuation.*

Since X and L are finite sets, the number of representative configurations is finite. Our next task is to show that every configurations in a representative configuration behave similarly. Let $\langle l, [v] \rangle$ and $\langle l', [v'] \rangle$ be two representative configurations. Define $\langle l, [v] \rangle \rightsquigarrow \langle l', [v'] \rangle$ if

- for each $u \in [v]$, there is a valuation $u' \in [v']$ and a data symbol $\alpha(\bar{d_n})$ such that $\langle l, u \rangle \xrightarrow{\alpha(\bar{d_n})} \langle l', u' \rangle$; and
- for each $u' \in [v']$, there is a valuation $u \in [v]$ and a data symbol $\alpha(\bar{d_n})$ such that $\langle l, u \rangle \xrightarrow{\alpha(\bar{d_n})} \langle l', u' \rangle$.

Let $\langle \Sigma, A, X, L, l_0, \Delta \rangle$ be a register automaton and $\langle l_k, [v_k] \rangle$ a representative configuration. We say $\langle l_k, [v_k] \rangle$ is *reachable* if there is a sequence of representative configurations $\langle l_0, [v_0] \rangle\ \langle l_1, [v_1] \rangle \cdots \langle l_k, [v_k] \rangle$ such that $\langle l_i, [v_i] \rangle \rightsquigarrow \langle l_{i+1}, [v_{i+1}] \rangle$ for every $0 \leq i < k$. The following three propositions are useful to our key lemma.

Proposition 1. *Let $v \in V_{(X,\Sigma)}$ be a valuation, $v_{\bar{d_n}} \in V_{(P,\Sigma)}$ a parameter valuation, and $g \in \Gamma$ a guard. $v, v_{\bar{d_n}} \models g$ if and only if $\sigma \circ v, \sigma \circ v_{\bar{d_n}} \models g$ for every automorphism σ on Σ which is invariant on C.*

Proposition 2. *Let $v, w \in V_{(X,\Sigma)}$ be valuations, $v_{\bar{d_n}} \in V_{(P,\Sigma)}$ a parameter valuation, and $\pi \in \Pi$ an assignment. $w \in [\![\pi]\!]_{v, v_{\bar{d_n}}}$ if and only if $\sigma \circ w \in [\![\pi]\!]_{\sigma \circ v, \sigma \circ v_{\bar{d_n}}}$ for every automorphism σ on Σ which is invariant on C.*

Proposition 3. *Let $(\Sigma, A, X, L, l_0, \Delta)$ be a register automaton, $l, l' \in L$ locations, $v, v' \in V_{(X,\Sigma)}$ valuations, and $\alpha(\bar{d_n})$ a data symbol with $\bar{d_n} = d_1 d_2 \cdots d_n$. If $\langle l, v \rangle \xrightarrow{\alpha(\bar{d_n})} \langle l', v' \rangle$, then $\langle l, \sigma \circ v \rangle \xrightarrow{\alpha(\sigma(\bar{d_n}))} \langle l', \sigma \circ v' \rangle$ for every automorphism σ on Σ which is invariant on C, where $\sigma(\bar{d_n}) \triangleq \sigma(d_1)\sigma(d_2)\cdots\sigma(d_n)$.*

By Proposition 3, we get the following key lemma.

Lemma 1. *Let $(\Sigma, A, X, L, l_0, \Delta)$ be a register automaton, $l, l' \in L$, and $v, v' \in V_{(X,\Sigma)}$. $\langle l, v \rangle \xrightarrow{\alpha(\bar{d_n})} \langle l', v' \rangle$ for some $\alpha(\bar{d_n})$ if and only if $\langle l, [v] \rangle \rightsquigarrow \langle l', [v'] \rangle$.*

Lemma 1 shows that representative configurations are exact representations for configurations with respect to transitions. The configuration $\langle l, v \rangle$ transits to another configuration $\langle l', v' \rangle$ in one step precisely when their representative configurations have a transition. There are however infinitely many valuations. In order to enumerate $[v]$ effectively, we use a matrix-based representation.

Let $[v]$ be a representative valuation with $v \in V_{(X,\Sigma)}$. Assume $\{\bar{0}, \bar{1}\} \cap \Sigma = \emptyset$. A *representative matrix* $R_{[v]} \in (\{\bar{0}, \bar{1}\} \cup C)_{|X| \times |X|}$ of $[v]$ is defined as follows.

$$(R_{[v]})_{ij} \quad \triangleq \quad \begin{cases} v(x_i) & \text{if } v(x_i) = v(x_j) \in C \\ \bar{1} & \text{if } v(x_i) = v(x_j) \notin C \\ \bar{0} & \text{otherwise} \end{cases}$$

Let $v \in V_{(X,\Sigma)}$ be a valuation. The entry $(R_{[v]})_{ij}$ denotes the equality relation among registers x_i, x_j, and constant c for every $c \in C$. If $v(x_i) = v(x_j)$, $(R_{[v]})_{ij} \in \{\bar{1}\} \cup C$; otherwise, $(R_{[v]})_{ij} = \bar{0}$; moreover, if $v(x_i) = c \in C$, $(R_{[v]})_{ii} = c$. The following proposition shows that $R_{[v]}$ is well-defined.

Proposition 4. *For any $u, v \in V_{(X,\Sigma)}$, $[u] = [v]$ if and only if $R_{[u]} = R_{[v]}$.*

By Proposition 4, we will also call $R_{[v]}$ a representative valuation and write $R_{[v]}$ for $[v]$. Subsequently, $\langle l, R_{[v]} \rangle \rightsquigarrow \langle l', R_{[v']} \rangle$ if and only if $\langle l, [v] \rangle \rightsquigarrow \langle l', [v'] \rangle$.

Example 2. *By example 1, we have $v_0 = 77$, $v_1 = v_2 = 13$, $v_3 = 23$, $v_4 = 69$ and $R_{[v_0]} = \begin{pmatrix} \bar{1} & \bar{1} \\ 1 & 1 \end{pmatrix}, R_{[v_1]} = R_{[v_2]} = R_{[v_4]} = \begin{pmatrix} \bar{1} & \bar{0} \\ 0 & 1 \end{pmatrix}, R_{[v_3]} = \begin{pmatrix} 2 & \bar{0} \\ 0 & 1 \end{pmatrix}$. Hence, $\langle l, R_{[v_0]} \rangle \rightsquigarrow \langle l', R_{[v_1]} \rangle \rightsquigarrow \langle l', R_{[v_2]} \rangle \rightsquigarrow \langle l', R_{[v_3]} \rangle \rightsquigarrow \langle l', R_{[v_4]} \rangle$.*

Every representative valuation corresponds to a matrix. However, not every matrix has a corresponding representative valuation. For instance, the zero matrix $(\bar{0}) \in \{\bar{0}, \bar{1}\}_{1\times 1}$ does not correspond to any representative valuation. If $(\bar{0}) = R_{[v]}$ for some valuation v, one would have the absurdity $v(x_1) \neq v(x_1)$. Such matrices are certainly not of our interests and should be excluded.

For any $R \in (\{\bar{0}, \bar{1}\} \cup C)_{|X|\times|X|}$, define the equality logic formula $E(R)$ as follows.

$$E(R) \stackrel{\triangle}{=} \bigwedge_{R_{ij} \in C} (x_i = x_j \wedge x_i = R_{ij}) \wedge \bigwedge_{R_{ij} = \bar{1}} (x_i = x_j \wedge \bigwedge_{c \in C} x_i \neq c) \wedge \bigwedge_{R_{ij} = \bar{0}} x_i \neq x_j$$

Idea: If we do not add the equalities of form $x_i = c \in C$ for some i or the inequalities $x_i \neq c \in C$ for some i to the conjunction $E(R)$, we can not distinguish the following four kinds of matrices:
(1) $\begin{pmatrix} c & \bar{1} \\ \bar{1} & c \end{pmatrix}$ (2) $\begin{pmatrix} c & \bar{1} \\ c & c \end{pmatrix}$ (3) $\begin{pmatrix} c & c \\ \bar{1} & c \end{pmatrix}$ (4) $\begin{pmatrix} c & c \\ c & c \end{pmatrix}$
The fourth kind of matrix is the matrix we hope for.

