A Taxonomy and Reductions for Common Register Automata Formalisms*

Simon Dierl[0000-0001-9730-9335] and Falk Howar[0000-0002-9524-4459]

Department of Computer Science, TU Dortmund University, Germany
{simon.dierl,falk.howar}@cs.tu-dortmund.de

Abstract. Register automata model languages over infinite alphabets. A number of publications define different register automata formalisms. Equal expressiveness has been conjectured for many formalisms but a formal analysis is still open. In this paper on the occasion of the 63rd birthday of Bengt Jonsson we examine if these formalisms are equally expressive. We define a taxonomy to describe the different formalisms. By combining small-step reductions, we demonstrate that all models have equal expressiveness. We link these to model-specific complexity results for the NonEmptiness problem and decide which taxonomy features determine the complexity of NonEmptiness. The taxonomy enables formal classification of future models. The reductions permit transfer of formalism-specific results to other formalisms.

Keywords: Register Automata · Non-Emptiness · Decidability · Expressiveness

1 Introduction

Finite state machines are a common tool for modeling languages over finite alphabets and are amenable to algorithmic analysis. In recent years, the study of automata operating on *infinite* alphabets has gained some attention, e.g., in the field of automata learning [12]. The first extension of finite state machines to infinite alphabets was proposed by Kaminski and Francez [15]. An example of such a *register automaton* that can recognize strings in which a single character is repeated is given in Fig. 1. Subsequently, register automata have been extensively studied in literature. Many publications choose to use their own register automaton models, or variants of existing models. These models are more amenable to proofs or can express real-world concerns more succinctly. The succinct canonical register automaton [5,6] in Fig. 2, for example, expresses a fragment of the XMPP instant messaging protocol [21] in a way that is concise, easily understood, and can be inferred algorithmically [13] from tests.

While occasionally, the expressiveness of such formalisms has been noted to at least capture finite-memory automata (e.g., in [11]), no formal study of

* The final authenticated version is available online at https://doi.org/10.1007/978-3-030-91384-7_10

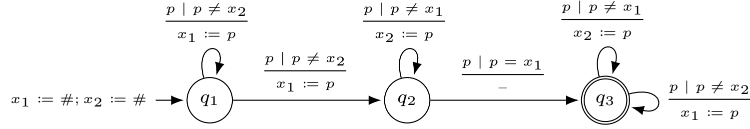


Fig. 1. Nondeterministic finite-memory automaton recognizing inputs in which at least one symbol occurs twice [15, Figure 1]. In the initial state q_1 the automaton reads and stores input until the duplicated symbol occurs. The symbol is recognized nondeterministically and state q_2 is entered. Input is read and stored until the saved symbol reoccurs. Then, the accepting state q_3 is reached.

the different models' expressiveness has been conducted. Babari et al. provide a taxonomy for extensions to the model of Kaminsky and Francez [1]. The complexity of NonEmptiness for different models of register interaction ("register disciplines") was examined by Murawski et al. in [17,18]. Cassel et al. [6] show that for some variants of their register automata that mimic restrictions imposed by other register automata formalisms the size of the automaton representation can blow up exponentially while expressiveness is not affected by the restrictions. Correspondingly, different complexity results have been proven for NonEmptiness: for Kaminsky and Francez's finite memory automata [15], the problem is NP-complete [22], for Murawski et al.'s more restricted model, it is NL-complete [17,18] and for Demri and Lazić's automata, it is PSPACE-complete [11]. A formal relation of different types of register automata would not only further our understanding of automata over infinite alphabets as a whole but could also serve as a basis for implementing and porting existing algorithms, e.g., in libraries for automata learning algorithms like LearnLib [14] or RALib [4]. Learning algorithms have already been extended from register automata to some classes of more expressive extended finite state machines [7]. A formal analysis of the differences in expressiveness of different automata models (or lack thereof) may serve as a basis for further extensions and help understanding limits of expressivity.

In this paper on the occasion of the 63rd birthday of Bengt Jonsson, we provide a four-feature-taxonomy describing the most common types of register automata. For each feature, we define variants describing the types and individual reductions between variants to prove their equal expressiveness. We also present upper and lower bounds on the complexity of these reductions. Together with results from the literature on the complexity of deciding NonEmptiness in several register automaton models, we obtain a detailed characterization of the subtle differences between automata models as well as their expressiveness in relation to their size.

Outline. Section 2 will introduce data languages and provide a definition of generic register automata that encompasses all definitions found in the literature. Next, we will introduce a taxonomy of reductions in Section 3 and the taxonomy of register automata in Section 4, which will be applied to register automata

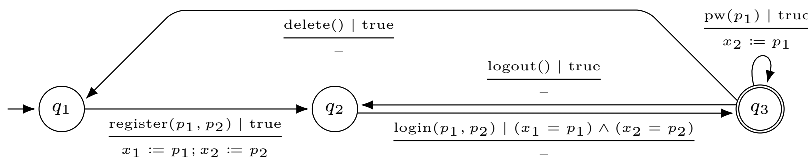


Fig. 2. Succinct Canonical Register Automaton that recognizes successful XMPP registrations and logins for a single user [5, Figure 1]. In the initial state q_1, no user is registered. By registering an account, state q_2 is entered and the credentials stored. A login with matching credentials enters state q_3. The logged-in user can update their password, log out or delete the account altogether.

definitions from the literature. Section 5 describes small-step reductions between the taxonomy elements, including some lower complexity bounds; Section 6 combines these to construct reductions and lower bounds between the existing models. Section 7 summarizes our findings and describes possible extensions.

2 Preliminaries

We start by defining notation for some fundamental concepts.

Definition 1 (Power Set). *Given a set* S*,* $\mathfrak{P}(S) = \{S' \mid S' \subseteq S\}$ *is the* power set *of* S*.*

Definition 2 (Image). *Given a function* $f : A \to B$*,* $\mathfrak{I}(f) = \{f(a) \mid a \in A\}$ *denotes the* image *of* f*.*

Register automata operate on a combination of a finite and an infinite alphabet. The finite alphabet defines labels that are then combined with values from the infinite alphabet. We now formally define these combinations.

Definition 3 (Data Universe, Symbol, Word, Language). *A* data universe *is a tuple* $\mathcal{D} = (\Lambda, D, \mathfrak{a})$ *with a finite set* Λ *of* labels*, an infinite set* D *of* (data) values*, and an* arity *function* $\mathfrak{a} : \Lambda \to \mathbb{Z}_{\geq 0}$*. For a given label* λ*, the vector of* formal parameters *is* $P^\lambda = (p_1, \ldots, p_{\mathfrak{a}(\lambda)})$*. A* data symbol *is a tuple* $(\lambda, \vec{d})$ *with* $\lambda \in \Lambda$ *and a vector of data values* $\vec{d}$ *with* $|\vec{d}| = \mathfrak{a}(\lambda)$*. We usually write a symbol as* $\lambda(d_1, \ldots, d_{\mathfrak{a}(\lambda)})$*. A* data word *is a sequence of data symbols. A set of data words from the same data universe is a* data language*.*

Now, we provide a definition for register automata with equality and inequality comparisons. While in the literature, similar automata with more operations (e.g., less-than comparisons) have been studied, their expressiveness and theoretical properties are different from classic register automata. Our definition is designed to subsume all equality-based models present in the literature. These will later be represented as constraints for the following definitions.

Definition 4 (Register Automaton). *A* register automaton (RA) *is a tuple* $\mathcal{A} = (\mathcal{D}, Q, q_0, Q^+, X, X_Q, \chi_0, \Gamma)$*, defining*

- *a data universe* $\mathcal{D} = (\Lambda, D, \mathfrak{a})$*,*
- *a finite set of* states Q*,*
- *an* initial state $q_0 \in Q$*,*
- accepting states $Q^+ \subseteq Q$*,*
- *a finite set of* registers X *that can store data values,*
- *a* visibility function $X_Q : Q \to \mathfrak{P}(X)$*,*
- *an* initial valuation $\chi_0 : X \to D \cup \{\#\}$ *such that* $\# \notin D$ *is the* empty value *and for all* $x \notin X_Q(q_0)$*,* $\chi_0(x) = \#$*, and*
- *a set* Γ *of* transitions $\langle q, q', \lambda, g, u\rangle$*, each defining*
  - *a* source state $q \in Q$*,*
  - *a* target state $q' \in Q$*,*
  - *a label* $\lambda \in \Lambda$*,*
  - *a* guard g*, i.e. a propositional logic formula with an equality relation over free variables from* $X_Q(q) \cup P^\lambda$*, and*
  - *an* update $u : X_Q(q') \to (X_Q(q) \cup P^\lambda)$ *that selects new values for the registers visible in the target state, i.e.,* $u(x) = v$ *if the value of register or parameter* v *is copied to* x*.*

A transition $\langle q, q', \lambda, g, u\rangle$ is always written as

$$\frac{\lambda(p_1, \dots, p_{|\mathfrak{a}(\lambda)|}) \mid g}{u},$$

where $p_1, \dots, p_{|\mathfrak{a}(\lambda)|}$ are the formal parameters, g is the guard and u is a set of parallel updates $x_i \coloneqq v$ with $v \in X_Q(q) \cup P^\lambda$. If no explicit assignment to a register $x_i \in X_Q(q) \cap X_Q(q')$ is given, the assignment $x_i \coloneqq x_i$ is implicitly assumed.

Definition 5 (Deterministic Register Automaton). *A register automaton is* deterministic *if for each pair of transitions* $\langle q, q'_1, \lambda, g_1, u_1\rangle$ *and* $\langle q, q'_2, \lambda, g_2, u_2\rangle$ *with identical source state and label,* $(g_1 \wedge g_2)$ *is unsatisfiable.*

Note that our definition does not demand that a valid transition exists (i.e., the disjunction of all guards is a tautology), while e.g. [15, Definition 2] does. This can be rectified by adding a trap state and "missing" transitions. Now, we define how a register automaton processes words.

Definition 6 (State Transition). *For a register automaton* $(\mathcal{D}, Q, q_0, Q^+, X, X_Q, \chi_0, \Gamma)$ *with a transition* $\gamma = \langle q, q', \lambda, g, u\rangle \in \Gamma$*, a* state transition $\mathcal{T} = \langle q, \chi, \lambda(d_1, \dots, d_{\mathfrak{a}(\lambda)}), q', \chi'\rangle$ *is a tuple of*

- source *and* target state q, q'*,*
- *a data symbol* $\lambda(d_1, \dots, d_{\mathfrak{a}(\lambda)}) \in \mathbb{D}_{\mathcal{D}}$*,*

– *a source valuation* $\chi : X_Q(q) \to D \cup \{\#\}$ *such that* g *is satisfied by the valuation* $\nu : X_Q(q) \cup P^\lambda \to D \cup \{\#\}$ *defined as*

$$\nu(v) \coloneqq \begin{cases} \chi(v) & \text{if } v \in X_Q(q) \\ d_i & \text{if } v = p_i, \text{ and} \end{cases}$$

– *a* target valuation $\chi' : X_Q(q') \to D \cup \{\#\}$ *such that* $\chi'(x) = \nu(u(x))$.

Intuitively speaking, an RA automaton accepts a data word if there exists a sequence of state transitions from the initial to an accepting state using the word's symbols.

