Scoped MSO, Register Automata, and Expressions: Equivalence over Data Words

Radosław Piórkowski
University of Warwick, Coventry, UK

Abstract

This paper establishes logical and expression-based characterizations of the class of languages recognized by nondeterministic register automata with guessing (NRA) over infinite alphabets. We introduce Scoped MSO, a logic featuring a novel segment modality and syntactic restrictions on data comparisons. We prove this logic is expressively equivalent to NRA over data domains where "strong guessing" can be eliminated. Furthermore, we define Data Regular Expressions, a minimalist regular expression calculus built from quantifier-free regions and equipped with k-contracting concatenation, and demonstrate its equivalence to NRA over arbitrary relational structures. Together, these formalisms and the translations between them establish a robust correspondence between register automata, logic, and expressions over data words.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Logic and verification; Theory of computation → Regular languages

Keywords and phrases register automata, data words, monadic second-order logic, infinite alphabets, Kleene theorem, nominal sets

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.188

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2602.13120` [16]

Funding *Radosław Piórkowski*: The author is supported by the Engineering and Physical Sciences Research Council [EP/X03027X/1].

Acknowledgements The author would like to thank Dmitry Chistikov for his valuable guidance and insightful discussions, and the Centre for Discrete Mathematics and its Applications (DIMAP) at the University of Warwick. The author also thanks the anonymous reviewers for their helpful comments and suggestions.

1 Introduction

A remarkable property of the class of *regular languages* is its striking *robustness*: one and the same class of languages admits several radically different descriptions. Over a finite alphabet Σ, finite automata, regular expressions, and monadic second-order logic over word structures (MSO) define exactly the regular languages. This three-way equivalence (and its numerous refinements via algebra, temporal logics, varieties, etc.) forms the backbone of classical language theory and explains why "regularity" is such a stable and reusable concept.

When moving from finite to *infinite alphabets* – modelling systems with process IDs, database keys, large alphabets such as Unicode, or data values in XML – this robustness largely disappears. The standard abstraction for such traces is *data words*: sequences over $\Sigma \times \mathbb{A}$ where Σ is a finite set of labels and $\mathbb{A}$ is a data domain called *atoms*, which is modelled as a relational structure with countably-infinite universe. The intended access to elements of $\mathbb{A}$ (called *data values*) is limited to testing relations from the signature of $\mathbb{A}$; common examples of atoms include equality atoms $\mathbb{A}_{=} = (\mathbb{N}, =)$ and dense order atoms $\mathbb{A}_{<} = (\mathbb{Q}, <, =)$. The landscape of formalisms for data words is fragmented, featuring a

wide array of automata types and logics whose expressive powers are often incomparable and whose decision problems vary wildly in complexity and decidability [14, 18, 1, 2, 13]. Unlike the finite-alphabet case, there is no single notion that plays the role of "*the*" regular languages for data words.

Register Automata. A natural candidate for a finite-state model over data words is the *register automaton*, introduced by Kaminski and Francez [10]. These automata extend finite control with a fixed set of registers capable of storing data values and comparing them for equality. The model was subsequently extended to arbitrary atoms $\mathbb{A}$ [5]. In this paper, we target the class of *nondeterministic register automata with guessing* ($\mathrm{NRA}^{\mathrm{G}}$). The "guessing" capability is the most permissive out of the three possible variants of nondeterminism in register automata. While $\mathrm{NRA}^{\mathrm{G}}$ are a standard baseline, they have historically lacked an MSO-style logical characterisation.

MSO Logic for NRA. Establishing a correspondence between $\mathrm{NRA}^{\mathrm{G}}$ and logic requires careful calibration of the expressive power. Simply adding data atomic formulas to MSO[<] leads to undecidability [14, 3], even for the first-order (FO) fragment with three variables and the simplest variant of atoms $\mathbb{A}_{=}$: one may encode grid-like structures of arbitrary dimension in data words and simulate runs of Turing machines on them. Conversely, restricting the logic too aggressively (e.g., to rigid guards, or to the two-variable FO fragment) yields decidability but fails to capture the full expressive power of $\mathrm{NRA}^{\mathrm{G}}$ [8, 4, 3]. The lack of closure of $\mathrm{NRA}^{\mathrm{G}}$ under language complement [13] poses yet another challenge: any logic of the same expressive power must impose limitations on the negation operation.

Expressions for NRA. Regarding expression formalisms for data languages over equality atoms $\mathbb{A}_{=}$, Kaminski and Tan proposed unification based expressions [11]. Subsequently, Brunet and Silva defined expressions featuring explicit binders for name allocation and deallocation [6]. Their formalism provides a versatile *specification* language, allowing for concise and readable expressions at the cost of a more involved definition of the semantics. We aim for a complementary goal: a compact expression formalism whose syntax is close in spirit to classical regular expressions and captures languages of NRA over arbitrary atoms.

Contributions. In this paper, we provide two new formalisms for languages of data words over $\Sigma \times \mathbb{A}$: Scoped MSO logic ($\mathrm{MSO}_{\mathrm{s}}[<, \Sigma, \mathbb{A}]$) and Data Regular Expressions ($\mathrm{DRE}[\Sigma, \mathbb{A}]$), recovering the classical trinity for register automata:

$$\text{Automata } (\mathrm{NRA}^{\mathrm{G}}[\Sigma, \mathbb{A}]) \;\equiv\; \text{Expressions } (\mathrm{DRE}[\Sigma, \mathbb{A}]) \;\equiv\; \text{Logic } (\mathrm{MSO}_{\mathrm{s}}[<, \Sigma, \mathbb{A}])\,.$$

1. **Data Regular Expressions:** We introduce a compact expression formalism over $\Sigma \times \mathbb{A}$. DRE are built from *quantifier-free regions* of data words. The expressions include a parameterized operation on languages, called *k-contracting concatenation*, that enables expressing non-local data value properties: k-contraction simulates the "transfer" of k data values between factors, mirroring the finite set of registers of an NRA.
   - ▶ We prove that $\mathcal{L}(\mathrm{DRE}[\Sigma, \mathbb{A}]) = \mathcal{L}\big(\mathrm{NRA}^{\mathrm{G}}[\Sigma, \mathbb{A}]\big)$ for arbitrary atoms $\mathbb{A}$.
2. **Scoped MSO:** We define a logic tailored to the expressive power of $\mathrm{NRA}^{\mathrm{G}}$. $\mathrm{MSO}_{\mathrm{s}}[<]$ extends MSO with two features:
   - A *segment modality* $\partial_X\, \varphi$, which evaluates φ on sub-intervals defined by a set of cut-positions X; it constitutes a variant of quantifier relativization.
   - *Restricted atomic predicates* $R(\bar{x})$ with a *syntactic restriction* preventing arbitrary data comparisons that would exceed the capacity of finite registers.

▶ We prove that $\mathcal{L}(\mathrm{MSO}_{\mathrm{s}}[<, \Sigma, \mathbb{A}]) = \mathcal{L}\big(\mathrm{NRA}^{\mathrm{G}}[\Sigma, \mathbb{A}]\big)$, provided the atoms $\mathbb{A}$ admit the *elimination of strong guessing*: a technical property we formalize. This condition holds for commonly studied atom structures, including equality atoms $\mathbb{A}_{=}$ and dense order atoms $\mathbb{A}_{<}$. We discuss natural adaptations of $\mathrm{MSO}_{\mathrm{s}}[<]$ for ω-words and subclasses of $\mathrm{NRA}^{\mathrm{G}}$.