We say the matrix R is *consistent* if $E(R)$ is consistent. It can be shown that a consistent matrix is also a representative matrix. Indeed, Algorithm 1 computes a valuation v such that $R_{[v]} = R$ for any consistent matrix R.

Algorithm 1 starts from a valuation where the register x_i is assigned to R_{ii} for every $R_{ii} \in C$, the rest of registers are assigned to distinct elements in $\Sigma \setminus C$. It goes through entries of the given consistent matrix R by rows. At row i, the algorithm assigns $w(x_i)$ to the register x_j if $R_{ij} \in \{\bar{1}\} \cup C$. Hence the first i rows of R are equal to the first i rows of $R_{[w]}$ after iteration i. When Algorithm 1 returns, we obtain a valuation whose representative matrix is R.

Lemma 2. *Let $R \in (\{\bar{0}, \bar{1}\} \cup C)_{|X|\times|X|}$ be a consistent matrix and $w = \mathsf{CanonicalVal}(R)$. $R = R_{[w]}$.*

For a consistent matrix R, the valuation computed by $\mathsf{CanoncalVal}(R)$ is called the *canonical valuation* of R. The following lemma follows from Lemma 2.

Lemma 3. *Let $R \in (\{\bar{0}, \bar{1}\} \cup C)_{|X|\times|X|}$. R is consistent if and only if $R = R_{[v]}$ for some $v \in V_{(X,\Sigma)}$.*

By Lemma 3, it is now straightforward to enumerate all representative matrices. Algorithm 2 computes the set of all representative matrices.

4 Reachability

Let $(\Sigma, A, X, L, l_0, \Delta)$ be a register automaton and $\langle l, v \rangle$ a configuration with $l \in L$ and $v \in V_{(X,\Sigma)}$. In order to solve the reachability problem for register automata, we show how to compute all $\langle l', R_{[v']} \rangle$ such that $\langle l, R_{[v]} \rangle \rightsquigarrow \langle l', R_{[v']} \rangle$.

```
// c_1, c_2, ..., c_|X| are distinct elements in Σ \ C
Input: R : a consistent matrix
Output: w ∈ V_(X,Σ) : R = R_[w]
foreach 1 ≤ i ≤ |X| do
    if R_ii ∈ C then
        w(x_i) ← R_ii;
    else
        w(x_i) ← c_i;
    end
end
foreach i = 1 to |X| − 1 do
    foreach j = i + 1 to |X| do
        if R_ij ∈ {1̄} ∪ C then w(x_j) ← w(x_i);
    end
end
return w;
```

Algorithm 1: $\mathsf{CanonicalVal}(R)$

```
Output: ℛ : ℛ = {R_[v] : v ∈ V_(X,Σ)}
ℛ ← ∅;
foreach matrix R ∈ ({0̄, 1̄} ∪ C)_{|X|×|X|} do
    if R is consistent then ℛ ← ℛ ∪ {R};
end
return ℛ;
```

Algorithm 2: $\mathsf{UniverseR}(X)$

By Lemma 1, $\langle l, R_{[v]}\rangle \rightsquigarrow \langle l', R_{[v']}\rangle$ if $\langle l, v\rangle \xrightarrow{\alpha(\bar{d_n})} \langle l', v'\rangle$ for some $\alpha(\bar{d_n})$. A first attempt to find $\langle l', R_{[v']}\rangle$ with $\langle l, R_{[v]}\rangle \rightsquigarrow \langle l', R_{[v']}\rangle$ is to compute all $\langle l', v'\rangle$ with $\langle l, v\rangle \xrightarrow{\alpha(\bar{d_n})} \langle l', v'\rangle$ for some $\alpha(\bar{d_n})$. The intuition however would not work. Since Σ is infinite, there can be infinitely many data symbols $\alpha(\bar{d_n})$ and valuations v' with $\langle l, v\rangle \xrightarrow{\alpha(\bar{d_n})} \langle l', v'\rangle$. It is impossible to enumerate them.

Instead, we compute $\langle l', R'\rangle$ with $\langle l, R_{[v]}\rangle \rightsquigarrow \langle l', R'\rangle$ directly. Based on equality relations among registers in the given configuration $\langle l, v\rangle$, we infer equality relations among registers in a configuration $\langle l', v'\rangle$ with $\langle l, v\rangle \xrightarrow{\alpha(\bar{d_n})} \langle l', v'\rangle$. Since there are finitely many representative matrices, we enumerate those representative matrices conforming to the inferred equality relations among registers. The conforming representative matrices give desired representative configurations.

We start with extracting equality relations among registers in the given configuration $\langle l, v\rangle$. For any valuation $v \in V_{(X,\Sigma)}$, define

$$E(v) \stackrel{\triangle}{=} \bigwedge_{v(x)=c\in C} x = c \wedge \bigwedge_{v(x)=v(y)} x = y \wedge \bigwedge_{v(x)\neq v(y)} x \neq y, \text{ and}$$

$$E'(v) \stackrel{\triangle}{=} \bigwedge_{v(x)=c\in C} x' = c \wedge \bigwedge_{v(x)=v(y)} x' = y' \wedge \bigwedge_{v(x)\neq v(y)} x' \neq y'.$$

Let (l, α, g, π, l') be a transition and $\langle l, v\rangle \xrightarrow{\alpha(\bar{d_n})} \langle l', v'\rangle$. Equality relations among registers in $\langle l', v'\rangle$ are determined by the assignment π. Let $\pi = (x_{k_1} x_{k_2} \ldots x_{k_n}) \mapsto (e_{l_1} e_{l_2} \ldots e_{l_n})$. Define

$$E(\pi) \stackrel{\triangle}{=} \bigwedge_{i=1}^{n} x'_{k_i} = e_{l_i}.$$

Observe that $E(v)$ and $E(\pi)$ are equality logic formulae for any valuation v and assignment π. By Lemma 3, $\langle l, R\rangle$ is a representative configuration when R is a consistent matrix. For any representative configuration $\langle l, R\rangle$, we characterize a representative configuration $\langle l', R'\rangle$ with $\langle l, R\rangle \rightsquigarrow \langle l', R'\rangle$ as follows.

Definition 5. *Let $RA = (\Sigma, A, X, L, l_0, \Delta)$ be a register automaton, $(l,\ \alpha,\ g,\ \pi,\ l') \in \Delta$ a transition, and R a consistent matrix. Define the set $Post_{RA}(\langle l, R\rangle)$ of representative matrices as follows. $\langle l', R'\rangle \in Post_{RA}(\langle l, R\rangle)$ if $g \wedge E(w) \wedge E(\pi) \wedge E'(w')$ is consistent, where w and w' are the canonical valuations of R and R' respectively.*

Example 3. *Let $\Sigma = \mathbb{N}$, $X = \{x_1, x_2, x_3\}$, and $R = \begin{pmatrix} \overline{1} & \overline{0} & \overline{0} \\ \overline{0} & \overline{1} & \overline{1} \\ \overline{0} & \overline{1} & \overline{1} \end{pmatrix}$. By Algorithm 1, $w = 122$ is the canonical valuation of R. Consider a transition (l, α, g, π, l') where g is $(x_1 \neq x_2) \wedge (p_1 \neq p_2)$ and π is $(x_1 x_2 x_3) \mapsto (x_2 p_1 p_2)$. Then $E(v)$ is $(x_1 \neq x_2) \wedge (x_1 \neq x_3) \wedge (x_2 = x_3)$ and $E(\pi)$ is $(x'_1 = x_2) \wedge (x'_2 = p_1) \wedge (x'_3 = p_2)$. Let F denote the equality logic formula $g \wedge E(v) \wedge E(\pi)$. F is consistent. Observe that $\vdash F \implies x'_2 \neq x'_3$. Consider the following three cases:*