Definition 7 (Acceptance Behavior). *A register automaton* $A = (\mathcal{D}, Q, q_0, Q^+, X, X_Q, \chi_0, \Gamma)$ accepts *or* rejects *data words from its data universe. A data word* $\lambda_1(\vec{d_1}) \ldots \lambda_k(\vec{d_k})$ *is* accepted *if a sequence of state transitions* $\mathcal{T}_1, \ldots, \mathcal{T}_k$ *exists such that*

- *the source state of* $\mathcal{T}_1$ *is* q_0,
- *the source valuation of* $\mathcal{T}_1$ *is* χ_0,
- *the target state of* $\mathcal{T}_k$ *is in* Q^+,
- *for* $1 \leq i < k$, *the target state and valuation of* $\mathcal{T}_i$ *are the source state and valuation of* $\mathcal{T}_{i+1}$, *and*
- *for* $1 \leq i \leq k$, *the data symbol of* $\mathcal{T}_i$ *is* $\lambda_i(\vec{d_i})$.

A data word that is not accepted is rejected. *The language of words accepted by the automaton is* $L(A)$.

3 Reductions

In the previous section, we described a generic automaton model. Register automata with additional, disparate constraints have been studied in the literature. We call a set of automata with identical constraints a *class (of automata)*. To transfer decidability and complexity results between classes, we define reductions between classes. If the specific type of reduction is apparent from the context, we denote reducibility with the $\preceq$ operator.

Definition 8 (NonEmptiness-Turing Reduction). *Given two classes* C_1 *and* C_2 *of register automata,* C_1 *is* NonEmptiness-Turing reducible (NETR) *to* C_2 *if there exists an algorithm* $\mathcal{A}$ *that determines* NonEmptiness *for any* C_1*-automaton if* $\mathcal{A}$ *has access to an oracle that decides* NonEmptiness *for any* C_2*-automaton.*

Definition 9 (Membership-Turing Reduction). *Given two classes* C_1 *and* C_2 *of register automata,* C_1 *is* Membership-Turing reducible (MTR) *to* C_2 *if there exists an algorithm* $\mathcal{A}$ *that determines acceptance for any* C_1*-automaton and data word in its data universe if* $\mathcal{A}$ *has access to an oracle that decides* Membership *for any* C_2*-automaton and word from its data universe.*

We also define a reduction's complexity as the complexity of the underlying algorithm:

Definition 10 (Reduction Complexity). *Let $\mathfrak{R}$ be a type of reduction. Given a reduction of type $\mathfrak{R}$ such that the algorithm a is computable with complexity T, the reduction is a T-$\mathfrak{R}$.*

Definition 11 (Turing Reduction). *Given two classes C_1 and C_2 of register automata, C_1 is* Turing reducible (TR) *to C_2 if it is both NonEmptiness- and Membership-Turing reducible to C_2. If it is T-NonEmptiness- and T-Membership-Turing reducible for any complexity T, it is* T-Turing-reducible (T-TR)*.*

These reductions are useful to prove lower bounds. We define a more constrained type of reduction as a transformation between automata and inputs analogously to Post [20]. Since the input-modifying function of this model is dependent on both automata's data universes, it is defined as a family of functions.

Definition 12 (Many-One Reduction). *Given two classes C_1 and C_2 of register automata, a* many-one reduction (M1R) *from C_1 to C_2 is a tuple $(f_A, f_{\mathbb{D}^\star}^{\mathcal{D},\mathcal{E}})$, where*

- *$\mathcal{D}$ and $\mathcal{E}$ are variables for data universes,*
- *$f_A : C_1 \to C_2$ is the* automaton reduction*,*
- *$f_{\mathbb{D}^\star}^{\mathcal{D},\mathcal{E}} : C_1 \times \mathbb{D}_{\mathcal{D}}^\star \to \mathbb{D}_{\mathcal{E}}^\star$ is a family of* data reductions*,*
- *given $A \in C_1$ with data universe $\mathcal{D}$ such that $f_A(A)$ has data universe $\mathcal{E}$,*

$$w \in L(A) \iff f_{\mathbb{D}^\star}^{\mathcal{D},\mathcal{E}}(A, w) \in L(f_A(A)), \text{ and}$$

- *$L(A) = \emptyset \iff L(f_A(A)) = \emptyset$.*

We will make this generic definition more specific, since the resulting automaton does not need to resemble the original automaton and, given enough computation time, it is possible to "solve" the original automaton during the reduction and create a trivial reduced instance. We define a more constrained reduction type that requires all modifications to the input to be computationally inexpensive. A linear time bound ensures that each symbol can be examined, but that no non-trivial computation can be performed on it. Additionally, all changes to the input must be independent of the source automaton and the surrounding symbols. Automaton- or word-specific information can only be added to the word as a prefix.

Definition 13 (Linear-Local Reduction). *Given two classes C_1 and C_2 of register automata, a* linear-local reduction (LLR) *from C_1 to C_2 is a tuple $(f_A, f_D^{\mathcal{D},\mathcal{E}}, f_{P^A}^{\mathcal{E}}, f_{P^D}^{\mathcal{D},\mathcal{E}})$, where*

- *$\mathcal{D}$ and $\mathcal{E}$ are variables for data universes,*
- *$f_A : C_1 \to C_2$ is the* automaton reduction*,*

- $f_D^{\mathcal{D},\mathcal{E}} : \mathbb{D}_\mathcal{D} \to \mathbb{D}_\mathcal{E}^\star$ *is a family of linear time computable* data reductions*,*
- $f_{PA}^{\mathcal{E}} : C_1 \to \mathbb{D}_\mathcal{E}^\star$ *is a family of linear time computable* automaton prefix-generating reductions*, and*
- $f_{PD}^{\mathcal{D},\mathcal{E}} : \mathbb{D}_\mathcal{D}^\star \to \mathbb{D}_\mathcal{E}^\star$ *is a family of linear time computable* data prefix-generating reductions

such that $(f_A, f_{\mathbb{D}^\star}^{\mathcal{D},\mathcal{E}})$ *with*

$$f_{\mathbb{D}^\star}^{\mathcal{D},\mathcal{E}}(A, w) = f_{PA}^{\mathcal{E}}(A) f_{PD}^{\mathcal{D},\mathcal{E}}(w) f_D^{\mathcal{D},\mathcal{E}}(w_1) \ldots f_D^{\mathcal{D},\mathcal{E}}(w_{|w|})$$

is a many-one reduction.

In some cases, we can omit the reduction functions to obtain an even simpler reduction.

Definition 14 (Prefix Free). *Given two classes C_1 and C_2 of register automata, a* prefix free reduction (PFR) *from C_1 to C_2 is a tuple $(f_A, f_D^{\mathcal{D},\mathcal{E}})$, where*

- $\mathcal{D}$ *and* $\mathcal{E}$ *are variables for data universes,*
- $f_A : C_1 \to C_2$ *is the* automaton reduction*, and*
- $f_D^{\mathcal{D},\mathcal{E}} : \mathbb{D}_\mathcal{D} \to \mathbb{D}_\mathcal{E}^\star$ *is a family of linear-time-computable* data reductions

such that $(f_A, f_D^{\mathcal{D},\mathcal{E}}, f_{PA}^{\epsilon}, f_{PD}^{\epsilon})$ *with*

$$f_{PA}^{\epsilon}(A) = \epsilon \text{ for all } A \in C_1 \text{ and } f_{PD}^{\epsilon}(w) = \epsilon \text{ for all } w \in \mathbb{D}_\mathcal{D}^\star$$

is a linear-local reduction.

Independently of the presence or absence of prefixes, some reductions do not modify the input word itself, e.g., when constants are transformed into a prefix, but the word is unchanged. We also formalize this property.

Definition 15 (Data Stable). *Given two classes C_1 and C_2 of register automata, a* data stable reduction (DSR) *from C_1 to C_2 is a tuple $(f_A, f_{PA}^{\mathcal{D}}, f_{PD}^{\mathcal{D},\mathcal{D}})$, where*

- $\mathcal{D}$ *is a variable for a data universe,*
- $f_A : C_1 \to C_2$ *is the* automaton reduction*,*
- $f_{PA}^{\mathcal{D}} : C_1 \to \mathbb{D}_\mathcal{D}^\star$ *is a family of linear-time-computable* automaton prefix-generating reductions*, and*
- $f_{PD}^{\mathcal{D},\mathcal{D}} : \mathbb{D}_\mathcal{D}^\star \to \mathbb{D}_\mathcal{D}^\star$ *is a family of linear-time-computable* data prefix-generating reductions

such that $(f_A, f_D^{\mathrm{id}}, f_{PA}^{\mathcal{D}}, f_{PD}^{\mathcal{D},\mathcal{D}})$ *with*

$$f_D^{\mathrm{id}}(d) = d \text{ for all } d \in \mathbb{D}_\mathcal{D}$$

is a linear-local reduction.

Some reductions satisfy both properties, i.e., only the automaton structure is modified.

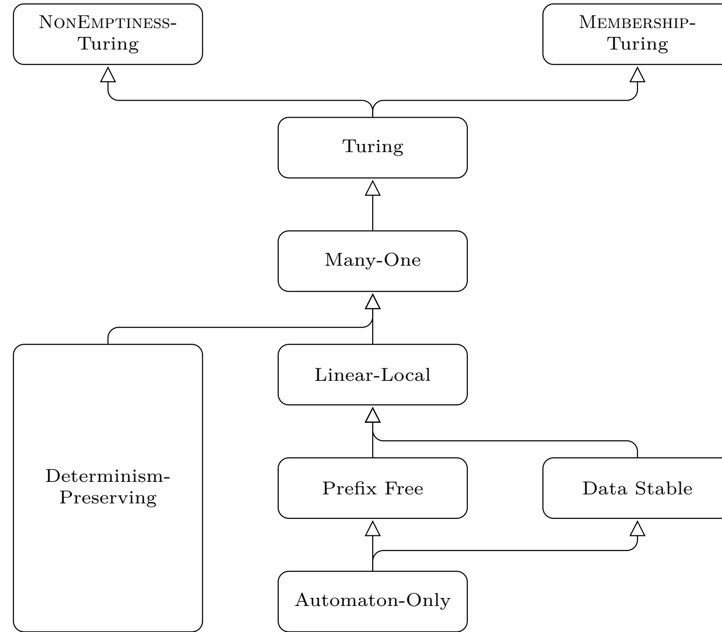


Fig. 3. The different types of reduction. Arrows indicate an is-a relation. If types are horizontally adjacent, a reduction can be a member of any subset of them.

Definition 16 (Automaton-Only Reduction). *Given two classes C_1 and C_2 of register automata, a* automaton-only reduction (AOR) *from C_1 to C_2 is a function $f_A : C_1 \to C_2$ such that $(f_A, f_D^{\mathrm{id}}, f_{P^A}^{\epsilon}, f_{P^D}^{\epsilon})$ is a linear-local reduction.*

The complexity of the Membership problem is lower for deterministic register automata, so we distinguish reductions that preserve the automaton's determinism.

Definition 17 (Determinism-Preserving). *A many-one reduction $(f_A, f_{\mathbb{D}^\star}^{\mathcal{D},\mathcal{E}})$ is* determinism-preserving (DP) *if for a deterministic register automaton A, $f_A(A)$ is deterministic.*

All types of reduction introduced in this section and their relations are outlined in Fig. 3.

4 Taxonomy for Register Automata Formalisms

This section describes the proposed taxonomy for RA models and applies it to existing models from the literature.

Feature	Variants
Data Universe Type	(U-UL) Unlabeled
	(U-LU) Labeled Unary
	(U-LV) Labeled Variadic
Register Availability	(R-UA) Update-Activated
	(R-IN) Initialized
	(R-IE) Initialized or Empty
Update Granularity	(A-PS) Per State
	(A-PT) Per Transition
Guard-Update Model	(G-UP) Update-or-Present
	(G-UA) Update-if-Absent
	(G-FG) Full Guard with Single Update
	(G-NR) No Register-Register Operations
	(G-CC) Conjunction of Comparisons

Table 1. The register automaton taxonomy.