Implications and Outlook. Our results complement the operational theory of NRA with logical and expression-based characterisations paralleling the classical finite-alphabet setting, offering new perspectives on languages over infinite alphabets. We leave open whether there is any strengthening of $\mathrm{MSO}_{\mathrm{s}}[<, \Sigma, \mathbb{A}]$ which retains decidable satisfiability, while going beyond the class of $\mathrm{NRA}^{\mathrm{G}}[\Sigma, \mathbb{A}]$ languages.

2 Preliminaries

We use standard notations to denote natural numbers $\mathbb{N}$, positive natural numbers $\mathbb{N}_+$ or rationals $\mathbb{Q}$. We denote ranges of natural numbers by $[i, j] \coloneqq \{i, i+1, \ldots, j\}$ and $[j] \coloneqq [1, j]$.

Words. Fix an alphabet Σ and a word $w = a_1 a_2 \cdots a_n \in \Sigma^*$. We write $w[i] \coloneqq a_i$ and, for an interval $I = [i, j]$, we write $w[I]$ for the infix $a_i a_{i+1} \cdots a_j$. The reversed word $a_n \cdots a_1$ is denoted as w^R. The length n of w is denoted as $|w|$, and the set of *positions* of w is $\mathrm{pos}(w) = [n]$. The unique word of length 0 is called the *empty word* and denoted by ε.

Word structures. Given a $w \in \Sigma^*$, a *word structure* $\mathbb{W}(w)$ is the relational structure defined as $\mathbb{W}(w) = (\mathrm{pos}(w), <, (\sigma)_{\sigma \in \Sigma})$ where "$<$" is interpreted as the natural linear ordering on $\mathrm{pos}(w)$, and for each letter $\sigma \in \Sigma$ we have a unary relation symbol $\sigma = \{p \in \mathrm{pos}(w) \mid w[p] = \sigma\}$ interpreted as the set of positions in w labelled with σ. While we use σ both for letters of Σ and position predicates, the intended meaning is clear from the context.

2.1 Formalisms for regular languages

This paper's starting point is the equivalence of the following three formalisms defining regular languages. Fix a finite alphabet Σ.

Nondeterministic finite automata. A *Nondeterministic Finite Automaton* (NFA) is a tuple $\mathcal{A} = (Q, \Sigma, Q_{\mathrm{ini}}, Q_{\mathrm{fin}}, \delta)$, where Q is a finite set of states, Q_{ini} and Q_{fin} its initial and accepting subsets, and $\delta \subseteq Q \times \Sigma \times Q$. We call the elements of δ *transitions* and denote $(q, a, r) \in \delta$ using the arrow notation $q \xrightarrow{a} r$. A *run* of $\mathcal{A}$ of length n over $w = a_1 a_2 \cdots a_n$ is a w-labelled path $\pi = t_1 t_2 \cdots t_n \in \delta^*$ in the labelled graph (Q, δ), i.e., a sequence such that t_i has the form $(\cdot, a_i, \cdot)$ for every i, and for any two consecutive transitions $p \xrightarrow{a} q$ and $r \xrightarrow{b} s$ we have $q = r$. We say that $\mathcal{A}$ *accepts* $w \in \Sigma^*$ if there exists a run of $\mathcal{A}$ over w that begins in some initial state $q_{\mathrm{ini}} \in Q_{\mathrm{ini}}$ and ends in an accepting state $q_{\mathrm{fin}} \in Q_{\mathrm{fin}}$. The language of $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w\}$. We write $\mathrm{NFA}[\Sigma]$ for the family of all NFA over Σ.

Regular expressions. *Regular expressions* over Σ ($\mathrm{RE}[\Sigma]$) are terms generated by the grammar

$$E, F ::\equiv \emptyset \mid \varepsilon \mid a \in \Sigma \mid E + F \mid E \cdot F \mid E^* .$$

The language $\mathcal{L}(E)$ of an expression E is defined as follows

$$\mathcal{L}(\emptyset) = \emptyset \qquad \mathcal{L}(\varepsilon) = \{\varepsilon\} \qquad \mathcal{L}(a) = \{a\}$$
$$\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F) \qquad \mathcal{L}(E \cdot F) = \mathcal{L}(E) \cdot \mathcal{L}(F) \qquad \mathcal{L}(E^*) = \bigcup_{n \geq 0} \mathcal{L}(E)^n \,,$$

where the language concatenation is $K \cdot L \coloneqq \{uv \mid u \in K, v \in L\}$ for $K, L \subseteq \Sigma^*$, and K^n denotes n-fold concatenation of K with itself; $K^0 \coloneqq \{\varepsilon\}$.

▶ **Theorem 1** (Kleene [12]). *$\mathcal{L}(\mathrm{NFA}[\Sigma]) = \mathcal{L}(\mathrm{RE}[\Sigma])$ for any finite Σ.*

Syntax of MSO. The syntax of *Monadic Second-Order Logic over words* ($\mathrm{MSO}[<, \Sigma]$) features first-order variables $x, y, \ldots$, second-order variables $X, Y, \ldots$ and is given by the grammar

$$\varphi, \psi ::\equiv \exists x\ \varphi \mid \exists X\ \varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi \mid x < y \mid X(x) \mid a(x)\,,$$

where $a \in \Sigma$. We write $\varphi \in \mathrm{MSO}[<, \Sigma]$ when φ is generated by the above grammar. A variable in φ is free, if it is not bound by any quantifier. We write $\mathrm{free}(\varphi)$ (or $\mathrm{free}_1(\varphi)$) for the set of free variables (of first-order variables, respectively) of φ. A formula is a *sentence*, if it has no free variables.

Semantics of MSO. The truth value of $\mathrm{MSO}[<, \Sigma]$ formulas depends on the interpretation $\mathfrak{I} = (\mathbb{W}(w), \nu)$ consisting of the word structure $\mathbb{W} = \mathbb{W}(w)$ and valuation $\nu\colon \mathrm{free}(\varphi) \to \mathrm{pos}(w) \cup \mathcal{P}(\mathrm{pos}(w))$. Boolean connectives and atomic formulas ($x < y, X(y), a(x)$) are interpreted in a standard way over $\mathbb{W}$. In particular, $\mathfrak{I} \models a(x)$ iff $\nu(x) \in a^{\mathbb{W}}$. Quantifiers range over $\mathrm{pos}(w)$:

- $\mathfrak{I} \models \exists x\ \varphi$ iff there is a position $p \in \mathrm{pos}(w)$ such that $\mathbb{W}(w), \nu[x \mapsto p] \models \varphi$,
- $\mathfrak{I} \models \exists X\ \varphi$ iff there is a set $P \subseteq \mathrm{pos}(w)$ such that $\mathbb{W}(w), \nu[X \mapsto P] \models \varphi$.

For a sentence $\varphi \in \mathrm{MSO}[<, \Sigma]$, we write $\mathbb{W}(w) \models \varphi$ whenever $\mathbb{W}(w), \nu_0 \models \varphi$, where ν_0 is the empty valuation. The language of φ is defined as $\mathcal{L}(\varphi) = \{w \in \Sigma^* \mid \mathbb{W}(w) \models \varphi\}$.