1. *R'_0 is $\begin{pmatrix} \overline{1} & \overline{1} & \overline{0} \\ \overline{1} & \overline{1} & \overline{0} \\ \overline{0} & \overline{0} & \overline{1} \end{pmatrix}$. Since $\vdash F \implies x'_1 = x'_2 \wedge x'_1 \neq x'_3$, $\langle l', R'_0\rangle \in Post_{RA}(\langle l, R\rangle)$;*
2. *R'_1 is $\begin{pmatrix} \overline{1} & \overline{0} & \overline{1} \\ \overline{0} & \overline{1} & \overline{0} \\ \overline{1} & \overline{0} & \overline{1} \end{pmatrix}$. Since $\vdash F \implies x'_1 = x'_3 \wedge x'_1 \neq x'_2$, $\langle l', R'_1\rangle \in Post_{RA}(\langle l, R\rangle)$;*

3. R'_2 *is* $\begin{pmatrix} \overline{1} & \overline{0} & \overline{0} \\ \overline{0} & \overline{1} & \overline{0} \\ 0 & 0 & 1 \end{pmatrix}$. *Since* $\vdash F \implies x'_1 \neq x'_2 \wedge x'_1 \neq x'_3$, $\langle l', R'_2 \rangle \in Post_{RA}(\langle l, R \rangle)$.

Lemma 4. *$E(v)$ is consistent for every $v \in V_{(X,\Sigma)}$. Moreover, $E(v) = E(w)$ if $[v] = [w]$.*

Lemma 5. *Let $RA = (\Sigma, A, X, L, l_0, \Delta)$ be a register automaton, R and R' be consistent. $\langle l, R \rangle \rightsquigarrow \langle l', R' \rangle$ iff there is $(l, \alpha, g, \pi, l') \in \Delta$ and $g \wedge E(w) \wedge E(\pi) \wedge E'(w')$ is consistent, where w and w' are the canonical valuations of R and R' respectively.*

The following lemma is directly from Lemma 5. It shows that Definition 5 correctly characterizes successors of any given representative configuration.

Lemma 6. *$Post_{RA}(\langle l, R \rangle) = \{\langle l', R' \rangle | \langle l, R \rangle \rightsquigarrow \langle l', R' \rangle\}$.*

Using Algorithm 2 to enumerate representative matrices, it is straightforward to compute the set $Post_{RA}(\langle l, R \rangle)$ for any representative configuration $\langle l, R \rangle$ (Algorithm 3). We first obtain the canonical valuation w for R. The algorithm iterates through transitions of the given register automaton. For a transition (l, α, g, π, l'), define the equality logic formula F to be $g \wedge E(w) \wedge E(\pi)$. The algorithm then checks if F is consistent. If so, it goes through every representative matrices and adds them to the successor set U by Lemma 6.

Input: *RA*: $(\Sigma, A, X, L, l_0, \Delta)$; $\langle l, R \rangle$: a representative configuration
$\mathscr{R} \leftarrow \mathsf{UniverseR}(X)$;
$U, w \leftarrow \emptyset, \mathsf{CanonicalVal}(R)$;
foreach $(l, \alpha, g, \pi, l') \in \Delta$ **do**
  $F \leftarrow g \wedge E(w) \wedge E(\pi)$;
  if F *is consistent* **then**
    foreach $R' \in \mathscr{R}$ **do**
      $w' \leftarrow \mathsf{CanonicalVal}(R')$;
      $F' \leftarrow g \wedge E(w) \wedge E(\pi) \wedge E'(w')$;
      if F' *is consistent* **then** $U \leftarrow U \cup \{R'\}$;
    end
end
return U;

Algorithm 3: $\mathsf{Post}(RA, \langle l, R \rangle)$

Theorem 2. *Let $RA = (\Sigma, A, X, L, l_0, \Delta)$ be a register automaton and $\langle l, R \rangle$ a representative configuration. $R' \in Post_{RA}(\langle l, R \rangle)$ iff $R' \in \mathsf{Post}(RA, \langle l, R \rangle)$.*

With the algorithm $\mathsf{Post}(RA, \langle l, R \rangle)$ at hand, we are ready to present our solution to the reachability problem for register automata. By Lemma 1, $\langle l_0, v_0 \rangle \langle l_1, v_1 \rangle \cdots \langle l_k, v_k \rangle$ is a run precisely when $\langle l_0, R_{[v_0]} \rangle \rightsquigarrow \langle l_1, R_{[v_1]} \rangle \rightsquigarrow \cdots \rightsquigarrow \langle l_k, R_{[v_k]} \rangle$. In order to check if the configuration $\langle l, v \rangle$ is reachable, we compute reachable representative configurations and check if the $\langle l, R_{[v]} \rangle$ belongs to the reachable representative configurations (Algorithm 4).

Our first technical result is summarized in the following theorem.

Theorem 3. *Let $(\Sigma, A, X, L, l_0, \Delta)$ be a register automaton and $\langle l, v \rangle$ a configuration. $\langle l, v \rangle$ is reachable iff $\mathsf{Reach}((\Sigma, A, X, L, l_0, \Delta), (l, R_{[v]}))$ returns true.*

Input: $(\Sigma, A, X, L, l_0, \Delta)$: a register automaton; $\langle l, R\rangle$: a representative configuration
Output: *true* if $\langle l, R\rangle$ is reachable; *false* otherwise
$\mathscr{R} \leftarrow \mathsf{UniverseR}(X)$;
$U, V \leftarrow \{\langle l_0, R_0\rangle | R_0 \in \mathscr{R}\}, \emptyset$;
while $U \neq V$ **do**
  $U' \leftarrow \bigcup_{\langle l,R\rangle \in U} \mathsf{Post}(RA, \langle l, R\rangle)$;
  $V, U \leftarrow U, U \cup U'$;
end
result $\leftarrow$ **if** $\langle l, R\rangle \in U$ **then** *true* **else** *false*;
return *result*;

Algorithm 4: $\mathsf{Reach}((\Sigma, A, X, L, l_0, \Delta), \langle l, R\rangle)$

5 $CTL(X,L)$ Model Checking

In addition to checking whether a configuration is reachable, it is often desirable to check patterns of configurations in runs of a register automaton. We define a computation tree logic to specify patterns of configurations in register automata. Representative configurations are then used to design an algorithm that solves the model checking problem for register automata.

Let X be the set of registers and L the set of locations. An *atomic formula* is an equality over X, an equality one side over X another side over C, or a location $l \in L$. We write AP for the set of atomic formulae. Consider the computation tree logic $CTL(X,L)$ defined as follows [3].

- If $f \in AP$, f is a $CTL(X,L)$ formula;
- If f_0 and f_1 are $CTL(X,L)$ formulae, $\neg f_0$ and $f_0 \wedge f_1$ are $CTL(X,L)$ formulae;
- If f_0 and f_1 are $CTL(X,L)$ formulae, $EX\, f_0$, $E(f_0 U f_1)$, and $EG f_0$ are $CTL(X,L)$ formulae.

We use the standard abbreviations: $\mathit{false}(\equiv \neg(x = x))$, $\mathit{true}(\equiv \neg \mathit{false})$, $f_0 \vee f_1(\equiv \neg(\neg f_0 \wedge \neg f_1))$, $f_0 \implies f_1(\equiv \neg f_0 \vee f_1)$, $AX\, f_0(\equiv \neg EX \neg f_0)$, $EF\, f_0(\equiv E(\mathit{true}\, U\, f_0))$, $AG\, f_0(\equiv \neg EF \neg f_0)$, and $AF\, f_0(\equiv \neg EG \neg f_0)$. Examples of $CTL(X,L)$ are $AF(l_{end} \wedge x_1 = x_2)$, $AG((l_{start} \wedge \neg(x_1 = x_2)) \implies EF(l_{end} \wedge (x_1 = x_2)))$.