4.1 Proposed Taxonomy

In the literature, more restricted models of register automata than that described in Definition 4 have been studied. Each model adds constraints to certain aspects of the automaton. To describe these classes of register automata, we introduce a taxonomy using four features. It is outlined in Table 1. A class of automata with a common feature variant is described by its variant label, e.g., (G-CC). An intersection of feature variants is denoted with a plus, e.g., (A-PT)+(G-CC). If each automaton in a class is automatically member of another, this is denoted by the $\sqsubseteq$ operator, e.g., (G-NR) $\sqsubseteq$ (G-CC).

For each variant, deterministic and non-deterministic automata can be constructed. Since these are known to differ in expressiveness, we do not include determinism in this taxonomy. The taxonomy is therefore "orthogonal" to the question of determinism.

Data Universe Type. The first feature is the automaton's type of data universe, i.e., the number of labels and their arities. The variants are:

- *Unlabeled* (U-UL). The data universe has a single label λ with arity $\mathfrak{a}(\lambda) = 1$. For brevity, the label is usually omitted.
- *Labeled Unary* (U-LU). The universe contains an arbitrary number of labels, each with arity one.
- *Labeled Variadic* (U-LV). The universe has an arbitrary number of labels, each with arbitrary arity.

By definition, (U-UL) $\sqsubseteq$ (U-LU) $\sqsubseteq$ (U-LV).

Register Availability. The second feature are the semantics of register availability, i.e., if registers are visible in every state and if initial values are provided for the registers. Three models are described in the literature:

- *Update-Activated* (R-UA). Under this model, no registers are visible in the initial state. Therefore, $X_Q(q_0) = \emptyset$ and $\chi_0(x) = \#$ for all $x \in X$. All

registers must be activated by an update operation before becoming visible. Since this will overwrite the register's contents, the empty values are effectively invisible to the automaton.

Initialized (R-IN). Registers are initialized to a non-empty value and all registers are visible in every state, so $X_Q(q) = X$ for all $q \in Q$ and $\chi_0(x) \neq \#$ for all $x \in X$.

Initialized or Empty (R-IE). Registers are initialized to a data value or # and are visible in every state. This is the only type of automaton that can encounter the empty value during a guard evaluation.

By definition, (R-IN) $\sqsubseteq$ (R-IE).

Update Granularity. The third feature describes the scope of update rules. Two models are present in the literature:

Per State (A-PS). For each state, there exists a single canonical update function. Each outgoing transition must either use the source state's canonical update function or keep all registers unchanged. As a result, all outgoing transitions must discard their parameter or write to the same register.

Per Transition (A-PT). Each transition's update can be arbitrarily defined.

By definition, (A-PS) $\sqsubseteq$ (A-PT).

Guard-Update Model. The fourth feature describes the form of guards and updates. Five models are prevalent in the literature:

Update-or-Present (G-UP). This model ensures that no duplicate values (except for #) can be present in the registers. Initial values – if present – must be distinct as well. All data symbols must have arity one. This invariant is maintained by the following transition semantics:

1. First, the update operation is executed. Four scenarios can occur:
   (a) The value is not assigned to a register.
   (b) The value v is assigned to a register x_i, but there exists a register $x_j \neq x_i$ with valuation $\chi(x_j) = v$. The assignment is then ignored.
   (c) The value v is assigned to a register x_i and $\chi(x_i) = v$. The assignment has no observable effect.
   (d) The value v is assigned to a register x_i and $\chi(x) \neq v$ for all $x \in X$. The update then stores the value in x_i.
2. Afterwards, the parameter is tested for equality with a single register, i.e., contrary to our definition, the guard takes the update operation into account. It can thereby check if a write operation was successful under the no-duplicates rule outlined above.

Under this model, two types of transitions can be expressed. They can be transformed to use guard-before-update semantics as follows:

1. The parameter p is assigned to x_i, then x_i is tested for equality with p. This test succeeds if either p was not stored in any register and the assignment was executed or if x_i contained p previously. The previous value of x_i is therefore irrelevant. Using a source state q, this yields following transition:

$$\frac{\lambda(p) \mid \bigwedge_{x \in X_Q(q) \setminus \{x_i\}} (p \neq x)}{x_i \coloneqq p}$$

2. The parameter p is either not assigned or assigned to a register x_j with $j \neq i$, then x_i is tested for equality with p. This can only be the case if the value of p was already present in that register. If p was assigned to x_j, the assignment can therefore have had no effect. Therefore, both cases yield the transition:

$$\frac{\lambda(p) \mid p = x_i}{-}$$

Update-if-Absent (G-UA). This model also does not allow duplicate values and all data symbols must have arity one. This invariant is maintained by allowing two classes of transition:

1. The parameter is tested for equality with a single register. The definition allows for multiple tests, but since no duplicates are present, multiple comparisons cannot succeed:

$$\frac{\lambda(p) \mid p = x_i}{-}.$$

2. Alternatively, an update operation is executed if the value is present in no register. Two scenarios can occur:
   (a) The value v is assigned to a register x_i, but there exists a register $x \in X$ (including $x = x_i$) with valuation $\chi(x) = v$. The transition fails.
   (b) The value v is assigned to a register x_i and $\chi(x) \neq v$ for all $x \in X$. The update then stores the value in x_i.

   Again, the definition allows multiple assignment, but only one attempt can succeed. This can be expressed as:

$$\frac{\lambda(p) \mid \bigwedge_{x \in X_Q(q)} (p \neq x)}{x_i \coloneqq p}.$$

Full Guard with Single Update (G-FG). Duplicate values in registers are permitted. The parameter must be compared (using $=$ or $\neq$) to every visible register. The update may then write the parameter to a single register; no register-to-register assignments aside from the implicit self-assignments are permitted. For example, if $X = \{x_1, x_2, x_3\}$, the following transition satisfies these constraints:

$$\frac{\lambda(p) \mid (x_1 = p) \wedge (x_2 \neq p) \wedge (x_3 \neq p)}{x_2 \coloneqq p}.$$

No Register-Register Operations (G-NR). Guards are a conjunction of comparisons between the parameter and registers. The parameter does not need to be compared to every register. The update may copy the parameter to multiple registers. For example, the following transition satisfies these constraints:

$$\frac{\lambda(p) \mid (x_1 = p) \wedge (x_3 \neq p)}{x_2 \coloneqq p; x_3 \coloneqq p}.$$

Conjunction of Comparisons (G-CC). Guards are a conjunction of parameter-to-parameter, parameter-to-register, and register-to-register comparisons. Updates may copy data from parameters and registers. For example, the following transition satisfies these constraints:

$$\frac{\lambda(p) \mid (x_1 = p) \wedge (x_2 = x_3) \wedge (x_3 \neq p)}{x_2 \coloneqq p; x_3 \coloneqq p}.$$

By definition, (G-UP) $\sqsubseteq$ (G-NR), (G-UA) $\sqsubseteq$ (G-NR), (G-FG) $\sqsubseteq$ (G-NR), and (G-NR) $\sqsubseteq$ (G-CC). (G-UP), (G-UA), and (G-FG) are not contained in one another.

We do not allow several combinations of (U-LV) that are difficult to define and are not present in the literature:

- (A-PS)+(U-LV) would require all outgoing updates to be identical. Given transitions on $\lambda(p_1)$ and $\mu(p_1, p_2)$, p_2 could not be assigned, since this update would not match the formal parameters of λ.
- (U-LV)+(G-NR) would permit circumventing the lack of register-to-register-comparisons by introducing "witness" parameters and using guards $(x_1 = p) \wedge (x_2 = p)$ to imply equality. Consequently, we also disallow (U-LV)+(G-UP), (U-LV)+(G-UA), and (U-LV)+(G-FG).

When discussing reductions between automata classes, a reduction might only be defined for a variant of a secondary feature and its supervariants. For example, a reduction might reduce (A-PT) to (A-PS), but will only be defined for automata that are at least (G-CC). A (A-PT)+(G-NR) automaton will be reduced to a (A-PS)+(G-NR) one, while a (A-PT)+(G-CC) automaton will yield a (A-PS)+(G-CC) one. We denote a reduction that requires a secondary feature and preserves it in the transformed automaton, as (A-PT)+(G-NR)$^{\sqsubseteq}$ to (A-PS)+(G-NR)$^{\sqsubseteq}$.

4.2 Classification of Some Existing Models

This taxonomy can now be applied to models from the literature. The descriptions are summarized in Table 2. For some of these models, complexity results are present in the literature and are recapped below.

Finite-memory Automata. Kaminsky and Francez were the first to define a register automaton model [15]. They defined update-or-present semantics, enforced identical updates per state and initialization with empty registers and did not use labels. Bojańczyk et al. previously proved these automata to be equivalent in expressiveness to G-automata [3]. The model is characterized as (A-PS)+(U-UL)+(R-IE)+(G-UP).

Figure 4b shows a sample finite-memory automaton accepting the language D (i.e., all single-symbol words). Note that the transition from q_1 to q_2 is unusable since x_1 always has value # in q_1.

Automaton Model		Variant
Initialized finite-memory automata	[17,18]	(A-PS)+(U-UL)+(R-IN)+(G-UP)
Finite-memory automata	[15]	(A-PS)+(U-UL)+(R-IE)+(G-UP)
Neven-Schwentick-Vianu automata	[19]	(A-PT)+(U-UL)+(R-IE)+(G-UA)
Segoufin automata	[23]	(A-PT)+(U-LU)+(R-IN)+(G-FG)
Demri-Lazić automata	[11]	(A-PT)+(U-LU)+(R-IN)+(G-NR)
Succinct canonical register automata	[5,6]	(A-PT)+(U-LV)+(R-UA)+(G-CC)

Table 2. Taxonomy of automata models.

Lemma 1 ([22, Theorem 1]). *Membership of words in deterministic finite-memory automata is* P*-complete.*

Lemma 2 ([22, Theorem 2]). *Membership of words in finite-memory automata is* NP*-complete.*

Lemma 3 ([22, Theorem 4]). *NonEmptiness of finite-memory automata is* NP*-complete.*

Initialized Finite-memory Automata. Murawski et al. remarked on a variant of finite-memory automata [17,18] in which all registers are initialized with data values (i.e., no empty value # is used). The model is characterized as (A-PS)+(U-UL)+(R-IN)+(G-UP).

Figure 4a shows a sample initialized finite-memory automaton accepting the language

$$\big(D \setminus \{\mathtt{a}\}\big) \cup \{\mathtt{ba}\}.$$

In contrast to the automaton in Fig. 4b, all transitions are usable.

Lemma 4 ([17, Footnote 5]). *NonEmptiness of initialized finite-memory automata is* NL*-complete.*

Neven-Schwentick-Vianu Automata. Neven et al. presented a slight modification of finite-memory automata [19]. Their model additionally permits ε-transitions, i.e., transitions that do not consume input values. Additionally, it requires every input word to start with a designated start symbol. We propose this theorem, the proof of which is outside the scope of this paper:

Proposition 1. *Every automaton satisfying the model by Neven et al. can be transformed into a equivalent* (A-PT)+(U-UL)+(R-IE)+(G-UA) *automaton using our notation.*

Figure 4c shows a sample (transformed) Neven-Schwentick-Vianu automaton accepting the language

$$\big(D \setminus \{\mathtt{b}\}\big) \cup \big\{dd \mid d \in D \setminus \{\mathtt{b}\}\big\}.$$

In comparison with Fig. 4b's automaton, different assignment targets for transitions with the same origin are permitted and guards for assignments must compare the value to the target register.