▶ **Theorem 2** (Büchi, Elgot, Trakhtenbrot [7, 9, 19]). *For every finite Σ, the languages definable by $\mathrm{MSO}[<, \Sigma]$ are exactly the regular languages over Σ.*

Syntactic sugar. We use $\forall\xi\ \varphi$ as a shortcut for $\neg\exists\xi\ (\neg\varphi)$; position equality is defined as $x = y \equiv \neg(x < y \vee y < x)$; Boolean constants can be defined as $\top \equiv \exists x\ x = x \vee \neg\exists x\ x = x$ and $\bot \equiv \neg\top$. Additionally, we add easily definable relations like the subset relation $S' \subseteq S$.

2.2 Words over infinite alphabets

While the definitions of a word and a language do not require *finiteness of the alphabet*, all the basic language-defining formalisms we have recalled in Section 2.1 critically depend on that assumption. For instance, a finite automaton for the language Σ consisting of one-letter words has to contain a transition for every $a \in \Sigma$, and making Σ infinite would cause the transition relation to become infinite as well. One solution to this problem is to add more structure to the alphabet and allow the model to make less precise queries about the letters. This gives rise to the realm of *atoms* [15] and *register automata* [13].

Atoms. We call atoms any relational structure $\mathbb{A} = (U, R_1, \ldots, R_\ell)$ with a countably infinite universe U and a finite number of relational symbols in its signature. We assume that one of R_i is the equality of universe elements; it is sometimes written as $\sim$ to avoid confusion with other uses of $=$. Abusing notation, we write $\alpha \in \mathbb{A}$ to mean α is an element in the domain U of $\mathbb{A}$. Atoms commonly found in the literature include equality atoms $\mathbb{A}_= = (\mathbb{N}, =)$ and dense order atoms $\mathbb{A}_< = (\mathbb{Q}, <, =)$.

Data words. Data words are words over a *two-track* alphabet $\Sigma \times \mathbb{A}$, consisting of a finite label from Σ and a data value from $\mathbb{A}$. It is often convenient to decompose multi-track words into separate tracks using *convolution*. For alphabets $A_1, \ldots, A_k$, the convolution of words $w_1 \in A_1^*, \ldots, w_k \in A_k^*$ of equal length is the word $\otimes(w_1, \ldots, w_k) \in (A_1 \times \cdots \times A_k)^*$ defined by $\otimes(w_1, \ldots, w_k)[i] \coloneqq (w_1[i], \ldots, w_k[i])$. Thus, a data word can be viewed as the convolution of a label word over Σ and a data sequence over $\mathbb{A}$.

▶ **Example 3.** Let $\mathbb{A} = (\mathbb{N}, =)$ and $\Sigma = \{a, b\}$. A two-track word $\otimes(w, \varpi)$ over $\Sigma \times \mathbb{A}$ is a convolution of two tracks $w \in \Sigma^*$ and $\varpi \in \mathbb{A}^*$:

$$\begin{aligned} w &= \mathrm{abaab} \\ \varpi &= 10, 42, 5, 97, 42 \\ \otimes(w, \varpi) &= \begin{bmatrix} \mathrm{a} \\ 10 \end{bmatrix} \begin{bmatrix} \mathrm{b} \\ 42 \end{bmatrix} \begin{bmatrix} \mathrm{a} \\ 5 \end{bmatrix} \begin{bmatrix} \mathrm{a} \\ 97 \end{bmatrix} \begin{bmatrix} \mathrm{b} \\ 42 \end{bmatrix} . \end{aligned}$$

Above, $\varpi[2] = \varpi[5] = 42$ and $w[3]$ is a.

Quantifier-free formulas and regions over atoms. Let $\mathcal{X}$ be a finite set of variables. The set $\mathrm{Lit}_{\mathbb{A}}(\mathcal{X})$ of *atomic formulas* over $\mathbb{A}$ using variables $\mathcal{X}$ consists of all expressions $R(x_1, \ldots, x_r)$ where R is a relation symbol from the signature of $\mathbb{A}$ and $x_i \in \mathcal{X}$. A *quantifier-free formula* over $\mathbb{A}$ using variables $\mathcal{X}$ is a Boolean combination of atomic formulas:

$$\Phi, \Psi ::\equiv \top \mid \neg\Phi \mid \Phi \wedge \Psi \mid R(x_1, \ldots, x_r) .$$

The formulas are interpreted in a standard way over a pair $(\mathbb{A}, \nu)$, where $\nu \colon \mathcal{X} \to \mathbb{A}$.

Fix $k \in \mathbb{N}$ and $\mathcal{X}_k \coloneqq \{x_1, \ldots, x_k\}$. For a word $w = a_1 \cdots a_k \in \mathbb{A}^k$, the *quantifier-free type* of w, denoted $\mathrm{type}(w)$, is the set of all literals satisfied when variables $\mathcal{X}_k$ are interpreted as data values in w. Formally, $\mathrm{type}(w) \coloneqq \{\phi \in \mathrm{Lit}_{\mathbb{A}}(\mathcal{X}_k) \mid \mathbb{A}, \nu \models \phi\}$ where $\nu(x_i) = a_i$. A *quantifier-free region* of a word $w \in \mathbb{A}^*$, denoted $\mathrm{region}_{\mathbb{A}}(w)$, is the set of all words that have the same quantifier-free type as w. Intuitively, words in $\mathrm{region}_{\mathbb{A}}(w)$ are not distinguishable from w by quantifier-free formulas. We lift $\mathrm{region}_{\mathbb{A}}$ in a natural way to words over $(\mathbb{A} \cup \Sigma \times \mathbb{A})^*$, with some letters having additional labels from a finite alphabet Σ. It is easy to see that $\mathbb{A}^k$ is a finite union of quantifier-free regions for any k.

Data word structures. Fix atoms $\mathbb{A} = (U, R_1^{\mathbb{A}}, \ldots, R_\ell^{\mathbb{A}})$ and let $\otimes(w, \varpi) \in (\Sigma \times U)^*$ with $|w| = |\varpi|$. The *data word structure* is the expansion of $\mathbb{W}(w) = (\mathrm{pos}(w), <, (\sigma)_{\sigma \in \Sigma})$

$$\mathbb{W}(w, \varpi) \coloneqq (\mathrm{pos}(w), <, (\sigma)_{\sigma \in \Sigma}, (R_i)_{i \in [\ell]}) ,$$

where, if R_i has arity m_i, then for all $p_1, \ldots, p_{m_i} \in \mathrm{pos}(w)$,

$$R_i(p_1, \ldots, p_{m_i}) \quad \text{iff} \quad R_i^{\mathbb{A}}(\varpi[p_1], \ldots, \varpi[p_{m_i}]) .$$

2.3 Register automata: a formalism for data languages

One of the common formalisms for defining languages of data words is a nondeterministic register automaton ($\mathrm{NRA}^{\mathrm{G}}$). Fix atoms $\mathbb{A} = (U, R_1, \ldots, R_\ell)$ and a finite set $\mathcal{R}$ whose elements we call *registers*. Let $k := |\mathcal{R}|$ and let $\mathbb{A}_{\mathrm{nil}} := \mathbb{A} \uplus \{\underline{\mathrm{nil}}\}$, where nil is a fresh element modelling an empty register, and the signature of $\mathbb{A}_{\mathrm{nil}}$ is extended with a unary symbol nil interpreted as $\{\underline{\mathrm{nil}}\}$. A register valuation is a function $\mu \colon \mathcal{R} \to \mathbb{A}_{\mathrm{nil}}$.