Let $\langle l, v\rangle$ be a configuration of a register automaton $RA = (\Sigma, A, X, L, l_0, \Delta)$ and f a $CTL(X,L)$ formula. Define $\langle l, v\rangle$ *satisfies* f *in RA* ($\langle l, v\rangle \models_{RA} f$) by

- $\langle l, v\rangle \models_{RA} l$;
- $\langle l, v\rangle \models_{RA} x = y$ if $v(x) = v(y)$;
- $\langle l, v\rangle \models_{RA} \neg f$ if not $\langle l, v\rangle \models_{RA} f$;
- $\langle l, v\rangle \models_{RA} f_0 \wedge f_1$ if $\langle l, v\rangle \models_{RA} f_0$ and $\langle l, v\rangle \models_{RA} f_1$;
- $\langle l, v\rangle \models_{RA} EX f$ if $\langle l', v'\rangle \models_{RA} f$ for some $\alpha(\bar{d_n})$ such that $\langle l, v\rangle \xrightarrow{\alpha(\bar{d_n})} \langle l', v'\rangle$;
- $\langle l, v\rangle \models_{RA} E(f_0 U f_1)$ if there are $k \geq 0$, $\alpha_i(\bar{d^i_{n_i}})$, $\langle l_i, v_i\rangle$ with $\langle l_0, v_0\rangle = \langle l, v\rangle$, and $\langle l_i, v_i\rangle \xrightarrow{\alpha_i(\bar{d^i_{n_i}})} \langle l_{i+1}, v_{i+1}\rangle$ for every $0 \leq i < k$ such that (1) $\langle l_k, v_k\rangle \models_{RA} f_1$; and (2) $\langle l_i, v_i\rangle \models_{RA} f_0$ for every $0 \leq i < k$.
- $\langle l, v\rangle \models_{RA} EG f$ if there are $\alpha_i(\bar{d^i_{n_i}})$, $\langle l_i, v_i\rangle$ with $\langle l_0, v_0\rangle = \langle l, v\rangle$, and $\langle l_i, v_i\rangle \xrightarrow{\alpha_i(\bar{d^i_{n_i}})} \langle l_{i+1}, v_{i+1}\rangle$ for every $i \geq 0$ such that $\langle l_i, v_i\rangle \models_{RA} f$.

Let $RA = (\Sigma, A, X, L, l_0, \Delta)$ be a register automaton and f a $CTL(X,L)$ formula. We say *RA satisfies* f (written $\models_{RA} f$) if $\langle l_0, v\rangle \models_{RA} f$ for every $v \in V_{(X,\Sigma)}$. The $CTL(X,L)$ *model checking* problem for register automata is to decide whether $\models_{RA} f$. The following lemma shows that any two configurations in a representative configuration satisfy the same $CTL(X,L)$ formulae.

Lemma 7. *Let $RA = (\Sigma, A, X, L, l_0, \Delta)$ be a register automaton, $l \in L$, $u, v \in V_{(X,\Sigma)}$, and f a $CTL(X,L)$ formula. If $u \sim_C v$, then*

$$\langle l, u\rangle \models_{RA} f \text{ if and only if } \langle l, v\rangle \models_{RA} f.$$

By Lemma 7, it suffices to compute representative configurations for any $CTL(X,L)$ formula. For any $CTL(X,L)$ formula f, we compute the set of representative configurations $\{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} f\}$. Our model checking algorithm essentially follows the classical algorithm for finite-state models.

Input: RA: $(\Sigma, A, X, L, l_0, \Delta)$; ap : a $CTL(X,L)$ atomic formula
Output: $\{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} ap\}$
$\mathscr{R} \leftarrow \mathsf{UniverseU}(X)$;
switch ap **do**
  case l: **return** $\{\langle l, R\rangle | R \in \mathscr{R}\}$;
  case $x_i = x_j$: **return** $L \times \{R \in \mathscr{R} | R_{ij} = \bar{1} \text{ or } R_{ij} = c \in C\}$;
  case $x_i = c$: **return** $L \times \{R \in \mathscr{R} | R_{ii} = c \in C\}$;
endsw

Algorithm 5: $\mathsf{ComputeAP}(RA, ap)$

Algorithm 5 computes the set of representative configurations for atomic propositions. Clearly, $\mathsf{ComputeAP}(RA, ap) = \{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} ap\}$.

Input: RA: $(\Sigma, A, X, L, l_0, \Delta)$; S : $\{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} f\}$
Output: $\{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} \neg f\}$
$\mathscr{R} \leftarrow \mathsf{UniverseR}(X)$;
return $(L \times \mathscr{R}) \setminus S$;

Algorithm 6: $\mathsf{ComputeNot}(RA, S)$

Input: RA : $(\Sigma, A, X, L, l_0, \Delta)$; S_0 : $\{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} f_0\}$; S_1 : $\{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} f_1\}$
Output: $\{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} f_0 \wedge f_1\}$
return $S_0 \cap S_1$;

Algorithm 7: $\mathsf{ComputeAnd}(RA, S_0, S_1)$

For Boolean operations, we assume that representative configurations for operands have been computed. Algorithm 6 and 7 give details for the negation and conjunction of $CTL(X,L)$ formulae respectively.

Given the set S of representative configurations for a $CTL(X,L)$ formula f, Algorithm 8 shows how to compute representative configurations for $EX f$. For every possible representative configuration $\langle l, R\rangle$, it checks if $\langle l', R'\rangle \in S$ for some $\langle l', R'\rangle$ with $\langle l, R\rangle \rightsquigarrow \langle l', R'\rangle$. If so, $\langle l, R\rangle$ is added to the result.

To compute representative configurations for $f_0 U f_1$, recall that $f_0 U f_1$ is the least fixed point of the function $\Psi(Z) = f_1 \vee (f_0 \wedge EXZ)$. Algorithm 9 thus follows the standard fixed point computation for the $CTL(X,L)$ formula $f_0 U f_1$.

Input: RA: $(\Sigma, A, X, L, l_0, \Delta)$; $S : \{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} f\}$
Output: $\{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} EX\, f\}$
$\mathscr{R}, U \leftarrow \mathsf{UniverseR}(X), \emptyset$;
foreach $\langle l, R\rangle \in L \times \mathscr{R}$ **do**
  if $\mathsf{Post}(RA, \langle l, R\rangle) \cap S \neq \emptyset$ **then** $U \leftarrow U \cup \{\langle l, R\rangle\}$;
end
return U;

Algorithm 8: $\mathsf{ComputeEX}(RA, S)$

Input: RA: $(\Sigma, A, X, L, l_0, \Delta)$; $S_0 : \{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} f_0\}$; $S_1 : \{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} f_1\}$
Output: $\{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} f_0\, U\, f_1\}$
$U, V \leftarrow S_1, \emptyset$;
while $U \neq V$ **do**
  $W \leftarrow \mathsf{ComputeEX}(RA, U)$;
  $V, U \leftarrow U, U \cup (W \cap S_0)$;
end
return U;

Algorithm 9: $\mathsf{ComputeEU}(RA, S_0, S_1)$

Input: $RA : (\Sigma, A, X, L, l_0, \Delta)$; $S : \{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} f\}$
Output: $\{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} EG\, f\}$
$U, V \leftarrow S, \mathsf{UniverseR}(X)$;
while $U \neq V$ **do**
  $W \leftarrow \mathsf{ComputeEX}(RA, U)$;
  $V, U \leftarrow U, U \cap W$;
end
return U;

Algorithm 10: $\mathsf{ComputeEG}(RA, S)$

For the $CTL(X,L)$ formula $EG f$, recall that $EG f$ is the greatest fixed point of the function $\Phi(Z) = f \wedge EXZ$. Algorithm 10 performs the greatest fixed point computation to obtain representative configurations for $EG f$.