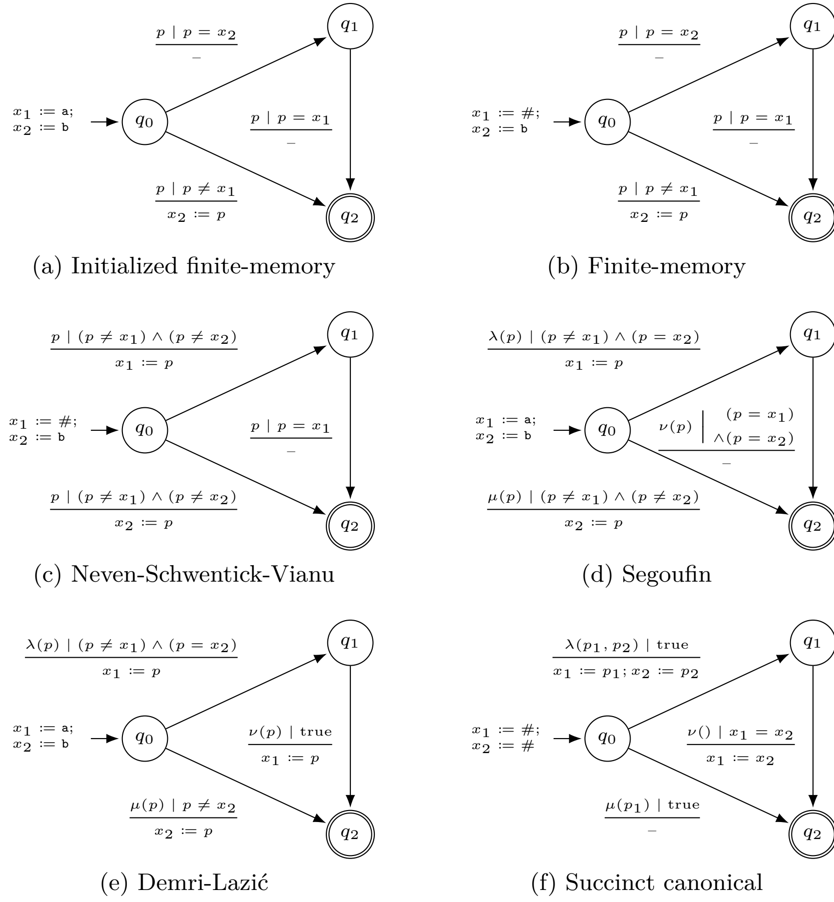


Fig. 4. Sample automata demonstrating the expressiveness of existing models.

Segoufin Automata. Segoufin's automaton model [23] extends finite-memory automata with per-transition updates and labels, does forbid empty registers and defines single update with full test semantics. The model is characterized as (A-PT)+(U-LU)+(R-IN)+(G-FG).

Figure 4d shows a sample Segoufin automaton accepting the language

$$\big\{\mu(d) \mid d \in D \setminus \{\mathtt{a}, \mathtt{b}\}\big\} \cup \{\lambda(\mathtt{b})\nu(\mathtt{b})\}.$$

In contrast to the automaton shown in Fig. 4c, #-intialized registers are not permitted. Guards and assignments can be used in arbitrary combinations, but guards must compare p to every register. This permits indirect register-to-register comparisons using witness parameters, as exemplified by the q_1-q_2-transition's guard. In addition, data values are labeled under this model.

Demri-Lazić Automata. Demri and Lazić defined an automaton model [11] for use in acceptance games. The model as-defined does not match this taxonomy, but we again propose a theorem:

Proposition 2. *Every automaton satisfying the model by Demri and Lazić can be transformed into a equivalent* (A-PT)+(U-LU)+(R-IN)+(G-NR) *automaton using our notation.*

Figure 4e shows a sample Demri-Lazić automaton accepting the language

$$\big\{\mu(d) \mid d \in D \setminus \{\mathsf{b}\}\big\} \cup \big\{\lambda(\mathsf{b})\nu(d) \mid d \in D\big\}.$$

When comparing this to the automaton in Fig. 4d, it can be seen that the permissible guard statements do not need to compare every register.

Lemma 5 ([11, Theorem 5.1(a)]). *NonEmptiness of Demri-Lazić automata is* PSPACE*-complete.*

Succinct Canonical Register Automata. Succinct canonical RAs [5] use variadic labeled data, update-activated registers instead of initialized ones and allow conjunctions of arbitrary comparisons in their guards. This model is characterized as (A-PT)+(U-LV)+(R-UA)+(G-CC).

Figure 4f shows a sample succinct canonical register automaton accepting the language

$$\big\{\mu(d) \mid d \in D \setminus \{\mathsf{b}\}\big\} \cup \big\{\lambda(d,d)\nu() \mid d \in D\big\}.$$

Note that in contrast to the other models such as the automaton in Fig. 4e, guards can compare registers, i.e., the availability of the q_1-q_2 transition depends on the values of p_1 and p_2 in the q_0-q_1-transition. Additionally, registers can be #-initialized, but must be written to before reading. Data values can have arbitrary arity, including arity zero.

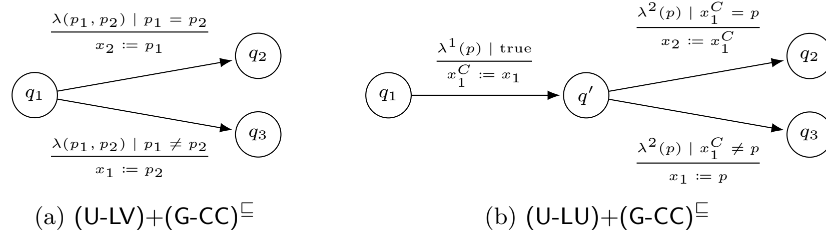
5 Reductions between Variants

In this section, we will examine reductions between the RA variants. While all variants have equal expressiveness, two lower complexity bounds for reductions as well as the non-existence of a specific reduction type are proven, outlining differences between the variants. This section considers each automaton feature in turn.

5.1 Data Universe Type

Theorem 1. *There exists a determinism-preserving* P*-prefix free reduction from* (U-LV)+(G-CC)$^{\sqsubseteq}$ *to* (U-LU)+(G-CC)$^{\sqsubseteq}$*.*

Note that (U-LV)+(G-CC)$^{\sqsubseteq}$ = (U-LV), since (U-LV) is only defined for (G-CC). Intuitively, this reduction replaces each symbol of arity k with k symbols of arity one and modifies the automaton accordingly.



(a) (U-LV)+(G-CC)$^{\sqsubseteq}$ (b) (U-LU)+(G-CC)$^{\sqsubseteq}$

Fig. 5. Sample transformation from two (U-LV)+(G-CC)$^{\sqsubseteq}$ transitions to multiple (U-LU)+(G-CC)$^{\sqsubseteq}$ transitions.

Proof Sketch. We construct a new input language in which we replace every label λ with $\mathfrak{a}(\lambda)$ labels $\lambda^1, \ldots, \lambda^{\mathfrak{a}(\lambda)}$. The data reduction then replaces every instance of $\lambda(\vec{d})$ with $\lambda^1(d_1) \ldots \lambda^{\mathfrak{a}(\lambda)}(d_{\mathfrak{a}(\lambda)})$. The automaton reduction modifies each state's outgoing transitions. For every label present, $\mathfrak{a}(\lambda)$ transitions to intermediate states are created. Since guards and updates can only be safely evaluated in the last transition, these transition only store their parameters in *cache registers*. The automaton requires a total of $\max_{\lambda \in \Lambda} \mathfrak{a}(\lambda) - 1$ cache registers X^C. For each transition, a final step from the last intermediate state to the target state is generated during which guards and updates are evaluated, substituting cached values for the parameters.

For each word

$$\lambda_1(d_1, \ldots, d_{\mathfrak{a}(\lambda_1)}) \ldots \lambda_k(d_1, \ldots, d_{\mathfrak{a}(\lambda_k)}) \tag{LV}$$

accepted by the original, the word

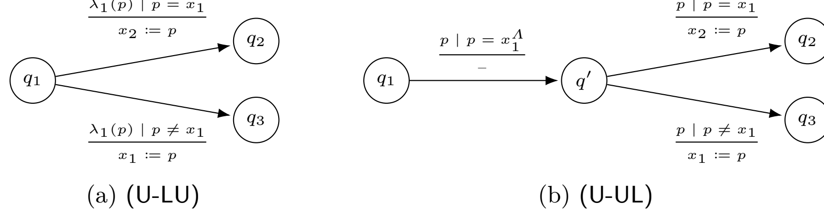
$$\lambda_1^1(d_1) \ldots \lambda_1^{\mathfrak{a}(\lambda_1)}(d_{\mathfrak{a}(\lambda_1)}) \ldots \lambda_k^1(d_1) \ldots \lambda_l^{\mathfrak{a}(\lambda_k)}(d_{\mathfrak{a}(\lambda_k)}) \tag{LU}$$

is accepted by the newly created automaton. The construction ensures that each word accepted by the new automaton is of form LU, i.e., for each original label, all partial labels are present in the word in correct order. Such words can be reassembled into an input of form LV accepted by the original automaton, preserving acceptance behavior. □

The process is exemplified in Fig. 5. Note that in the example, the intermediate state q' is shared between transitions to preserve determinism.

Theorem 2. *There exists a determinism-preserving* P*-many-one reduction from* (U-LU) *to* (U-UL).

Proof Sketch. The reduction designates data values as proxies for labels and alternatingly reads a proxy and a "real" value. We require $|\Lambda|$ data symbols as *label proxies*. These proxies are stored in additional registers X^Λ, with λ_i being replaced by x_i^Λ. The input word $\lambda_1(d)\lambda_2(e)$ would then be replaced with $x_1^\Lambda d x_2^\Lambda e$.



(a) (U-LU) (b) (U-UL)

Fig. 6. Sample transformation from two (U-LU) transitions to multiple (U-UL) transitions.

The proxy values are then added as a prefix and are assigned to the registers during an initialization before the first original transition. They must be selected to differ from any value in the input so that (G-UP) and (G-UA) semantics are retained, requiring access to both input and automaton.

For each label Λ_i present on a state's outgoing transitions, the automaton reduction creates a transition with guard $p = x_i^\Lambda$ and no assignment to an intermediate state similar to Theorem 1. For each original transition, a second transition from the matching intermediate state is created that uses the original guard, update, and target. □

If the automaton permits duplicate values in registers ((G-FG) and above), the proxies can be chosen at random instead, yielding a narrower reduction type.

Corollary 1. *There exists a determinism-preserving* P*-linear-local reduction from* (U-LU)+(G-FG)$^\sqsubseteq$ *to* (U-UL)+(G-FG)$^\sqsubseteq$*.*

An example of the transformation is shown in Fig. 6. As with the last example, the intermediate state q' is shared between transitions to preserve determinism.

All results presented in this section are outlined in Fig. 11a.