Register constraints. Let $\mathcal{V}_{\mathcal{R}} := \bigcup_{r \in \mathcal{R}} \{\overset{\triangleleft}{r}, \overset{\triangleright}{r}\}$ be the variables representing register values before and after a transition, and let $\triangledown$ represent the current input symbol. A *register constraint* Φ is a quantifier-free formula over $\mathbb{A}_{\mathrm{nil}}$ with the variable set $\{\triangledown\} \cup \mathcal{V}_{\mathcal{R}}$. We denote the set of all register constraints for $\mathcal{R}$ as $\mathrm{constr}_{\mathbb{A}}(\mathcal{R})$. To evaluate a constraint, consider two valuations $\mu, \mu' \colon \mathcal{R} \to \mathbb{A} \cup \{\mathrm{nil}\}$ and an input symbol $\alpha \in \mathbb{A}$. These induce a combined valuation $\nu(\mu, \alpha, \mu')$ on the variables $\{\triangledown\} \cup \mathcal{V}_{\mathcal{R}}$ such that $\overset{\triangleleft}{r} \mapsto \mu(r)$, $\overset{\triangleright}{r} \mapsto \mu'(r)$, and $\triangledown \mapsto \alpha$.

Register automata. A *nondeterministic register automaton with guessing* ($\mathrm{NRA}^{\mathrm{G}}$) is a tuple $\mathcal{A} = (Q, \Sigma, \mathbb{A}, \mathcal{R}, Q_{\mathrm{ini}}, Q_{\mathrm{fin}}, \delta)$, comprising a finite set of states Q, a finite alphabet Σ, initial states $Q_{\mathrm{ini}} \subseteq Q$, accepting states $Q_{\mathrm{fin}} \subseteq Q$ and a set of *transition rules* $\delta \subseteq Q \times \Sigma \times \mathrm{constr}_{\mathbb{A}}(\mathcal{R}) \times Q$. A transition rule $(p, \sigma, \Phi, q) \in \delta$ is written using double arrow notation $p \overset{\sigma,\Phi}{\Longrightarrow} q$.

The semantics is defined by an infinite transition system $[\![\mathcal{A}]\!] := (C_{\mathcal{A}}, \Delta_{\mathcal{A}})$ over *configurations* $C_{\mathcal{A}} := Q \times (\mathcal{R} \to \mathbb{A}_{\mathrm{nil}})$, where $\Delta_{\mathcal{A}} \subseteq C_{\mathcal{A}} \times \Sigma \times \mathbb{A} \times C_{\mathcal{A}}$. A transition $(p, \mu) \xrightarrow{\sigma,\alpha} (q, \mu') \in \Delta_{\mathcal{A}}$ exists iff there is a rule $p \overset{\sigma,\Phi}{\Longrightarrow} q$ in δ such that $\mathbb{A}, \nu(\mu, \alpha, \mu') \models \Phi$. A configuration (q, μ) is initial if $q \in Q_{\mathrm{ini}}$ and $\mu(r) = \mathrm{nil}$ for every $r \in \mathcal{R}$. A configuration (q, μ) is *accepting* when $q \in Q_{\mathrm{fin}}$. The notion of a run $\pi \in \Delta_{\mathcal{A}}^*$ is analogous to that in an NFA; it is accepting if it starts in some initial configuration and ends in some accepting one. The language $\mathcal{L}(\mathcal{A})$ consists of data words labelling accepting runs of $[\![\mathcal{A}]\!]$. Every run π of $[\![\mathcal{A}]\!]$ has an associated sequence $t_1 \cdots t_n \in \delta^*$ of transition rules. We denote the family of k-register NRA over $\Sigma \times \mathbb{A}$ by $\mathrm{NRA}^{\mathrm{G}}_k[\Sigma, \mathbb{A}]$.

Variants of nondeterminism in register automata. Fix $\mathcal{A} \in \mathrm{NRA}^{\mathrm{G}}_k$ and its run $\pi = (q_0, \mu_0) \xrightarrow{\sigma_1,\alpha_1} (q_1, \mu_1) \to \cdots \to (q_{n-1}, \mu_{n-1}) \xrightarrow{\sigma_n,\alpha_n} (q_n, \mu_n)$ over $w \in (\Sigma \times \mathbb{A})^n$. We say that range $[a, b]$ is a *live interval* of a data value ξ whenever $\xi \in \mu_m(\mathcal{R})$ for all $m \in [a-1, b]$, but $\xi \notin \mu_{a-2}(\mathcal{R})$ (if defined) and $\xi \notin \mu_{b+1}(\mathcal{R})$ (if defined). In other words, the live interval of ξ is a maximal range of letter positions for which ξ is in both surrounding register valuations.

We say that $\mathcal{A}$ is *guessing* the value of the jth register in valuation μ_a, if $\xi = \mu_a(r_j)$ is different from all previous register values (in $\mu_{a-1}(\mathcal{R})$) and from the data value α_a. The guess can be either *weak* or *strong*, depending on the run from configuration (q_a, μ_a) onwards. Let $[a+1, b]$ be the live interval of ξ. The guess of the value of jth register in valuation μ_a is strong whenever ξ does not appear in the input on positions $[a, b+1]$.

We distinguish two restricted subclasses of $\mathrm{NRA}^{\mathrm{G}}$: *weakly-guessing nondeterministic register automata* ($\mathrm{NRA}^{\mathrm{W}}$) and *deterministic register automata without guessing* ($\mathrm{NRA}^{\mathrm{N}}$). We say that $\mathcal{A} \in \mathrm{NRA}^{\mathrm{G}}[\Sigma, \mathbb{A}]$ is *weakly-guessing*, if $\mathcal{A}$ is not guessing strongly in its accepting runs. It is *without guessing*, if it never guesses a value in any of its runs.

▶ **Definition 4** (Elimination of strong guessing). *We say that atoms $\mathbb{A}$ have elimination of strong guessing, whenever for every $\mathcal{A} \in \mathrm{NRA}^{\mathrm{G}}[\Sigma, \mathbb{A}]$ there exists a weakly guessing $\mathcal{B} \in \mathrm{NRA}^{\mathrm{W}}[\Sigma, \mathbb{A}]$ that recognises the same language.*

3 New formalisms equivalent to register automata

3.1 Data regular expressions

Fix atoms $\mathbb{A}$ and a finite Σ. We introduce a notion of *data regular expressions* over $\Sigma \times \mathbb{A}$, which are generated by the following grammar

$$E, F ::\equiv \emptyset \mid \varepsilon \mid [w] \mid E + F \mid E \cdot_k F \mid E^{k,+} ,$$

where $k \in \mathbb{N}$ and $w \in \mathbb{A}^*(\Sigma \times \mathbb{A})\mathbb{A}^*$ contains exactly one letter with a label from the finite alphabet Σ. Let $\mathrm{DRE}_0[\Sigma, \mathbb{A}]$ be the set of all such data expressions. The language $\mathcal{L}(E)$ of $E \in \mathrm{DRE}_0[\Sigma, \mathbb{A}]$ is a subset of $(\mathbb{A} \cup \Sigma \times \mathbb{A})^*$ and is defined as follows:

$$\mathcal{L}(\emptyset) = \emptyset \qquad \mathcal{L}(\varepsilon) = \{\varepsilon\} \qquad \mathcal{L}([w]) = \mathrm{region}_{\mathbb{A}}(w)$$
$$\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F) \qquad \mathcal{L}(E \cdot_k F) = \mathcal{L}(E) \cdot_k \mathcal{L}(F) \qquad \mathcal{L}\big(E^{k,+}\big) = \textstyle\bigcup_{n \geq 1} \mathcal{L}(E)^{k,n} ,$$

where $\cdot_k$ is *k-contracting concatenation* defined as $K \cdot_k L \coloneqq \{uw \mid uv \in K, v^R w \in L, v \in \mathbb{A}^k\}$ for data word languages K, L, and $L^{n,k}$ for $n > 0$ denotes the k-contracting concatenation of n copies of L.