Input: $RA : (\Sigma, A, X, L, l_0, \Delta)$; f : a $CTL(X,L)$ formula
Output: $\{\langle l, R_{[v]} \rangle | \langle l, v \rangle \models_{RA} f\}$
switch f **do**
  case l, $x_i = x_j$, *or* $x_i = c$*:*
    $U \leftarrow \mathsf{ComputeAP}(RA, f)$;
  case $\neg f_0$*:*
    $V \leftarrow \mathsf{ComputeCTL}(RA, f_0)$;
    $U \leftarrow \mathsf{ComputeNot}(RA, V)$;
  case $f_0 \wedge f_1$*:*
    $V_0, V_1 \leftarrow \mathsf{ComputeCTL}(RA, f_0), \mathsf{ComputeCTL}(RA, f_1)$;
    $U \leftarrow \mathsf{ComputeAnd}(RA, V_0, V_1)$;
  case $EX f_0$*:*
    $V \leftarrow \mathsf{ComputeCTL}(RA, f_0)$;
    $U \leftarrow \mathsf{ComputeEX}(RA, V)$;
  case $E(f_0 U f_1)$*:*
    $V_0, V_1 \leftarrow \mathsf{ComputeCTL}(RA, f_0), \mathsf{ComputeCTL}(RA, f_1)$;
    $U \leftarrow \mathsf{ComputeEU}(RA, V_0, V_1)$;
  case $EG f_0$*:*
    $V \leftarrow \mathsf{ComputeCTL}(RA, f_0)$;
    $U \leftarrow \mathsf{ComputeEG}(RA, V)$;
endsw
return U;

Algorithm 11: $\mathsf{ComputeCTL}(RA, f)$

The representative configurations for a $CTL(X,L)$ formula are computed by induction on the formula (Algorithm 11). Theorem 4 summaries the algorithm.

Theorem 4. *Let* $RA = (\Sigma, A, X, L, l_0, \Delta)$ *be a register automaton,* f *a* $CTL(X,L)$ *formula,* $l \in L$*, and* $v \in V_{(X,\Sigma)}$*.* $\langle l, v \rangle \models_{RA} f$ *if and only if* $\langle l, R_{[v]} \rangle \in \mathsf{ComputeCTL}(RA, f)$*.*

It is easy to check whether $\models_{RA} f$ for any register automaton RA and $CTL(X,L)$ formula f by Theorem 4 (Algorithm 12). We compute the set $\{\langle l, R_{[v]} \rangle | \langle l, v \rangle \models_{RA} f\}$ of representative configurations and check if $\langle l, R \rangle$ belongs to the set for every representative matrix R.

6 An Example

In the Byzantine generals problem, one commanding and $n-1$ lieutenant generals would like to share information through one-to-one communication. However, not all generals are loyal. Some of them (the commanding general included) may be traitors. Traitors need not follow rules. The problem is to devise a mechanism so that all loyal generals share the same information at the end.

Consider the scenario with a commanding general, two loyal lieutenant, and one treacherous general. The emperor decides to send m soldiers to the front line, and asks the commanding general to inform

Input: $RA : (\Sigma, A, X, L, l_0, \Delta)$; f : a $CTL(X, L)$ formula
Output: *true* if $\models_{RA} f$; *false* otherwise
$U \leftarrow \mathsf{ComputeCTL}(RA, f)$;
$\mathscr{R} \leftarrow \mathsf{UniverseR}(X)$;
$W \leftarrow \{\langle l_0, R\rangle | R \in \mathscr{R}\}$;
$result \leftarrow$ **if** $W \subseteq U$ **then** *true* **else** *false*;
return *result*;

Algorithm 12: $\mathsf{ModelCheck}(RA, f)$

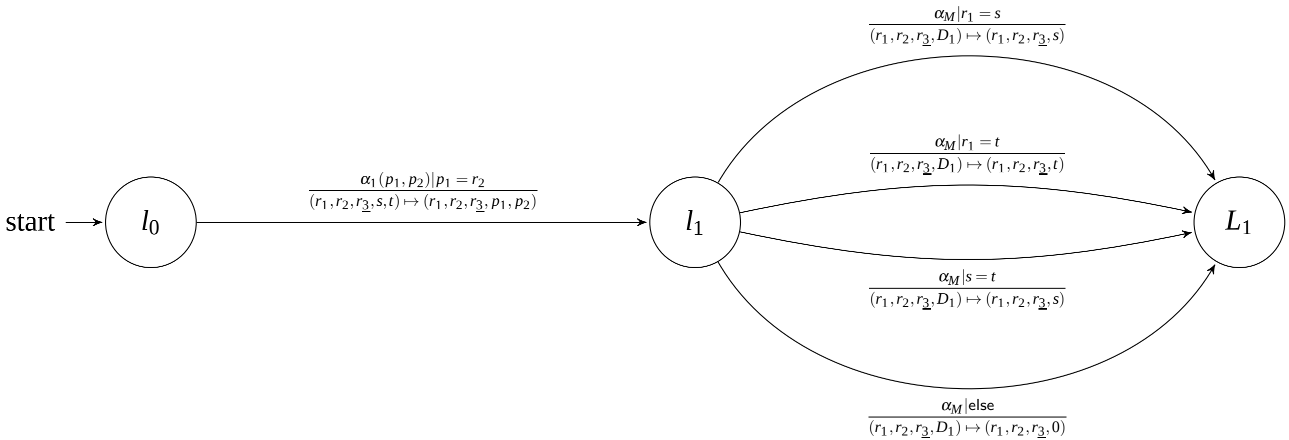


Figure 2: The Lieutenant General 1

the lieutenant generals. Based on the algorithm in [10], we give a model where a loyal, the treacherous, and the other loyal lieutenant generals act in turn. We want to know the initial configurations where both loyal generals agree upon the same information in this setting.

Since the number of soldiers is unbounded, we choose $\mathbb{N}$ as the infinite alphabet. The set of constants C is $\{0\}$, it is for default decision. When a lieutenant general cannot decide, he will take the default decision. We identify lieutenant generals by numbers: 1 and 2 are loyal, $\underline{3}$ is treacherous. The action set A has four actions: $\alpha_1, \alpha_2, \alpha_{\underline{3}}$, and α_M. The action α_i means that the lieutenant general i receives messages from the other lieutenant generals. Each lieutenant general computes the majority of messages in action α_M. Eight registers will be used. The registers $r_1, r_2, r_{\underline{3}}$ contain the commanding general's messages sent to each lieutenant general respectively. The final decisions of each lieutenant generals are stored in the registers D_1, D_2, and $D_{\underline{3}}$ respectively. Finally, s and t are temporary registers.

Assume the lieutenant generals have received a decision from the commanding general initially. Since the commanding general may be treacherous, the registers r_1, r_2, r_3 have arbitrary values at location l_0 (Figure 2).

In our scenario, the lieutenant general 1 acts first. He receives two messages from the other lieutenant generals in the action $\alpha_1(p_1, p_2)$. Since the lieutenant general 2 is loyal, he sends the message received from the commanding general. Thus we have the guard $p_1 = r_2$. The message from the lieutenant general $\underline{3}$ is arbitrary because the general is treacherous. We record the messages from the lieutenant generals 2 and $\underline{3}$ in the registers s and t respectively (location l_1). The lieutenant general 1 makes his decision by the majority of the message from the commanding general (r_1), the message from the lieutenant general

2 (s), and the message from the treacherous lieutenant general $\underline{3}$ (t). For instance, if the messages from the other lieutenant generals are equal ($s = t$), the lieutenant general 1 will have his decision equal to s through the transition $(l_1, \alpha_M, s = t, (r_1, r_2, r_{\underline{3}}, D_1) \mapsto (r_1, r_2, r_{\underline{3}}, s), L_1)$.

The other lieutenant generals are modeled similarly. Appendix A gives the model in register automata for the scenario where the location L_2 denotes the end of communication. Since the commanding general is not necessary loyal, we are interested in finding initial configurations that satisfy the $CTL(X,L)$ property $AF(D_1 = D_2) \equiv \neg EG\neg(D_1 = D_2)$.