5.2 Register Availability

Theorem 3. *There exists a determinism-preserving* LIN*-automaton-only reduction from* (R-UA) *to* (R-IN)*.*

Proof Sketch. The automaton-only reduction sets $X_Q(q) := X$ for all $q \in Q$, i.e., all registers are always visible. All updates are extended to assign the newly visible registers to themselves and the initial valuation is set to random values. Since these values are guaranteed to be overwritten before being accessed by a guard, this does not change the automaton's semantics. □

Theorem 4. *There exists a determinism-preserving* LIN*-data stable reduction from* (R-IE)+(G-FG)$^\sqsubseteq$ *to* (R-IN)+(G-FG)$^\sqsubseteq$*.*

Proof Sketch. We employ the proxy value technique presented in Theorem 2. The automaton reduction inserts an initial transition that reads a proxy value for # that is distinct from all non-# initial values and stores it in every #-initialized register and a new register, $x^{\#}$. If the proxy value is encountered in the input, the automaton's semantics could change. To preserve NonEmptiness, we ensure that every such input is rejected by modifying every guard g to $g \wedge (p_1 \neq x^{\#}) \wedge \cdots \wedge (p_{\mathfrak{a}(\lambda)} \neq x^{\#})$. The data prefix-generating reduction selects a proxy value from the data value set that does not occur in the remaining input. □

We can demonstrate that under common assumptions about complexity classes, more space-efficient reductions do not exist.

Theorem 5. *If* $\mathsf{NL} \neq \mathsf{NP}$*, there exists no* NL*-NonEmptiness-Turing reduction from* (R-IE) *to* (R-IN)*.*

Proof. We demonstrate that the existence of such a reduction permits the creation of an NL algorithm for an NP-complete problem. Assume that an NL-Turing reduction from (R-IE) to (R-IN) exists. Since NonEmptiness of initialized finite-memory automata can be decided in NL, we obtain an $\mathsf{NL}^{\mathsf{NL}}$ algorithm for NonEmptiness of finite-memory automata. Since the original problem is NP-complete and $\mathsf{NL}^{\mathsf{NL}} = \mathsf{NL}$ due to $\mathsf{NL} = \mathsf{coNL}$, $\mathsf{NL} = \mathsf{NP}$. □

Theorem 6. *There exists a determinism-preserving* LIN*-data stable reduction from* (R-IN) *to* (R-UA)*.*

Proof Sketch. Again, we employ a proxy value technique to substitute values for the initialization. We then use an existing result to demonstrate the automaton's emptiness is unchanged. The automaton reduction inserts initial steps that reads $|X|$ *proxy values* and assigns them to the correct registers, ensuring that values that were equal in the initial valuation remain so. The automaton prefix-generating reduction can write the original initialization to maintain the original behavior.

We demonstrate that this reduction preserves NonEmptiness. For each word accepted by the original automaton, the proxy values can be set to the original initialization to create an accepting input. For the inverse direction, consider a word accepted by the new automaton. It consists of a prefix of length $|X|$ that is used to initialize the registers and a remaining input word. We define an automorphism on data values that maps the prefix's values to the source automaton's initialization. Since a register automaton's language is closed under automorphisms on the data value set [15, Proposition 2], the new automaton will accept the resulting word. By definition, this word must also have been accepted by the original automaton. □

All results presented in this section are outlined in Fig. 11b.

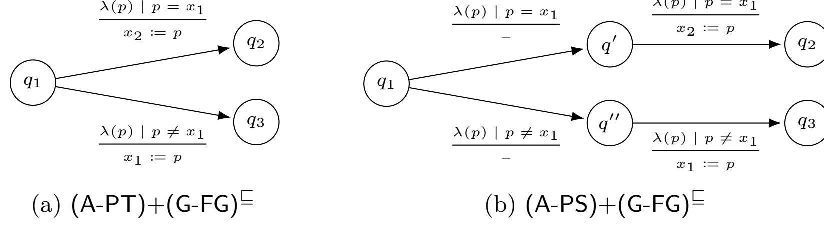


Fig. 7. Sample transformation from two (A-PT)+(G-FG)$^{\sqsubseteq}$ transitions to multiple (A-PS)+(G-FG)$^{\sqsubseteq}$ transitions.

5.3 Update Granularity

Theorem 7. *There exists a determinism-preserving* LIN*-prefix free reduction from* (A-PT)+(G-FG)$^{\sqsubseteq}$ *to* (A-PS)+(G-FG)$^{\sqsubseteq}$.

Proof Sketch. This reduction requires duplication of every input symbol. The first instance is used to make a transition to an intermediate state, while the second is used in the assignment.

The automaton reduction modifies all transitions. Given a transition

$$\frac{\lambda(p_1, \ldots, p_{\mathfrak{a}(\lambda)}) \mid g}{a},$$

it introduces an intermediate state. The transition from intermediate to target state is identical to the original and the transition from source to intermediate state is

$$\frac{\lambda(p_1, \ldots, p_{\mathfrak{a}(\lambda)}) \mid g}{-}.$$

Since all guards remain identical, determinism is preserved.

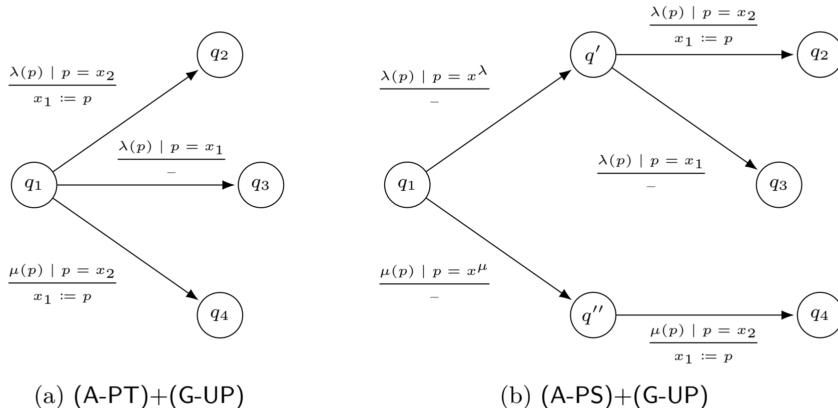
The data reduction duplicates every data symbol in the input word. Due to the structure of the new automaton, an accepted word can be transformed to an accepting word for the original by removing all symbols in odd positions, preserving emptiness. □

An example is shown in Fig. 7. For other guard-update models, the reduction needs to be modified slightly.

Theorem 8. *There exists a determinism-preserving* LIN*-prefix free reduction from* (A-PT)+(G-UP) *to* (A-PS)+(G-UP) *and from* (A-PT)+(G-UA) *to* (A-PS)+(G-UA).

Proof Sketch. Due to the limited types of transitions available, the technique used in the last proof needs to be adapted to these classes.

We alter the automaton by adding $|\Lambda|$ additional *scratch registers* X^Λ to the automaton and initializing it to data values D^Λ not present in the input. The



(a) (A-PT)+(G-UP) (b) (A-PS)+(G-UP)

Fig. 8. Sample transformation from two (A-PT)+(G-UP) transitions to multiple (A-PS)+(G-UP) transitions. The transitions to q_2 and q_3 both used label λ and now share the intermediate state q'. The transition to q_4 was labeled μ and is reached over a separate state q''.

reduction then introduces intermediate states similar to the proof of Theorem 7. However, we use the transition

$$\frac{\lambda(p) \mid p = x^\lambda}{-}$$

to transit from source to intermediate state. Since these transitions are mutually exclusive and transitions from the intermediate to the target state are copied as-is, determinism is preserved.

The data reduction inserts d_s before every data symbol in the input word. Again, an accepted word can be transformed to an accepting word for the original by removing all symbols in odd positions, preserving emptiness. □

An example for this variant (using (G-UP) semantics) is given in Fig. 8. Duplication or insertion of dummy symbols is required to efficiently perform the reduction. If the data language is untouched, no efficient algorithm exists:

Theorem 9. *There exists no determinism-preserving data stable reduction from* (A-PT) *to* (A-PS)*.*

Proof Sketch. Intuitively, a (A-PT)+(G-NR) automaton can store information by selecting an assignment's target register. A (A-PS)+(G-NR) automaton is forced to store the same information by transitioning to different states. This results in a superpolynomial amount of required states for certain languages.

We provide a (U-LU) language that can be recognized by a (A-PT) automaton of size k. Then, we prove by contradiction that a (A-PS) automaton must have $k!$ states to recognize the same language. The language uses labels $\lambda_1, \ldots, \lambda_\ell, \kappa_1, \ldots, \kappa_\ell$ to simulate an *ℓ-memory cell* storage as follows:

- Initially, all registers are empty.
- When reading $\lambda_k(p)$, the k-th memory cell is overwritten with p.
- When reading κ_k, the k-th memory cell is compared to p.
- The language is the set of all instruction sequences for which all κ-comparisons hold true.

A two-state deterministic (A-PT) automaton with k registers that implements memory cells using registers can be constructed for this language.

Now, we define the permutations $\pi : \{1, \dots, k\} \to \{1, \dots, k\}$ and the family of input strings $S^\pi \coloneqq \lambda_{\pi(1)}(p_1)\lambda_{\pi(2)}(p_2) \dots \lambda_{\pi(k)}(p_k)$ for distinct $p_0, \dots, p_k$. We now inductively show by contradiction that no two strings from this family can cause a (A-PS) automaton to enter the same state.

Assume that two such strings $S^\pi, S^{\pi'}$ exist, $\pi(1) \neq \pi'(1)$ and that the automaton enters the same state after both strings. Since the automaton started in the same state, p_1 must have been written to the same register x_1 and must not have been overwritten on any path (otherwise, $\kappa_{\pi(1)}$ and $\kappa_{\pi'(1)}$ cannot be handled).

Now, assume the automaton reads $\kappa_{\pi(1)}(p_1)$. It will accept after the input S^π and reject after $S^{\pi'}$. However, in that state, the guard

$$p = x_1 \wedge \left(\bigwedge_{x \in X_Q(q) \setminus \{x_1\}} p \neq x \right)$$

and all more general guards will be satisfied after both inputs, while all other (G-NR) guards will not be satisfied. Therefore, it must either accept or reject after both S^π and $S^{\pi'}$.

If we set $\pi(1) = \pi'(1)$, the argument can be repeated for the second input symbol. By induction, $\pi = \pi'$, i.e., no two different paths can merge. Since there are $k!$ permutations, we require at least as many states. □

All results presented in this section are outlined in Fig. 11c.

5.4 Guard-Update Model

Theorem 10. *There exists a determinism-preserving* LIN*-automaton-only reduction from* (G-UP) *to* (G-UA)*.*

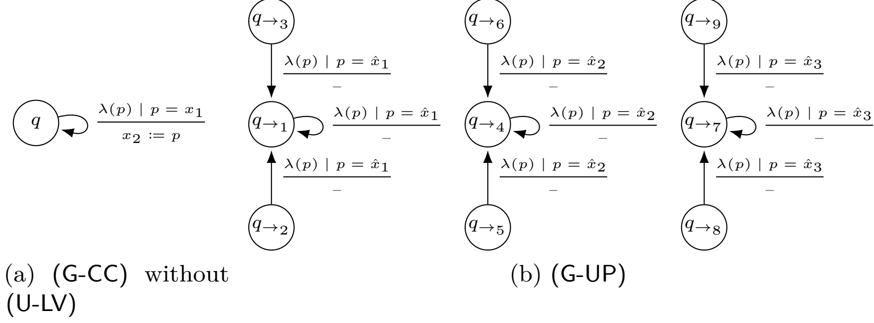
Proof Sketch. The reduction splits all transitions of form

$$\frac{\lambda(p) \mid \bigwedge_{x \in X_Q(q) \setminus \{x_i\}} (p \neq x)}{x_i \coloneqq p}$$

into two transitions with the same source and target states:

$$\frac{\lambda(p) \mid \bigwedge_{x \in X_Q(q)} (p \neq x)}{x_i \coloneqq p} \text{ and } \frac{\lambda(p) \mid p = x_i}{-},$$

which are equivalent to the original and compatible with (G-UA). Since the transitions are mutually exclusive, the reduction preserves determinism. □



(a) (G-CC) without (U-LV)

(b) (G-UP)

Fig. 9. Sample transformation from a (G-CC) without (U-LV) transition to (G-UP) transitions. For this example, the set of mapping functions are defined as $\to_i := \left\langle x_1 = \hat{x}_j, q_2 = \hat{x}_k \mid j = (i \bmod 3) + 1, k = \left\lfloor \frac{i-1}{3} \right\rfloor + 1 \right\rangle$.