We say that an expression is *well-formed* if its language is a subset of $(\Sigma \times \mathbb{A})^*$, i.e., all letters from $\mathbb{A}$ without a label from Σ vanish due to contracting concatenations. We denote the set of all well-formed expressions by $\mathrm{DRE}[\Sigma, \mathbb{A}] \subseteq \mathrm{DRE}_0[\Sigma, \mathbb{A}]$.

▶ **Example 5.** Consider equality atoms $\mathbb{A}_= = (\mathbb{N}, \sim)$ and let $\Sigma = \{\circ\}$. We write $\overset{\circ}{\alpha}$ for the pair $(\circ, \alpha) \in \Sigma \times \mathbb{A}$. Contracting concatenation enables expressing long-distance relationships between data values in a word. For instance, the language of data words that have the first and last data value equal can be defined by the following well-formed expression:

$$E = [\overset{\circ}{0}] + \Big([\overset{\circ}{0}\overset{\circ}{0}] \cdot_1 [\overset{\circ}{0}\overset{\circ}{0}]\Big) + \Big([\overset{\circ}{0}\overset{\circ}{0}] \cdot_1 \big([\overset{\circ}{0}\overset{\circ}{0}\overset{\circ}{0}] + [\overset{\circ}{0}\overset{\circ}{1}\overset{\circ}{0}]\big)^{1,+} \cdot_1 [\overset{\circ}{0}\overset{\circ}{0}]\Big) .$$

We have $\overset{\circ}{7}\overset{\circ}{8}\overset{\circ}{7}\overset{\circ}{7} \in \mathcal{L}(E)$ because $\overset{\circ}{7}\overset{\circ}{7} \in \mathcal{L}\big([\overset{\circ}{0}\overset{\circ}{0}]\big)$, $\overset{\circ}{7}\overset{\circ}{8}\overset{\circ}{7}, \overset{\circ}{7}\overset{\circ}{7}\overset{\circ}{7} \in \mathcal{L}\big([\overset{\circ}{0}\overset{\circ}{0}\overset{\circ}{0}] + [\overset{\circ}{0}\overset{\circ}{1}\overset{\circ}{0}]\big)$ and $\overset{\circ}{7}\overset{\circ}{7} \in \mathcal{L}\big([\overset{\circ}{0}\overset{\circ}{0}]\big)$. The information about the data value from the first position is present in the prefixes/suffixes of length one that get contracted.

▶ Remark 6. The operator $\cdot_k$ is associative, because the design of expressions ensures that the regions of k-contractions do not overlap: for every $E \in \mathrm{DRE}_0[\Sigma, \mathbb{A}]$, and for every $w \in \mathcal{L}(E)$, w is either empty, or contains a non-contractible letter from $\Sigma \times \mathbb{A}$.

▶ **Theorem 7.** $\mathcal{L}\big(\mathrm{NRA}^{\mathrm{G}}[\Sigma, \mathbb{A}]\big) = \mathcal{L}(\mathrm{DRE}[\Sigma, \mathbb{A}])$ *for any finite* Σ *and atoms* $\mathbb{A}$.

3.2 Scoped MSO

Syntax of Scoped MSO. Fix finite Σ and atoms $\mathbb{A} = (U, R_1, \ldots, R_\ell)$. The syntax of Scoped MSO uses first-order variables $x, y, \ldots$, second-order variables $X, Y, \ldots$, and is generated by

$$\varphi, \psi ::\equiv \exists x\ \varphi \mid \exists X\ \varphi \mid \wr_X\ \varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi \mid x < y \mid X(x) \mid a(x) \mid R(\bar{x}) ,$$

for all letters $a \in \Sigma$, relations $R \in \{R_1, \ldots, R_\ell\}$ and where $\bar{x} = (x_1, \ldots, x_r)$ matches the arity of R. There are two additions compared to the grammar of standard MSO (as per Section 2.1): $\wr_X$ is the *scope modality* and $R(\bar{x})$ is a *data atomic formula*.

A first-order variable in φ is *top-level* if its binding existential quantifier does not occur within the scope of a negation; otherwise, it is *nested*. (Note that variables in $\forall x\ \varphi \equiv \neg\exists x\ \neg\varphi$ are inherently nested).

▶ Condition 8 (Well-formedness). A formula φ is *well-formed* if every data atomic formula $R(\bar{x})$ occurring in φ contains at most one nested variable.

▶ **Example 9.** Consider equality atoms $\mathbb{A}_{=} = (\mathbb{N}, \sim)$. The formula $\exists x\, \neg[\exists y.\ x \sim y \wedge x \neq y]$ is well-formed because x is top-level. However, $\neg[\exists x\, \exists y.\ x \sim y \wedge x \neq y]$ is not, as the outer negation makes both x and y nested. The modality $\circlearrowleft_X$ preserves the top-level status of variables; thus, $\exists X\ \circlearrowleft_X\ \exists x\, \neg[\exists y.\ x \sim y \wedge x \neq y]$ remains well-formed.

We denote the set of all well-formed formulas as $\mathrm{MSO_s}[<, \Sigma, \mathbb{A}]$.

Subscopes. Let $I = [a, b]$ and let $X \subseteq \mathbb{N}$. We view X as a set of *split points*, and we define the set $\mathrm{subs}(I, X)$ of the contiguous sub-ranges obtained by cutting I at every position in X. Formally, $\mathrm{subs}(I, X)$ is defined using the set of left boundary points $B = (X \cap I) \cup \{a,\, b+1\}$. A range $[a', b']$ belongs to $\mathrm{subs}(I, X)$ whenever a' and $b'+1$ are two consecutive left boundary points from the list resulting from enumerating B in the increasing order.

▶ **Example 10.** Let $I = [a, b] = [10, 20]$ and $X = \{5, 7, 11, 13, 17, 23\}$. The set of left boundary points $B = (X \cap I) \cup \{a, b+1\}$ is $\{10, 11, 13, 17, 21\}$, and therefore

$$\mathrm{subs}(I, X) = \{\underset{\substack{\| \\ a}}{10}\}, \{\underset{\substack{\cap \\ X}}{11}, 12\}, \{\underset{\substack{\cap \\ X}}{13}, 14, 15, 16\}, \{\underset{\substack{\cap \\ X}}{17}, 18, 19, \underset{\substack{\| \\ b}}{20}\}\,.$$

Semantics of Scoped MSO. We interpret $\mathrm{MSO_s}[<, \Sigma, \mathbb{A}]$ formulas over a tuple $\mathfrak{I} = (\mathbb{W}(w, \varpi), \nu, I)$ consisting of the data word structure $\mathbb{W} = \mathbb{W}(w, \varpi)$, valuation $\nu\colon \mathrm{free}(\varphi) \to \mathrm{pos}(w) \uplus \mathcal{P}(\mathrm{pos}(w))$ and a discrete range $I = [i, j] \subseteq \mathrm{pos}(w)$ called a *scope*. Boolean connectives and atomic formulas $(x < y, X(y), a(x), R(\bar{x}))$ are interpreted in a standard way over $(\mathbb{W}, \nu)$, ignoring I. In particular, $\mathfrak{I} \models R(\bar{x})$ iff $\nu(\bar{x}) \in R^{\mathbb{W}}$. Quantifiers are relativised to the current scope I:

- $\mathfrak{I} \models \exists x\ \varphi$ iff there is a position $p \in I$ such that $\mathbb{W}(w, \varpi), \nu[x \leftarrow p], I \models \varphi$,
- $\mathfrak{I} \models \exists X\ \varphi$ iff there is a set of positions $P \subseteq I$ such that $\mathbb{W}(w, \varpi), \nu[X \leftarrow P], I \models \varphi$.