Let $\mathscr{R} = \mathsf{UniverseR}(X)$ be the set of representative matrices. We begin with $U_0 = \{\langle l, R_{[v]}\rangle | \langle l, v\rangle \models_{RA} \neg(D_1 = D_2)\} = L \times \{R_{[v]} | v(D_1) \neq v(D_2)\}$. Then $W_0 = \mathsf{ComputeEX}(RA, U_0) = (\{l_0, l_1, L_1, L_3\} \times \mathscr{R}) \cup \{\langle l_2, R_{[v]}\rangle | (v(r_2) = v(s) \wedge v(D_1) \neq v(s)) \vee (v(r_2) = v(t) \wedge v(D_1) \neq v(t)) \vee (v(s) = v(t) \wedge v(D_1) \neq v(s)) \vee (v(r_2) \neq v(s) \wedge v(r_2) \neq v(t) \wedge v(s) \neq v(t) \wedge v(D_1) \neq v(0))\} \cup \{\langle L_2, R_{[v]}\rangle | v(D_1) \neq v(D_2)\}$. Consider a configuration $\langle l_1, v_1\rangle \in \langle l_1, R_{[v_1]}\rangle \in W_0$. Since the outgoing transitions at location l_1 do not assign values to the register D_2, D_2 can have an arbitrary value at the location L_1. Particularly, $\langle l_1, v_1\rangle \xrightarrow{\alpha_M} \langle l_1, v_1'\rangle$ for some $v_1'(D_2) \neq v_1'(D_1)$. We have $\langle l_1, v_1\rangle \models_{RA} EX\neg(D_1 = D_2)$. More interestingly, let us consider another configuration $\langle l_2, v_2\rangle \in \langle l_2, R_{[v_2]}\rangle \in W_0$ with $v_2(s) = v_2(t) \wedge v_2(D_1) \neq v_2(s)$. Since $v_2(s) = v_2(t)$, the register D_2 will be assigned to the value of the register s by the transition $(l_2, \alpha_M, s = t, (r_1, r_2, r_{\underline{3}}, D_1, D_2, D_3) \mapsto (r_1, r_2, r_{\underline{3}}, D_1, s, D_3), L_2)$ (Figure 3). Particularly, define $v_2'(D_2) = v_2(s)$ and $v_2'(x) = v_2(x)$ for $x \neq s$. We have $\langle l_2, v_2\rangle \xrightarrow{\alpha_M} \langle L_2, v_2'\rangle$, $v_2'(D_2) = v_2(s) \neq v_2(D_1) = v_2'(D_1)$, and $\langle L_2, v_2'\rangle \models_{RA} \neg(D_1 = D_2)$. $\langle l_2, v_2\rangle \models_{RA} EX\neg(D_1 = D_2)$.

We manually compute the representative configurations obtained by $\mathsf{ComputeCTL}(RA, EG\neg(D_1 = D_2))$ (Appendix B). Particularly, we have $\{\langle l_0, R_{[v]}\rangle | D_1 = D_2 \vee r_1 = r_2\} \subseteq \mathsf{ComputeCTL}(RA, AF(D_1 = D_2))$. The loyal lieutenant generals will agree on the same information provided they have the same decision, or the commanding general sends them the same message initially.

7 Conclusion

We develop an exact finitary representation for valuations in register automata. Based on representative valuations, we show that the reachability problem for register automata is decidable. We also define $CTL(X,L)$ for register automata and propose a model checking algorithm for the logic. As an illustration, we model a scenario in the Byzantine generals problem. We discuss the initial condition for correctness by the $CTL(X,L)$ model checking algorithm in the example.

$CTL(X,L)$ has very primitive modal operators. We believe that our technique applies to more expressive modal μ-calculus. It will also be interesting to investigate structured infinite alphabets. For instance, a totally ordered infinite alphabet is useful in the bakery algorithm. Representative valuations for such infinite alphabets will be essential to verification as well.

References

[1] Sofia Cassel, Falk Hower, Bengt Jonsson, Maik Merten & Bernhard Steffen (2011): *A Succinct Canonical Register Automaton Model*. *ATVA 2011* LNCS 6996, pp. 366–380, doi:10.1007/978-3-642-24372-1_26.

[2] Sofia Cassel, Falk Hower, Bengt Jonsson & Bernhard Steffen (2012): *Inferring Canonical Register Automata*. *VMCAI 2012* LNCS 7148, pp. 251–266, doi:10.1007/978-3-642-27940-9_17.

[3] Edmund M. Clarke, Jr., Orna Grumberg & Doron A. Peled (1999): *Model Checking*. MIT Press.

[4] Stphane Demri & Ranko Lazic (2009): *LTL with the Freeze Quantifier and Register Automata*. *ACM Transactions on Computational Logic* 10(16), doi:10.1145/1507244.1507246.

[5] Diego Figueira, Piotr Hofman & Slawomir Lasota (2010): *Relating timed and register automata*. *EPTCS 41*, pp. 61–75, doi:10.4204/EPTCS.41.5.

[6] Michael Kaminiski & Nissim Francez (1994): *Finite-memory automata*. *Theoretical Computer Science* 134, pp. 329–363, doi:10.1016/0304-3975(94)90242-9.

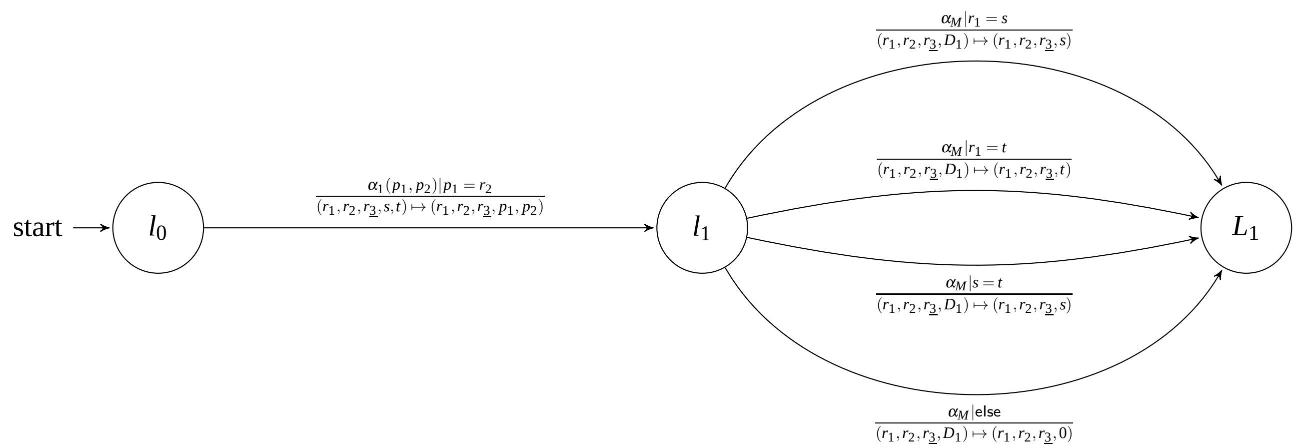
[7] D. Kroening & O. Strichman (2008): *Decision Procedures - an algorithmic point of view*. EATCS, Springer, doi:10.1007/s10817-013-9295-4.

[8] Leslie Lamport, Robert Shostak & Marshall Pease (1982): *The Byzantine Generals Problem*. *ACM Transactions on Programming Languages and Systems* 4(3), pp. 382–401, doi:10.1145/357172.357176.

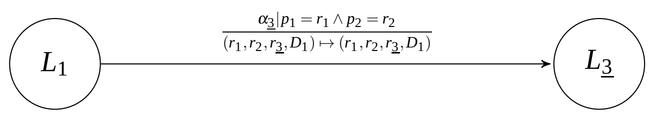
[9] Frank Neven, Thomas Schwentick & Victor Vianu (2004): *Finite State Machines for Strings Over Infinite Alphabets*. *ACM Transsactions on Computational Logic* 5(3), pp. 403–435, doi:10.1145/1013560.1013562.