Theorem 11. *There exists a determinism-preserving* P*-automaton-only reduction from* (G-UA) *to* (G-FG)*.*

Proof Sketch. The automaton reduction extends all guards to add the missing comparisons to registers. While this would normally result in an exponential number of transitions to describe all possible comparisons of parameters and registers, (G-UA) guarantees that a parameter can be equal to at most one register. A transition

$$\frac{\lambda(p) \mid \bigwedge_{x \in X_Q(q)} (p \neq x)}{a}$$

already satisfies (G-FG), while one of form

$$\frac{\lambda(p) \mid p = x_i}{-}$$

is equivalent to

$$\frac{\lambda(p) \mid p = x_i \wedge \bigwedge_{x \in X_Q(q) \setminus \{x_i\}} (p \neq x)}{-}.$$

□

Theorem 12. *There exists a determinism-preserving* EXP*-automaton-only reduction from* (G-CC) *without* (U-LV) *to* (G-UP)*.*

Proof Sketch. The construction circumvents the lack of register-to-register operations by virtualizing the automaton's registers. Each state of the resulting automaton is associated with a register mapping, making register-to-register operations essentially "free".

To prepare the transformation, we add an additional register to the automaton, resulting in the register set $\hat{X}$ and define a family of functions $f_{\to} : X \to \hat{X}$ that defines the register mapping. There is a maximum of $|X|^{|X|+1}$ such functions,

resulting in an exponential blow-up. For each function $f_\rightarrow$, we create a copy of the automaton's states. A state q associated with the mapping $f_\rightarrow$ is called $q_\rightarrow$. This permits conclusions about equality between registers. Two registers x_i and x_j are equal if and only if $f_\rightarrow(x_i) = f_\rightarrow(x_j)$. A register-to-register assignment becomes a change in storage: $x_i \coloneqq x_j$ is modeled as $f_\rightarrow(x_i) \coloneqq f_\rightarrow(x_j)$. Additionally, for each mapping function, a *scratch register* $\hat{x}_s$ is identified that is not in $\Im(f_\rightarrow)$. This register becomes the sole target of write operations. The initial state remains in the copy where $f_\rightarrow$ is the identity function.

Each source automaton's transition $\langle q, q', \lambda, g, u\rangle$ now needs to be translated to account for the register virtualization. First, a copy of the transition is created for each copy of its source, i.e., for every mapping $f_\rightarrow$. Next, we extend the guard clause for each resulting copy, to ensure that the parameter p is compared to every register in the original automaton. We refer to the registers that p is not compared to as $\bar{X}$. Now, we generate guards

$$(g \wedge p = \bar{x}_1 \wedge p = \bar{x}_2 \wedge \cdots \wedge p = \bar{x}_{|\bar{X}|}), (g \wedge p \neq \bar{x}_1 \wedge p = \bar{x}_2 \wedge \cdots \wedge p = \bar{x}_{|\bar{X}|}),$$
$$(g \wedge p = \bar{x}_1 \wedge p \neq \bar{x}_2 \wedge \cdots \wedge p = \bar{x}_{|\bar{X}|}), (g \wedge p \neq \bar{x}_1 \wedge p \neq \bar{x}_2 \wedge \cdots \wedge p = \bar{x}_{|\bar{X}|}),$$
$$\ldots, (g \wedge p \neq \bar{x}_1 \wedge p \neq \bar{x}_2 \wedge \cdots \wedge p \neq \bar{x}_{|\bar{X}|})$$

and create a copy of the transition for each guard variant. We can now use the knowledge that $x_i = x_j \iff f_\rightarrow(x_i) = f_\rightarrow(x_j)$ to check if register-to-register comparisons are satisfied in the source state. If not, the transition copy is discarded. Otherwise, redundant comparisons can be omitted, resulting in either a single comparison $p = x_i$ or $\wedge_{x \in X}(p \neq x_i)$.

In the first case, we can generate transitions

$$\frac{\lambda(p) \mid p = f_\rightarrow(x_i)}{-}. \tag{EQ1}$$

In the second case, the guard is always satisfiable in the original automaton, but the value of p might still be stored in any register not $\in \Im(f_\rightarrow)$ from a previous write operation. For each register $\bar{x} \notin \Im(f_\rightarrow) \cup \{\hat{x}_s\}$, we create a transition

$$\frac{\lambda(p) \mid p = \bar{x}_i}{-}. \tag{EQ2}$$

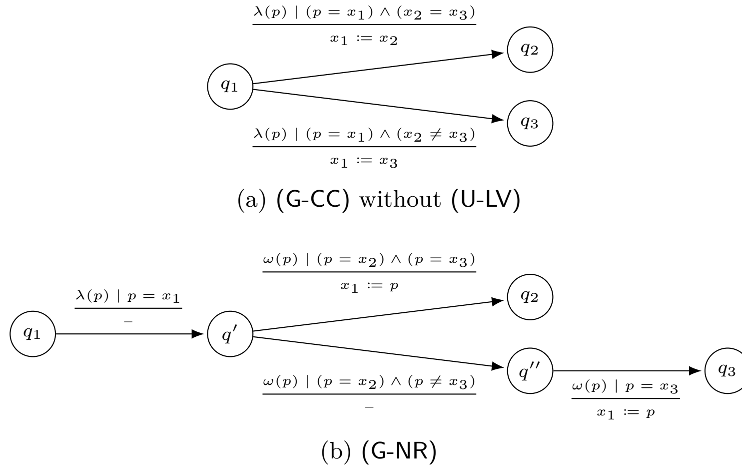
Finally, for the scratch register $\hat{x}_s$, we add the transition

$$\frac{\lambda(p) \mid \bigwedge_{\hat{x} \in \hat{X} \setminus \hat{x}_s}(p \neq \hat{x})}{\hat{x}_s \coloneqq p}. \tag{NEQ}$$

Since the generated transitions are mutually exclusive, the resulting automaton will be deterministic if the original automaton was.

Finally, the update is transformed into a new register mapping $f_{\rightarrow'}$. The transition's target is then set to $q'_{\rightarrow'}$.

- For all registers that are not explicitly written to by the update, $f_{\rightarrow'}$ behaves identically to $f_\rightarrow$.



(a) (G-CC) without (U-LV)

(b) (G-NR)

Fig. 10. Sample transformation from two (G-CC) without (U-LV) transitions to multiple (G-NR) transitions.

- For each register that is assigned another register's values using $x_j \coloneqq x_i$, the mapping function is modified such that $f_{\to'}(x_j) \coloneqq f_{\to}(x_i)$.
- Each register that is assigned a parameter using $x_j \coloneqq p$ must be handled differently for the three types of transitions. For EQ1-transitions, we can exploit that $p = x_i$, yielding $f_{\to'}(x_j) \coloneqq f_{\to}(x_i)$. For EQ2-transitions, we obtain $f_{\to'}(x_j) \coloneqq \bar{x}_i$. NEQ-transitions result in $f_{\to'}(x_j) \coloneqq \hat{x}_s$. □

A small example for the transformation of a transition with an equality test is given in Fig. 9. Note that the illustration assumes only two registers, for more, an even larger blow-up would result.

Theorem 13. *There exists a determinism-preserving* P*-many-one reduction from* (G-CC) *without* (U-LV) *to* (G-NR)*.*

Proof Sketch. For each transition, the reduction introduces up to $|X|^2$ "witness" values into the input to replace register-register operations, making the data reduction automaton dependent.

Given a transition with k register-to-register comparisons and ℓ register-to-register assignments, $k + \ell$ intermediate states are created. The original transition is stripped of all register-register operations. For each comparison $x_i = x_j$, an additional transition

$$\frac{\omega(p) \mid (p = x_i) \wedge (p = x_j)}{-}$$

is inserted and for each assignment $x_i \coloneqq x_j$, a transition

$$\frac{\omega(p) \mid p = x_j}{x_i \coloneqq p}$$

is created. The input is modified to include the “witness” values where required. To preserve determinism, intermediate states with identical incoming transitions can be merged. ☐

Figure 10 illustrates the process.

Theorem 14. *There exists a* P*-many-one reduction from* (G-NR) *to* (G-FG)*.*

Proof Sketch. The construction employs a technique similar to register virtualization used in the proof of Theorem 12. However, we store the equality information in registers and discard the actual values.

The reduction replaces the registers with $|X|^2 - |X|$ registers

$$x_{1,2}, \bar{x}_{1,2}, x_{1,3}, \bar{x}_{1,3}, \dots, x_{|X|-1,X}, \bar{x}_{|X|-1,X}$$

that encode the equality half-matrix of original registers. We will maintain the following invariants:

1. if in the original automaton for $i < j$, $x_i = x_j$, then $x_{i,j} = \bar{x}_{i,j}$,
2. if in the original automaton for $i < j$, $x_i \neq x_j$, then $x_{i,j} \neq \bar{x}_{i,j}$,
3. and for $i < j$ and i', j' with $i \neq i'$ or $j \neq j'$, $\{x_{i,j}, \bar{x}_{i,j}\} \cap \{x_{i',j'}, \bar{x}_{i',j'}\} = \emptyset$.

To avoid unnecessary concern for register order, we will also refer to $x_{i,j}$ as $x_{j,i}$. Using these registers, we can compare $x_i = x_j$ by using

$$\frac{\lambda(p) \mid (p = x_{i,j}) \wedge (p = \bar{x}_{i,j}) \wedge \bigwedge_{x \in X_Q(q) \setminus \{x_{i,j}, \bar{x}_{i,j}\}} (p \neq x)}{-}$$

and $x_i \neq x_j$ by using

$$\frac{\lambda(p) \mid (p = x_{i,j}) \wedge \bigwedge_{x \in X_Q(q) \setminus \{x_{i,j}\}} (p \neq x)}{-}.$$

Storing equalities needs to take into account the previous state. We store $x_i = x_j$ using two parallel transitions:

$$\frac{\lambda(p) \mid (p = x_{i,j}) \wedge (p = \bar{x}_{i,j}) \wedge \bigwedge_{x \in X_Q(q) \setminus \{x_{i,j}, \bar{x}_{i,j}\}} (p \neq x)}{-}$$

is used if the registers were previously equal and

$$\frac{\lambda(p) \mid (p = x_{i,j}) \wedge \bigwedge_{x \in X_Q(q) \setminus \{x_{i,j}\}} (p \neq x)}{\bar{x}_{i,j} \coloneqq p}$$

is used if they were not. Storing an inequality $x_i \neq x_j$ is possible in one transition:

$$\frac{\lambda(p) \mid \bigwedge_{x \in X_Q(q) \setminus \{x_{i,j}\}} (p \neq x)}{\bar{x}_{i,j} \coloneqq p}.$$

Now, all transitions need to be transformed. We distinguish two types of transitions, those that guarantee the equality of the parameter and a register

(e.g., $p = x_i$) and those that do not. In the first case, all other instances of p in the guard can be replaced with x_i. Since $p = x_i$ is always satisfiable, the guard can be replaced with test of equalities between registers. Each such comparison is done in a separate transition. The assignment is then done by updating all relevant equalities.