Finally, $\circlearrowleft_X$ enforces the formula on all induced sub-scopes:

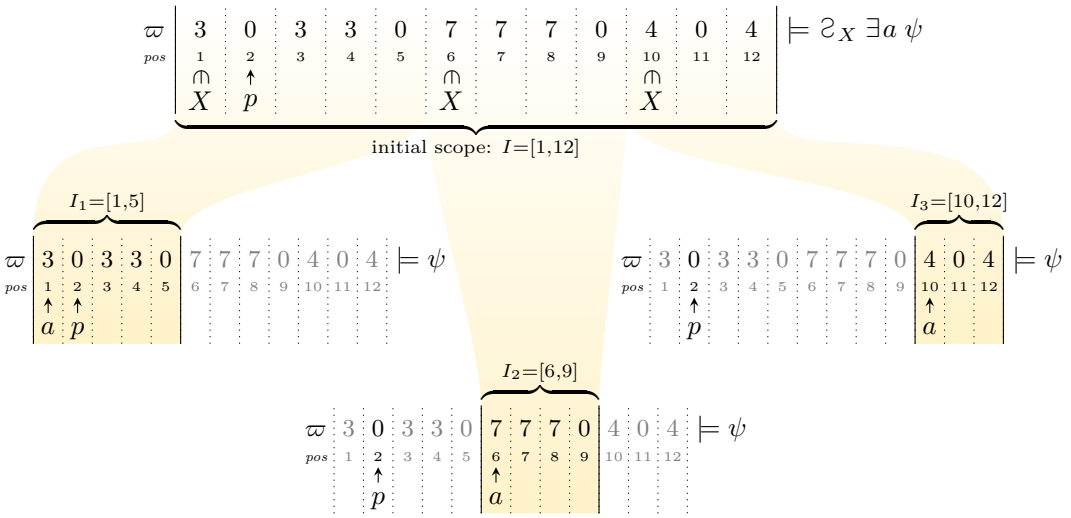
- $\mathfrak{I} \models \circlearrowleft_X\ \varphi$ iff $\mathbb{W}(w, \varpi), \nu, I' \models \varphi$ for every $I' \in \mathrm{subs}(I, \nu(X))$.

When the scope interval is the entire $\mathrm{pos}(w)$, we drop it in the interpretation tuple and write $\mathbb{W}(w, \varpi), \nu \models \varphi$. When φ is a sentence, we drop the valuation and write $\mathbb{W}(w, \varpi) \models \varphi$, and we define the *language* of φ as $\mathcal{L}(\varphi) = \{\otimes(w, \varpi) \mid \mathbb{W}(w, \varpi) \models \varphi\}$.

▶ **Example 11** (A formula of $\mathrm{MSO_s}[<]$ and its interpretations)**.** Consider equality atoms $\mathbb{A}_{=} = (\mathbb{N}, \sim)$ and let $\Sigma = \{\circ\}$ be a unary alphabet, allowing us to disregard the finite-alphabet letters. Consider the following formula of $\mathrm{MSO_s}[<, \Sigma, \mathbb{A}_{=}]$:

$$\varphi \equiv \exists p\, \exists X\ \circlearrowleft_X\ \exists a\ \underbrace{p \not\sim a \wedge \forall p'.\ a \leq p' \wedge (p' \sim a \vee p' \sim p)}_{\psi}\,.$$

The word $\varpi = 3, 0, 3, 3, 0, 7, 7, 7, 0, 4, 0, 4$ belongs to $\mathcal{L}(\varphi)$. To see this, consider the valuation ν of $\circlearrowleft_X\ \exists a\ \psi$ such that $\nu(p) = 2$ and $\nu(X) = \{1, 6, 10\}$. The modality $\circlearrowleft_X$ requires its subformula to be true on three subscopes $I_1 = [1, 5], I_2 = [6, 9]$ and $I_3 = [10, 12]$ obtained by splitting the initial scope $I = [1, 12]$ on points from X.



This is indeed the case. We have $\mathbb{W}(\varpi), \nu[a \leftarrow 1], I_1 \models \psi$, because a is the minimal element of I_1 and for every $p' \in I_1$ the data value $\varpi[p']$ belongs to the set $\{\varpi[0], \varpi[1]\} = \{0, 3\}$ containing the values at positions a and p. Analogously, we obtain $\mathbb{W}(\varpi), \nu[a \leftarrow 6], I_2 \models \psi$ and $\mathbb{W}(\varpi), \nu[a \leftarrow 10], I_3 \models \psi$.

▶ **Theorem 12.** *$\mathcal{L}\big(\mathrm{NRA}^{\mathrm{W}}[\Sigma, \mathbb{A}]\big) = \mathcal{L}(\mathrm{MSO}_{\mathrm{S}}[<, \Sigma, \mathbb{A}])$ for any finite Σ and any atoms $\mathbb{A}$.*

▶ **Corollary 13.** *If atoms $\mathbb{A}$ have elimination of strong guessing (as per Definition 4), then $\mathcal{L}\big(\mathrm{NRA}^{\mathrm{G}}[\Sigma, \mathbb{A}]\big) = \mathcal{L}(\mathrm{MSO}_{\mathrm{S}}[<, \Sigma, \mathbb{A}])$ and satisfiability is decidable for $\mathrm{MSO}_{\mathrm{S}}[<, \Sigma, \mathbb{A}]$ if and only if language emptiness for $\mathrm{NRA}^{\mathrm{G}}[\Sigma, \mathbb{A}]$ is decidable.*

Note that language emptiness is decidable for register automata over both equality atoms and dense order atoms. Furthermore:

▶ **Lemma 14.** *Both equality atoms $\mathbb{A}_= = (\mathbb{N}, =)$ and dense order atoms $\mathbb{A}_< = (\mathbb{Q}, <)$ have elimination of strong guessing.*

Not all atoms have elimination of strong guessing; for example the random (Rado) graph atoms $\mathbb{A} = (V, E)$ do not admit it. Section 5 discusses how assumption about strong guessing elimination can be lifted at the cost of extending the logic.

4 Translations between the new formalisms and register automata

We outline the core ideas behind Theorems 7 and 12; their detailed proofs can be found in the full version of the paper [16].