[10] Junxing Wang (2012): *A Simple Byzantine Generals Protocol*. *Journal of Combinatorial Optimization*, doi:10.1007/s10878-012-9534-3.

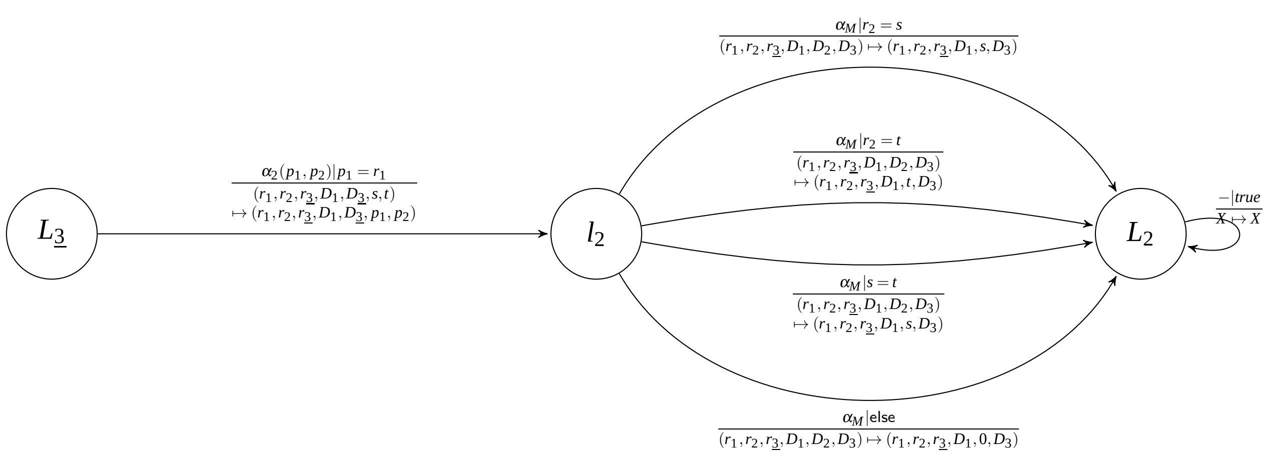
A A Scenario of the Byzantine Generals Problem



(a) Lieutenant General 1



(b) Lieutenant General $\underline{3}$



(c) Lieutenant General 2

Figure 3: The Byzantine Generals Problem

Figure 3 shows the register automaton for the scenario described in Section 6. The transition $\frac{-|true}{X\mapsto X}$ at location L_2 denotes that the automaton keeps the same valuation upon reading any data symbol at location L_2.

B ComputeCTL$(RA, EG\neg(D_1 = D_2))$

Let $\mathscr{R} = \mathsf{UniverseR}(X)$. In the following, the set comprehension represents the requirements of valuations. For instance, the notation $\{R_{[v]} \mid D_1 \neq D_1\}$ denotes the set $\{R_{[v]} \mid v(D_1) \neq v(D_2)\}$. The following table shows the details of computation.

U_0	$L \times \{R_{[v]} \mid D_1 \neq D_2\}$
W_0	$\{l_0, l_1, L_1, L_3\} \times \mathscr{R} \quad \cup$ $\left\{ \langle l_2, R_{[v]} \rangle \;\middle\vert\; \begin{array}{l} (r_2 = s \wedge D_1 \neq s) \vee (r_2 = t \wedge D_1 \neq t) \vee (s = t \wedge D_1 \neq s) \vee \\ (r_2 \neq s \wedge r_2 \neq t \wedge s \neq t \wedge D_1 \neq 0) \end{array} \right\} \quad \cup$ $\{\langle L_2, R_{[v]} \rangle \mid D_1 \neq D_2\}$
U_1	$\{l_0, l_1, L_1, L_3\} \times \{R_{[v]} \mid D_1 \neq D_2\} \quad \cup$ $\left\{ \langle l_2, R_{[v]} \rangle \;\middle\vert\; \begin{array}{l} (D_1 \neq D_2) \wedge \\ ((r_2 = s \wedge D_1 \neq s) \vee (r_2 = t \wedge D_1 \neq t) \vee (s = t \wedge D_1 \neq s) \vee \\ (r_2 \neq s \wedge r_2 \neq t \wedge s \neq t \wedge D_1 \neq 0)) \end{array} \right\} \quad \cup$ $\{\langle L_2, R_{[v]} \rangle \mid D_1 \neq D_2\}$
W_1	$\{l_0, l_1, L_1\} \times \mathscr{R} \quad \cup$ $\{\langle L_3, R_{[v]} \rangle \mid (D_1 \neq r_2) \vee (D_1 \neq 0 \wedge r_1 \neq r_2)\} \quad \cup$ $\left\{ \langle l_2, R_{[v]} \rangle \;\middle\vert\; \begin{array}{l} (D_1 \neq D_2) \wedge \\ ((r_2 = s \wedge D_1 \neq s) \vee (r_2 = t \wedge D_1 \neq t) \vee (s = t \wedge D_1 \neq s) \vee \\ (r_2 \neq s \wedge r_2 \neq t \wedge s \neq t \wedge D_1 \neq 0)) \end{array} \right\} \quad \cup$ $\{\langle L_2, R_{[v]} \rangle \mid D_1 \neq D_2\}$
U_2	$\{l_0, l_1, L_1\} \times \{R_{[v]} \mid D_1 \neq D_2\} \quad \cup$ $\{\langle L_3, R_{[v]} \rangle \mid (D_1 \neq D_2) \wedge (D_1 \neq r_2) \vee (D_1 \neq 0 \wedge r_1 \neq r_2))\} \quad \cup$ $\left\{ \langle l_2, R_{[v]} \rangle \;\middle\vert\; \begin{array}{l} (D_1 \neq D_2) \wedge \\ ((r_2 = s \wedge D_1 \neq s) \vee (r_2 = t \wedge D_1 \neq t) \vee (s = t \wedge D_1 \neq s) \vee \\ (r_2 \neq s \wedge r_2 \neq t \wedge s \neq t \wedge D_1 \neq 0)) \end{array} \right\} \quad \cup$ $\{\langle L_2, R_{[v]} \rangle \mid D_1 \neq D_2\}$
W_2	$\{l_0, l_1\} \times \mathscr{R} \quad \cup$ $\{\langle L_1, R_{[v]} \rangle \mid (D_1 \neq r_2) \vee (D_1 \neq 0 \wedge r_1 \neq r_2)\} \quad \cup$ $\{\langle L_3, R_{[v]} \rangle \mid (D_1 \neq D_2) \wedge (D_1 \neq r_2) \vee (D_1 \neq 0 \wedge r_1 \neq r_2))\} \quad \cup$ $\left\{ \langle l_2, R_{[v]} \rangle \;\middle\vert\; \begin{array}{l} (D_1 \neq D_2) \wedge \\ ((r_2 = s \wedge D_1 \neq s) \vee (r_2 = t \wedge D_1 \neq t) \vee (s = t \wedge D_1 \neq s) \vee \\ (r_2 \neq s \wedge r_2 \neq t \wedge s \neq t \wedge D_1 \neq 0)) \end{array} \right\} \quad \cup$ $\{\langle L_2, R_{[v]} \rangle \mid D_1 \neq D_2\}$
U_3	$\{l_0, l_1\} \times \{R_{[v]} \mid D_1 \neq D_2\} \quad \cup$ $\{\langle L_1, R_{[v]} \rangle \mid (D_1 \neq D_2) \wedge ((D_1 \neq r_2) \vee (D_1 \neq r_1) \vee (D_1 \neq 0 \wedge r_1 \neq r_2))\} \quad \cup$ $\{\langle L_3, R_{[v]} \rangle \mid (D_1 \neq D_2) \wedge (D_1 \neq r_2) \vee (D_1 \neq 0 \wedge r_1 \neq r_2))\} \quad \cup$ $\left\{ \langle l_2, R_{[v]} \rangle \;\middle\vert\; \begin{array}{l} (D_1 \neq D_2) \wedge \\ ((r_2 = s \wedge D_1 \neq s) \vee (r_2 = t \wedge D_1 \neq t) \vee (s = t \wedge D_1 \neq s) \vee \\ (r_2 \neq s \wedge r_2 \neq t \wedge s \neq t \wedge D_1 \neq 0)) \end{array} \right\} \quad \cup$ $\{\langle L_2, R_{[v]} \rangle \mid D_1 \neq D_2\}$