If p is only compared negatively to some registers, the guard is always satisfiable and can be removed. However, it is unknown if p is equal to registers it was not compared to. For each group of equal registers p might be equal to, we nondeterministically select either equality or inequality and write the corresponding information to the half-matrix. ◻

We now demonstrate that under widely-held assumptions about complexity classes, no efficient reduction between (G-NR) and (G-UP) can exist.

Theorem 15. *If* NP $\neq$ PSPACE*, there exists no* P*-many-one reduction from* (G-FG) *to* (G-UP)*.*

A similar result for deterministic (U-LV) automata has been proven by Cassel et al. [6]. Here, we demonstrate that such a reduction would allow us to construct an NP algorithm for a PSPACE-complete problem.

Proof. Assume that an NP-many-one reduction from (G-FG) to (G-UP) exists ($\star$). We can now construct an NP-algorithm for NonEmptiness of deterministic Demri-Lazić automata.

The following sequence of reductions reduces the Demri-Lazić automaton to a finite-memory automaton:

$$
\begin{aligned}
& \text{(A-PT)+(U-LU)+(R-IN)+(G-NR): Demri-Lazić automaton} \\
\overset{\text{Thm. 2}}{\preceq}\; & \text{(A-PT)+(U-UL)+(R-IN)+(G-NR)} \\
\overset{\text{Thm. 14}}{\preceq}\; & \text{(A-PT)+(U-UL)+(R-IN)+(G-FG)} \\
\overset{(\star)}{\preceq}\; & \text{(A-PT)+(U-UL)+(R-IN)+(G-UP)} \\
\sqsubseteq\; & \text{(A-PT)+(U-UL)+(R-IE)+(G-UP)} \\
\overset{\text{Thm. 8}}{\preceq}\; & \text{(A-PS)+(U-UL)+(R-IE)+(G-UP): finite-memory automaton}
\end{aligned}
$$

The finite-memory automaton's emptiness can be decided in NP; the result holds for the original automaton. Since the original problem is PSPACE-complete, NP = PSPACE. ◻

By using Turing reductions in the proof, permitting multiple oracle queries, we obtain statements conditional on the collapse of the polynomial hierarchy. These corollaries can be extended to arbitrary hierarchy levels.

Corollary 2. *If the polynomial hierarchy does not collapse, there exists no* PH*-NonEmptiness-Turing reduction from* (G-FG) *to* (G-UP)*.*

Theorem 16. *There exists a* LIN*-automaton-only reduction from* (G-UA) *to* (G-UP)*.*

Proof Sketch. For all transitions that store the parameter (e.g., using $x_i \coloneqq p$), the reduction removes $(p \neq x_i)$ from the guard, yielding a (G-UP) automaton. This operation can remove determinism from the automaton. To demonstrate that this does preserve NonEmptiness, consider an input that does "overwrite" a register with its value. This would not be permissible in the original automaton. We demonstrate that an accepted input must exist that does not overwrite a value.

Let the register valuation prior to overwriting be χ, the previous state be q and the overwritten value be $\bar{d}$. Consider a modification of the register automaton in which χ is the initial valuation and q the initial state. This register automaton's language is closed under automorphisms on the data value set [15, Proposition 2]. Let $\bar{d}'$ be a value not occurring in the remaining input and $\sigma : D \to D$ be the automorphism defined by

$$\sigma(d) = \begin{cases} \bar{d}' & \text{if } d = \bar{d} \\ \bar{d} & \text{if } d = \bar{d}' \\ d & \text{otherwise.} \end{cases}$$

If σ is applied to the remaining input, it is still accepted, but no overwriting occurs. This process can be repeated for each instance of overwriting. The resulting input is accepted by both the original and the newly created automaton. □

All results presented in this section are outlined in Fig. 11d.

6 Application to Existing Models

We now employ the feature-wise reductions from Section 5 to define reductions between the existing models from Section 4.2.

Theorem 17. *Every initialized finite-memory automaton is a valid finite-memory automaton.*

Theorem 18. *There exists a determinism-preserving* LIN*-automaton-only reduction from finite-memory to Neven-Schwentick-Vianu automata.*

Proof. We apply the following sequence of reductions:

$$\begin{aligned} & \text{(A-PS)+(U-UL)+(R-IE)+(G-UP): finite-memory automaton} \\ \sqsubseteq\ & \text{(A-PT)+(U-UL)+(R-IE)+(G-UP)} \\ \overset{\text{Thm. 10}}{\preceq}\ & \text{(A-PT)+(U-UL)+(R-IE)+(G-UA): Neven-Schwentick-Vianu automaton} \end{aligned}$$

□

Theorem 19. *There exists a determinism-preserving* P*-data stable reduction from Neven-Schwentick-Vianu to Segoufin automata.*

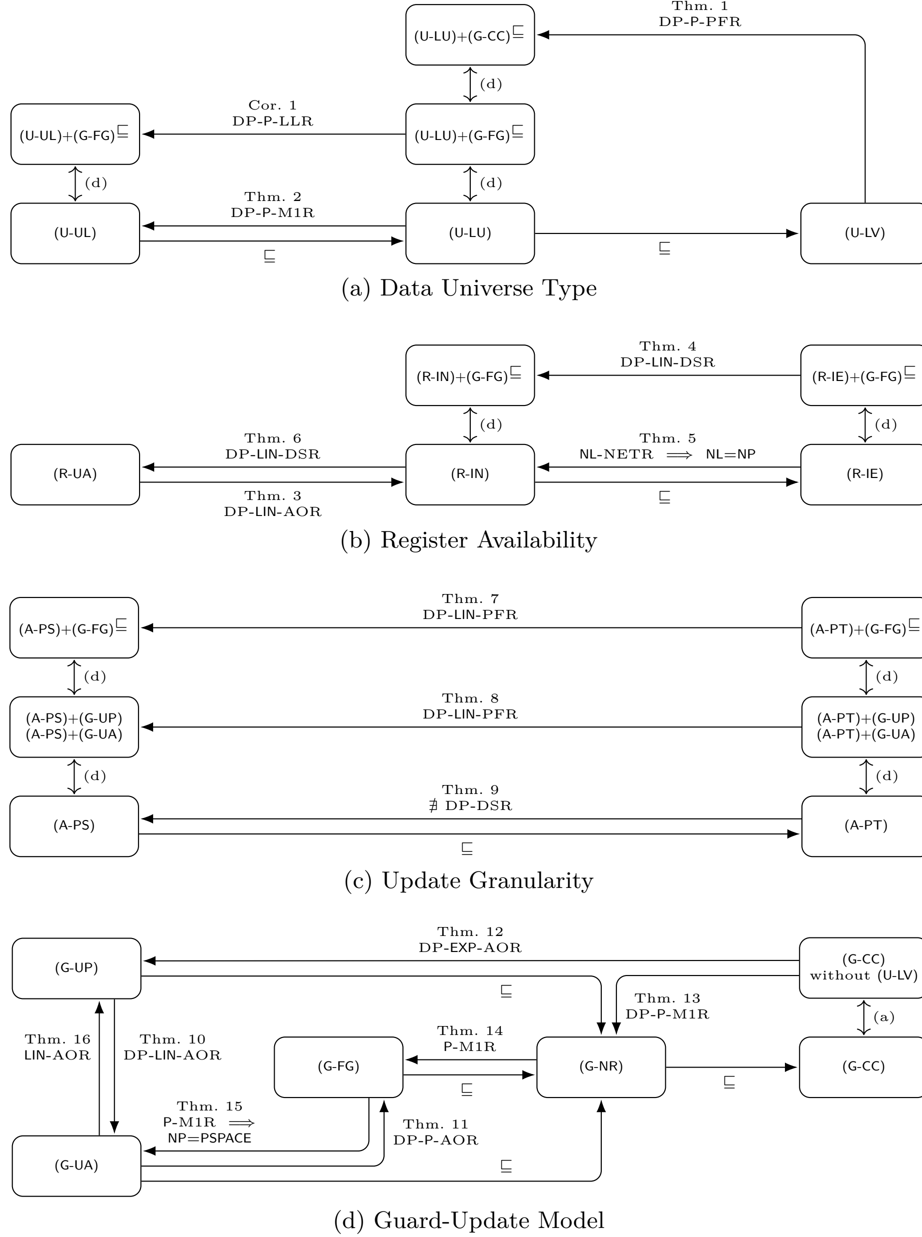


Fig. 11. Inequalities, reductions, and lower reduction complexity bounds between variants.

Proof. We apply the following sequence of reductions:

$$\begin{array}{cl} & \textsf{(A-PT)}+\textsf{(U-UL)}+\textsf{(R-IE)}+\textsf{(G-UA)}\text{: Neven-Schwentick-Vianu automaton} \\ \sqsubseteq & \textsf{(A-PT)}+\textsf{(U-LU)}+\textsf{(R-IE)}+\textsf{(G-UA)} \\ \overset{\text{Thm. 11}}{\preceq} & \textsf{(A-PT)}+\textsf{(U-LU)}+\textsf{(R-IE)}+\textsf{(G-FG)} \\ \overset{\text{Thm. 4}}{\preceq} & \textsf{(A-PT)}+\textsf{(U-LU)}+\textsf{(R-IN)}+\textsf{(G-FG)}\text{: Segoufin automaton} \end{array}$$

□

Theorem 20. *Every Segoufin automaton is a valid Demri-Lazić automaton.*

Theorem 21. *There exists a determinism-preserving* LIN*-data stable reduction from Demri-Lazić to succinct canonical register automata.*

Proof. We apply the following sequence of reductions:

$$\begin{array}{cl} & \textsf{(A-PT)}+\textsf{(U-LU)}+\textsf{(R-IN)}+\textsf{(G-NR)}\text{: Demri-Lazić automaton} \\ \overset{\text{Thm. 6}}{\preceq} & \textsf{(A-PT)}+\textsf{(U-LU)}+\textsf{(R-UA)}+\textsf{(G-NR)} \\ \sqsubseteq & \textsf{(A-PT)}+\textsf{(U-LU)}+\textsf{(R-UA)}+\textsf{(G-CC)} \\ \sqsubseteq & \textsf{(A-PT)}+\textsf{(U-LV)}+\textsf{(R-UA)}+\textsf{(G-CC)}\text{: succinct canonical RA} \end{array}$$

□

Theorem 22. *There exists a determinism-preserving* P*-many-one reduction from succinct canonical register automata to Demri-Lazić automata.*

Proof. We apply the following sequence of reductions:

$$\begin{array}{cl} & \textsf{(A-PT)}+\textsf{(U-LV)}+\textsf{(R-UA)}+\textsf{(G-CC)}\text{: succinct canonical RA} \\ \overset{\text{Thm. 1}}{\preceq} & \textsf{(A-PT)}+\textsf{(U-LU)}+\textsf{(R-UA)}+\textsf{(G-CC)} \\ \overset{\text{Thm. 3}}{\preceq} & \textsf{(A-PT)}+\textsf{(U-LU)}+\textsf{(R-IN)}+\textsf{(G-CC)} \\ \overset{\text{Thm. 13}}{\preceq} & \textsf{(A-PT)}+\textsf{(U-LU)}+\textsf{(R-IN)}+\textsf{(G-NR)}\text{: Demri-Lazić automaton} \end{array}$$

□

Theorem 23. *There exists a determinism-preserving* EXP*-many-one reduction from succinct canonical register automata to finite-memory automata.*

Proof. We apply the following sequence of reductions:

$$\begin{array}{cl} & \textsf{(A-PT)}+\textsf{(U-LV)}+\textsf{(R-UA)}+\textsf{(G-CC)}\text{: succinct canonical RA} \\ \overset{\text{Thm. 7}}{\preceq} & \textsf{(A-PS)}+\textsf{(U-LV)}+\textsf{(R-UA)}+\textsf{(G-CC)} \\ \overset{\text{Thm. 1}}{\preceq} & \textsf{(A-PS)}+\textsf{(U-LU)}+\textsf{(R-UA)}+\textsf{(G-CC)} \\ \overset{\text{Thm. 2}}{\preceq} & \textsf{(A-PS)}+\textsf{(U-UL)}+\textsf{(R-UA)}+\textsf{(G-CC)} \\ \overset{\text{Thm. 3}}{\preceq} & \textsf{(A-PS)}+\textsf{(U-UL)}+\textsf{(R-IN)}+\textsf{(G-CC)} \\ \sqsubseteq & \textsf{(A-PS)}+\textsf{(U-UL)}+\textsf{(R-IE)}+\textsf{(G-CC)} \\ \overset{\text{Thm. 12}}{\preceq} & \textsf{(A-PS)}+\textsf{(U-UL)}+\textsf{(R-IE)}+\textsf{(G-UP)}\text{: finite-memory automaton} \end{array}$$

□

Theorem 24. *There exists a* P*-many-one reduction from Demri-Lazić to Segoufin automata.*

Proof. Follows from Theorem 14.