4.1 Translations between Data Regular Expressions and register automata

A k-register automaton can only transport k data values across a cut in the input. The operator $\cdot_k$ of DRE is designed to reflect exactly this: a word belongs to $K \cdot_k L$ if and only if it can be split as uw such that there exists an overlap $v \in \mathbb{A}^k$ with $uv \in K$ and $v^R w \in L$. Conceptually, v is an interface that transfers k data values needed to continue recognition on the right. We provide reductions between DRE and $\mathrm{NRA}^{\mathrm{G}}$.

From expressions to automata. The translation is done with a structural induction on expressions. The only non-classical steps are the case of k-contracting concatenation $\cdot_k$ and its iterated variant $(\cdot)^{k,+}$. The core idea is to verify the existence of the appropriate k-contracted infix vv^R ($v \in \mathbb{A}^k$), the automaton guesses v nondeterministically and stores its k data values in registers. This allows us to faithfully simulate reading vv^R without seeing it in the input.

From automata to expressions. Here the main idea is to separate finite control from data value consistency. We first forget data and view a run as a word over the finite alphabet of transition rules; the set of control-consistent transition rule sequences is regular, hence described by an ordinary regular expression (via Theorem 1). We then insert data back by replacing each transition rule t by a language of words of length $2k+1$. On the data track, that language encodes the relationship between the pre-valuation of the k registers, the current data value, and the post-valuation, as prescribed by t. We ensure that the valuations are consistent using k-contracting concatenation.

4.2 Translations between Scoped MSO and register automata

We illustrate the translations underlying the proof of Theorem 12 on concrete examples, working out the main steps in each direction. These examples are meant to expose the structure of the problem while avoiding tedious details.

4.2.1 From logic to automata

Consider equality atoms $\mathbb{A}_= = (\mathbb{N}, \sim)$ and let $\Sigma = \{\circ\}$ be an unary alphabet allowing us to disregard the finite-alphabet component of the input word. Starting with a formula $\varphi \in \mathrm{MSO_s}[<, \Sigma, \mathbb{A}_=]$, our goal is to construct $\mathcal{A} \in \mathrm{NRA^W}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi)$. The construction has three main steps, in which we:

1. compile φ into a pair of formulas $\varphi_{\updownarrow}$ and $\varphi_{\leftrightarrow}$,
2. translate $\varphi_{\updownarrow}$ and $\varphi_{\leftrightarrow}$ into register automata $\mathcal{A}_{\updownarrow}$ and $\mathcal{A}_{\leftrightarrow}$, and
3. construct the final automaton $\mathcal{A}$ based on $\mathcal{A}_{\updownarrow}$ and $\mathcal{A}_{\leftrightarrow}$.

We fix the formula $\varphi \equiv \exists p\, \exists X\ \wr_X\ \exists a\ p \not\sim a \wedge \forall p'.\ a \leq p' \wedge (p' \sim a \vee p' \sim p)$ from Example 11.

4.2.1.1 Step 1: Compiling φ into a pair of formulas $\varphi_{\updownarrow}$ and $\varphi_{\leftrightarrow}$

The idea is to compile away the modality $\wr$, and decompose data value dependencies into two distinct types, both easily expressible with register automata.

Multi-track words. We slightly extend the logic to *multi-track words*. The construction uses two auxiliary tracks v_x and u_x for every top-level variable x of φ and one input word track ϖ. Intuitively, the role of u_x is to specify the set of disjoint scopes in which variable x is used, while v_x is supposed to carry the value of x relevant to the current scope. The alphabet of track v_x is data values $\mathbb{A}_=$, while the alphabet of u_x is $\{\bullet, \circ\}$. For our choice of φ, we have five tracks: v_p, u_p, v_a, u_a and ϖ. The logic is extended with track accessor terms, allowing us to write, for instance, $v_p[x] \sim \varpi[y]$ to express that the data value at position x on track v_p is equal to the one at position y on track ϖ. Let proj_ϖ be the projection function that given a multi-track word $\otimes(v_p, u_p, v_a, u_a, \varpi)$ erases auxiliary tracks and outputs just the input word track $\varpi \in \mathbb{A}_=^*$.

We compile φ into a pair of formulas $(\varphi_{\updownarrow}, \varphi_{\leftrightarrow})$. Here, we only state the crucial properties of the resulting formulas $\varphi_{\updownarrow}, \varphi_{\leftrightarrow}$; the translation is detailed in the full version [16].

▶ Property 1. The languages of formulas before and after translation satisfy

$$\mathcal{L}(\varphi) = \mathrm{proj}_{\varpi}\big(\mathcal{L}(\varphi_{\updownarrow}) \cap \mathcal{L}(\varphi_{\leftrightarrow})\big)\,.$$

Intuitively, a word ϖ belongs to $\mathcal{L}(\varphi)$ if and only if there exist values for the auxiliary tracks v_p, u_p, v_a and u_a such that $\otimes(v_p, u_p, v_a, u_a, \varpi)$ belongs to both $\mathcal{L}\big(\varphi_{\updownarrow}\big)$ and $\mathcal{L}(\varphi_{\leftrightarrow})$:

	pos 1	2	3	4	5	6	7	8	9	10	11	12
ϖ	3	0	3	3	0	7	7	7	0	4	0	4

$\models \varphi \quad \leftrightsquigarrow$

	pos 1	2	3	4	5	6	7	8	9	10	11	12
v_p	0	0	0	0	0	0	0	0	0	0	0	0
u_p	●	○	○	○	○	○	○	○	○	○	○	○
v_a	3	3	3	3	3	7	7	7	7	4	4	4
u_a	●	○	○	○	○	●	○	○	○	●	○	○
ϖ	3	0	3	3	0	7	7	7	0	4	0	4

$\models \varphi_{\updownarrow} \wedge \varphi_{\leftrightarrow}$

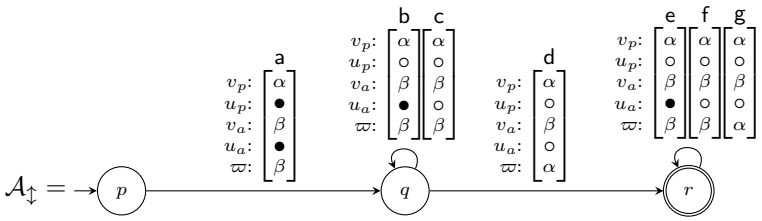
Vertical and horizontal constraints. In general, track accessor terms of the extended logic allow us to relate values of different tracks and at different positions. For instance, for a valuation ν such that $\nu(x) = 3$ and $\nu(y) = 9$, the multi-track word structure above satisfies $v_p[x] \sim \varpi[y]$ because $v_p[3] = \varpi[9] = 0$. We distinguish two special subclasses of data atomic formulas:

- *vertically-testing*: formulas relating the values of multiple data track accessors over a single position, for instance $v_p[x] \sim \varpi[x]$. These atomic formulas express local relationships – between data values at a single position.
- *horizontally-testing*: formulas relating the values from many positions, but on a single data track, for instance $v_a[x] \sim v_a[y]$, expressing non-local data value relationships.

▶ Property 2. All data atomic formulas of $\varphi_{\updownarrow}$ are vertically-testing, and all data atomic formulas of $\varphi_{\leftrightarrow}$ are horizontally-testing.