W_3	$\begin{array}{lr} \{l_0\}\times\mathscr{R} & \cup \\ \left\{ \langle l_1,R_{[v]}\rangle \;\middle\vert\; \begin{array}{l} [(r_1=s\wedge s\neq r_2)\vee(r_1=t\wedge t\neq r_2)\vee(s=t\wedge s\neq r_2)\vee \\ \quad (r_1\neq s\wedge r_1\neq t\wedge s\neq t\wedge r_2\neq 0)]\vee \\ [(r_1=s\wedge s\neq 0\wedge r_1\neq r_2)\vee(r_1=t\wedge t\neq 0\wedge r_1\neq r_2)\vee \\ \quad (s=t\wedge s\neq 0\wedge r_1\neq r_2)] \end{array}\right\} & \cup \\ \{\langle L_1,R_{[v]}\rangle \mid (D_1\neq D_2)\wedge((D_1\neq r_2)\vee(D_1\neq r_1)\vee(D_1\neq 0\wedge r_1\neq r_2))\} & \cup \\ \{\langle L_3,R_{[v]}\rangle \mid (D_1\neq D_2)\wedge(D_1\neq r_2)\vee(D_1\neq 0\wedge r_1\neq r_2))\} & \cup \\ \left\{ \langle l_2,R_{[v]}\rangle \;\middle\vert\; \begin{array}{l} (D_1\neq D_2)\wedge \\ ((r_2=s\wedge D_1\neq s)\vee(r_2=t\wedge D_1\neq t)\vee(s=t\wedge D_1\neq s)\vee \\ (r_2\neq s\wedge r_2\neq t\wedge s\neq t\wedge D_1\neq 0)) \end{array}\right\} & \cup \\ \{\langle L_2,R_{[v]}\rangle \mid D_1\neq D_2\} & \end{array}$
U_4	$\begin{array}{lr} \{\langle l_0,R_{[v]}\rangle \mid D_1\neq D_2\} & \cup \\ \left\{ \langle l_1,R_{[v]}\rangle \;\middle\vert\; \begin{array}{l} (D_1\neq D_2)\wedge \\ ([(r_1=s\wedge s\neq r_2)\vee(r_1=t\wedge t\neq r_2)\vee(s=t\wedge s\neq r_2)\vee \\ \quad (r_1\neq s\wedge r_1\neq t\wedge s\neq t\wedge r_2\neq 0)]\vee \\ \; [(r_1=s\wedge s\neq 0\wedge r_1\neq r_2)\vee(r_1=t\wedge t\neq 0\wedge r_1\neq r_2)\vee \\ \quad (s=t\wedge s\neq 0\wedge r_1\neq r_2)]) \end{array}\right\} & \cup \\ \{\langle L_1,R_{[v]}\rangle \mid (D_1\neq D_2)\wedge((D_1\neq r_2)\vee(D_1\neq r_1)\vee(D_1\neq 0\wedge r_1\neq r_2))\} & \cup \\ \{\langle L_3,R_{[v]}\rangle \mid (D_1\neq D_2)\wedge(D_1\neq r_2)\vee(D_1\neq 0\wedge r_1\neq r_2))\} & \cup \\ \left\{ \langle l_2,R_{[v]}\rangle \;\middle\vert\; \begin{array}{l} (D_1\neq D_2)\wedge \\ ((r_2=s\wedge D_1\neq s)\vee(r_2=t\wedge D_1\neq t)\vee(s=t\wedge D_1\neq s)\vee \\ (r_2\neq s\wedge r_2\neq t\wedge s\neq t\wedge D_1\neq 0)) \end{array}\right\} & \cup \\ \{\langle L_2,R_{[v]}\rangle \mid D_1\neq D_2\} & \end{array}$
W_4	$\begin{array}{lr} \{\langle l_0,R_{[v]}\rangle \mid r_1\neq r_2\} & \cup \\ \left\{ \langle l_1,R_{[v]}\rangle \;\middle\vert\; \begin{array}{l} (D_1\neq D_2)\wedge \\ ([(r_1=s\wedge s\neq r_2)\vee(r_1=t\wedge t\neq r_2)\vee(s=t\wedge s\neq r_2)\vee \\ \quad (r_1\neq s\wedge r_1\neq t\wedge s\neq t\wedge r_2\neq 0)]\vee \\ \; [(r_1=s\wedge s\neq 0\wedge r_1\neq r_2)\vee(r_1=t\wedge t\neq 0\wedge r_1\neq r_2)\vee \\ \quad (s=t\wedge s\neq 0\wedge r_1\neq r_2)]) \end{array}\right\} & \cup \\ \{\langle L_1,R_{[v]}\rangle \mid (D_1\neq D_2)\wedge((D_1\neq r_2)\vee(D_1\neq r_1)\vee(D_1\neq 0\wedge r_1\neq r_2))\} & \cup \\ \{\langle L_3,R_{[v]}\rangle \mid (D_1\neq D_2)\wedge(D_1\neq r_2)\vee(D_1\neq 0\wedge r_1\neq r_2))\} & \cup \\ \left\{ \langle l_2,R_{[v]}\rangle \;\middle\vert\; \begin{array}{l} (D_1\neq D_2)\wedge \\ ((r_2=s\wedge D_1\neq s)\vee(r_2=t\wedge D_1\neq t)\vee(s=t\wedge D_1\neq s)\vee \\ (r_2\neq s\wedge r_2\neq t\wedge s\neq t\wedge D_1\neq 0)) \end{array}\right\} & \cup \\ \{\langle L_2,R_{[v]}\rangle \mid D_1\neq D_2\} & \end{array}$

Finally, the following representative configurations satisfy $EG\neg(D_1=D_2)$.

$$\begin{array}{lr} \{\langle l_0,R_{[v]}\rangle \mid D_1\neq D_2\wedge r_1\neq r_2\} & \cup \\ \left\{ \langle l_1,R_{[v]}\rangle \;\middle\vert\; \begin{array}{l} (D_1\neq D_2)\wedge \\ ([(r_1=s\wedge s\neq r_2)\vee(r_1=t\wedge t\neq r_2)\vee(s=t\wedge s\neq r_2)\vee \\ \quad (r_1\neq s\wedge r_1\neq t\wedge s\neq t\wedge r_2\neq 0)]\vee \\ \; [(r_1=s\wedge s\neq 0\wedge r_1\neq r_2)\vee(r_1=t\wedge t\neq 0\wedge r_1\neq r_2)\vee \\ \quad (s=t\wedge s\neq 0\wedge r_1\neq r_2)]) \end{array}\right\} & \cup \\ \{\langle L_1,R_{[v]}\rangle \mid (D_1\neq D_2)\wedge((D_1\neq r_2)\vee(D_1\neq 0\wedge r_1\neq r_2))\} & \cup \\ \{\langle L_3,R_{[v]}\rangle \mid (D_1\neq D_2)\wedge((D_1\neq r_2)\vee(D_1\neq 0\wedge r_1\neq r_2))\} & \cup \\ \left\{ \langle l_2,R_{[v]}\rangle \;\middle\vert\; \begin{array}{l} (D_1\neq D_2)\wedge \\ [(r_2=s\wedge D_1\neq s)\vee(r_2=t\wedge D_1\neq t)\vee(s=t\wedge D_1\neq s)\vee \\ \quad (r_2\neq s\wedge r_2\neq t\wedge s\neq t\wedge D_1\neq 0)] \end{array}\right\} & \cup \\ \{\langle L_2,R_{[v]}\rangle \mid D_1\neq D_2\} & \end{array}$$