Theorem 25. *If the polynomial hierarchy does not collapse, there exists no* PH*-NonEmptiness-Turing reduction from Segoufin to Neven-Schwentick-Vianu automata.*

Proof. Follows from Corollary 2.

Theorem 26. *There exists a* LIN*-prefix free reduction from Neven-Schwentick-Vianu to finite-memory automata.*

Proof. We apply the following sequence of reductions:

$$
\begin{aligned}
& \text{(A-PT)+(U-UL)+(R-IE)+(G-UA): Neven-Schwentick-Vianu automaton} \\
\overset{\text{Thm. 8}}{\preceq}\ & \text{(A-PS)+(U-UL)+(R-IE)+(G-UA)} \\
\overset{\text{Thm. 16}}{\preceq}\ & \text{(A-PS)+(U-UL)+(R-IE)+(G-UP)} \\
& \text{(A-PS)+(U-UL)+(R-IE)+(G-UP): finite-memory automaton}
\end{aligned}
$$

□

Theorem 27. *If* $\mathsf{NL} \neq \mathsf{NP}$*, there exists no* NL*-NonEmptiness-Turing reduction from finite-memory to initialized finite-memory automata.*

Proof. Follows from Theorem 5.

Three categories of model can be distinguished by the complexity of deciding NonEmptiness: those for which the problem is NL-, NP-, and PSPACE-complete. These match the "register disciplines" *SF*, $S\#_0$, and *MF* by Murawski et al. [17,18]. The resulting structure is shown in Fig. 12.

For finite-memory automata and above, deciding Membership is P-complete for deterministic automata and NP-complete otherwise. P- and NP-hardness were proven for finite-memory automata. For every model, the Membership of a word can – depending on determinism – be verified in P or NP by "executing" the automaton. Reductions therefore are of little interest for deciding the Membership problem.

7 Conclusion and Future Work

We have described a taxonomy for several register automaton features and successfully applied it to several types of automaton in the literature. The examined feature variants have been shown to be mutually reducible, as outlined in Fig. 11. This shows that all variants have identical expressiveness. We also charted the complexity of the NonEmptiness problem for different features and identified three categories of automaton, those for which it is NL-, NP-, and PSPACE-complete. The possibility of transition guards to be unsatisfiable for certain

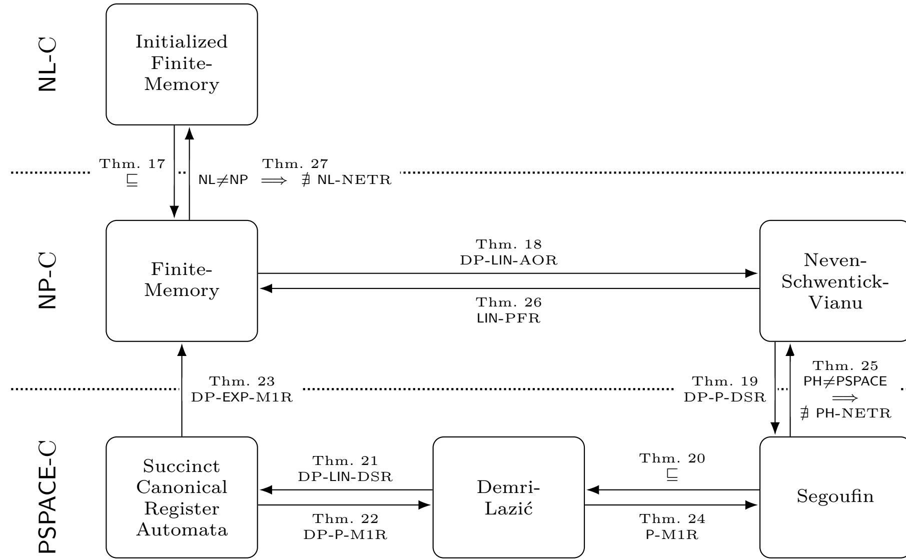


Fig. 12. Reductions between models and the complexity of NonEmptiness.

register valuations defines the difference between the first two, while the ability to store the same value in multiple registers defining the difference between the latter[1]. This implies that the size of automaton required to recognize a language varies between models, i.e., automata with PSPACE-complete NonEmptiness require less size to recognize a language.

Some register automaton formalisms, such as M-automata [15] and the automata defined by Benedikt et al. [2] cannot be described using our taxonomy. The former bears more similarity to pebble automata [19], while the latter's use of states is dissimilar to any other model's. In future work, our taxonomy could be extended to capture these formalisms.

Semantic extensions that strictly increase expressiveness such as register pushdown automata [9], fresh-register automata [24], register automata with non-deterministic reassignment [16] or with linear arithmetic [8] and symbolic register automata [10] have been proposed as extensions to the classical register automaton model studied by us. Again, these extensions could be taxonomized to permit the transfer of applicable results.

References

1. Babari, P., Droste, M., Perevoshchikov, V.: Weighted register automata and weighted logic on data words. In: Sampaio, A., Wang, F. (eds.) Theoretical Aspects of Computing – ICTAC 2016. pp. 370–384. No. 9965 in LNCS, Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-46750-4_21

[1] Segoufin mistakenly attributes this to the presence of labels in [23].

2. Benedikt, M., Ley, C., Puppis, G.: What you must remember when processing data words. In: Laender, A.H.F., Lakshmanan, L.V.S. (eds.) Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management. pp. 11.1–11.8. No. 619 in CEUR Workshop Proceedings, CEUR-WS.org, Aachen (2010), http://ceur-ws.org/Vol-619/paper11.pdf
3. Bojańczyk, M., Klin, B., Lasota, S.: Automata theory in nominal sets. Log. Methods Comput. Sci. **10**(4), 1–44 (Aug 2014). https://doi.org/10.2168/LMCS-10(3:4)2014
4. Cassel, S., Howar, F., Jonsson, B.: RALib: A LearnLib extension for inferring EFSMs. In: Proceedings of the 4th International Workshop on Design and Implementation of Formal Tools and Systems (2015), https://www.faculty.ece.vt.edu/chaowang/difts2015/papers/paper_5.pdf
5. Cassel, S., Howar, F., Jonsson, B., Merten, M., Steffen, B.: A succinct canonical register automaton model. In: Bultan, T., Hsiung, P.A. (eds.) Automated Technology for Verification and Analysis. pp. 366–380. No. 6996 in LNCS, Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24372-1_26
6. Cassel, S., Howar, F., Jonsson, B., Merten, M., Steffen, B.: A succinct canonical register automaton model. J. Log. Algebr. Methods Program. **84**(1), 54–66 (2015). https://doi.org/10.1016/j.jlamp.2014.07.004
7. Cassel, S., Howar, F., Jonsson, B., Steffen, B.: Active learning for extended finite state machines. Form. Asp. Comp. **28**, 233–263 (Apr 2016). https://doi.org/10.1007/s00165-016-0355-5
8. Chen, Y.F., Lengál, O., Tan, T., Wu, Z.: Register automata with linear arithmetic. In: 2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). pp. 1–12. IEEE (Jun 2017). https://doi.org/10.1109/LICS.2017.8005111
9. Cheng, E.Y.C., Kaminski, M.: Context-free languages over infinite alphabets. Acta Inform. **35**(3), 245–267 (Mar 1998). https://doi.org/10.1007/s002360050120
10. D'Antoni, L., Ferreira, T., Sammartino, M., Silva, A.: Symbolic register automata. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 3–21. No. 11561 in LNCS, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_1
11. Demri, S., Lazić, R.: LTL with the freeze quantifier and register automata. ACM Trans. Comput. Log. **10**(3), 16:1–16:30 (Apr 2009). https://doi.org/10.1145/1507244.1507246
12. Howar, F.: Active learning of interface programs. Ph.D. thesis, Technische Universität Dortmund (Jun 2012). https://doi.org/10.17877/DE290R-4817
13. Isberner, M., Howar, F., Steffen, B.: Learning register automata: from languages to program structures. Mach. Learn. **96**, 65–98 (Jul 2014). https://doi.org/10.1007/s10994-013-5419-7
14. Isberner, M., Howar, F., Steffen, B.: The open-source LearnLib. In: Kroening, D., Păsăreanu, C.S. (eds.) Computer Aided Verification. pp. 487–495. No. 9206 in LNCS, Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_32
15. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. **134**(2), 329–363 (1994). https://doi.org/10.1016/0304-3975(94)90242-9
16. Kaminski, M., Zeitlin, D.: Finite-memory automata with non-deterministic reassignment. Int. J. Found. Comput. Sci. **21**(05), 741–760 (2010). https://doi.org/10.1142/S0129054110007532
17. Murawski, A.S., Ramsay, S.J., Tzevelekos, N.: Reachability in pushdown register automata. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) Mathematical Foundations of Computer Science 2014. pp. 464–473. No. 8634 in LNCS, Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44522-8_39

18. Murawski, A.S., Ramsay, S.J., Tzevelekos, N.: Reachability in pushdown register automata. J. Comput. Syst. Sci **87**, 58–83 (2017). https://doi.org/10.1016/j.jcss.2017.02.008
19. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Logic **5**(3), 403–435 (Jul 2004). https://doi.org/10.1145/1013560.1013562
20. Post, E.L.: Recursively enumerable sets of positive integers and their decision problems. Bull. Amer. Math. Soc. **50**(5), 284–316 (1944). https://doi.org/10.1090/S0002-9904-1944-08111-1
21. Saint-Andre, P.: Extensible messaging and presence protocol (XMPP): Core. RFC 6120, RFC Editor (Mar 2011). https://doi.org/10.17487/RFC6120
22. Sakamoto, H., Ikeda, D.: Intractability of decision problems for finite-memory automata. Theor. Comput. Sci. **231**(2), 297–308 (2000). https://doi.org/10.1016/S0304-3975(99)00105-X
23. Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In: Ésik, Z. (ed.) Computer Science Logic. pp. 41–57. No. 4207 in LNCS, Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11874683_3
24. Tzevelekos, N.: Fresh-register automata. In: Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 295–306. ACM, New York, NY, USA (2011). https://doi.org/10.1145/1926385.1926420