4.2.1.2 Step 2: Translating formulas into register automata $\mathcal{A}_{\updownarrow}$ and $\mathcal{A}_{\leftrightarrow}$

Constructing $\mathcal{A}_{\updownarrow}$. Due to Property 2, the language $\mathcal{L}\big(\varphi_{\updownarrow}\big)$ can be recognised by a register automaton. Indeed, each position has only finitely many quantifier-free types over the data signature. Intuitively, given position p, there are only finitely many ways the values of data tracks at position p can relate to each other. In our case, there are three data tracks v_p, v_a and ϖ, and five possible equality types of a 3-tuple of data values, namely (1) all values pairwise equal (α, α, α), (2-4) two values equal and one different from them $(\alpha, \alpha, \beta), (\alpha, \beta, \alpha), (\beta, \alpha, \alpha)$, and lastly (5) values pairwise different (α, β, γ). This allows us to translate $\varphi_{\updownarrow}$ into a formula of the standard MSO, with quantifier-free types stored in the finite alphabet. By Theorem 2, we obtain a deterministic finite automaton, which we then convert into a 0-register NRA $\mathcal{A}_{\updownarrow}$. For our chosen φ, this process yields the following $\mathcal{A}_{\updownarrow}$:



Above, the vector labelling transition a is a shorthand notation for the guard expressing the equality type (α, β, β) on data value tracks and requiring ● on finite-alphabet tracks:

$$v_a[\nabla] = \varpi[\nabla] \neq v_p[\nabla] \quad \wedge \quad u_p[\nabla] = \bullet \quad \wedge \quad u_a[\nabla] = \bullet\,;$$

other transitions are interpreted analogously.

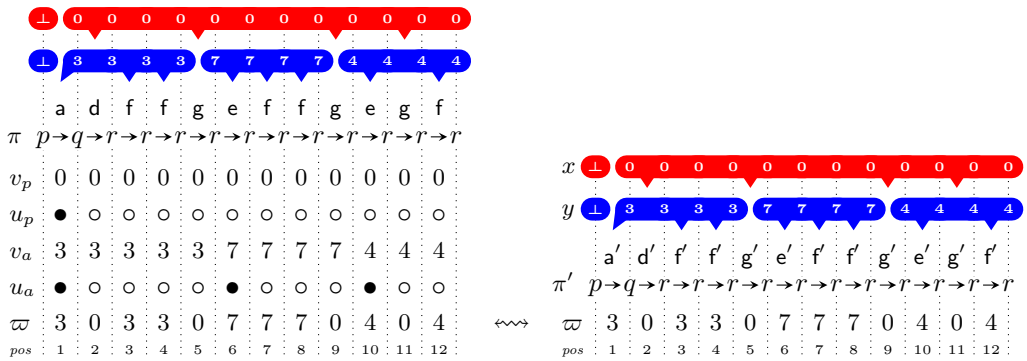
Constructing $\mathcal{A}_{\leftrightarrow}$. Conceptually, $\mathcal{A}_{\updownarrow}$ captures the "regular part" of the language $\mathcal{L}(\varphi)$, and the only missing part is the ability to equate data track values on different positions, which has to be done indirectly. For a top-level variable x and position $p \in \mathbb{N}$, marker $\circ$ at $u_x[p]$ signals that the current position should contain the same data value as $v_x[p-1]$, whereas $\bullet$ marks the beginning of a new scope with a potentially different data value on v_x. The role of $\varphi_{\leftrightarrow}$ is to enforce such rules, and its shape depends only on the set of data tracks. For our chosen φ, we obtain the following register automaton

$$\mathcal{A}_{\leftrightarrow} = \begin{pmatrix} u_p[\nabla] = \bullet \\ \wedge \\ \overset{\triangleright}{x} = v_p[\nabla] \end{pmatrix} \circlearrowright \begin{pmatrix} u_p[\nabla] = \circ \\ \wedge \\ \overset{\triangleleft}{x} = \overset{\triangleright}{x} = v_p[\nabla] \end{pmatrix} \cap \begin{pmatrix} u_a[\nabla] = \bullet \\ \wedge \\ \overset{\triangleright}{y} = v_a[\nabla] \end{pmatrix} \circlearrowright \begin{pmatrix} u_a[\nabla] = \circ \\ \wedge \\ \overset{\triangleleft}{y} = \overset{\triangleright}{y} = v_a[\nabla] \end{pmatrix},$$

where $\cap$ denotes the standard product construction for the intersection of languages of two $\mathrm{NRA}^{\mathrm{G}}$. Note that $\mathcal{A}_{\leftrightarrow}$ depends only on the set of top-level variables of the formula.

4.2.1.3 Step 3: Constructing $\mathcal{A}$

Let $\mathcal{A}_0 = \mathcal{A}_{\updownarrow} \cap \mathcal{A}_{\leftrightarrow}$. By Property 1, we have $\mathrm{proj}_{\varpi}(\mathcal{L}(\mathcal{A}_0)) = \mathcal{L}(\varphi)$. In the final step, we erase the auxiliary tracks. Finite labels are simply forgotten, whereas data values remain only in register value traces. Let $\mathcal{A}$ be an $\mathrm{NRA}^{\mathrm{W}}[\Sigma, \mathbb{A}]$ built from $\mathcal{A}_0$ by erasing all tracks but ϖ. The correspondence between accepting runs of $\mathcal{A}_0$ and $\mathcal{A}$ is illustrated below.

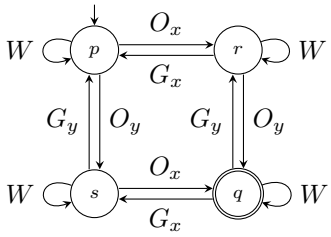


The values of the auxiliary tracks are duplicated in the register traces of $\mathcal{A}_0$; their live intervals are marked with thick red and blue lines. In $\mathcal{A}$, the data values from the erased tracks are preserved in register traces. This ensures that $\mathcal{L}(\mathcal{A}) = \mathrm{proj}_{\varpi}(\mathcal{L}(\mathcal{A}_0)) = \mathcal{L}(\varphi)$.

4.2.2 From automata to logic

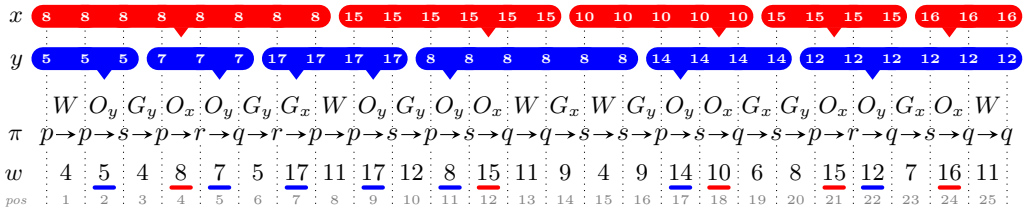
In this section, we construct the $\mathrm{MSO}_{\mathrm{s}}[<]$ formula for an example $\mathrm{NRA}^{\mathrm{W}}$. The construction illustrates the core difficulties of the translation. Fix unary $\Sigma = \{\circ\}$, so that we can omit the finite-alphabet component from transitions and words. Let $\mathbb{A} = (\mathbb{N}, R, =)$, where $R(a,b,c)$ holds for valid side lengths of some non-degenerate triangle: $a+b > c \wedge a+c > b \wedge b+c > a$. Let $\mathcal{A}$ be the following weakly-guessing $\mathrm{NRA}^{\mathrm{W}}[\Sigma, \mathbb{A}]$, with transition guards W (wait), O_r (observe the value stored in r) and G_r (guess a new value for r):

$$W \equiv R(\overset{\triangleleft}{x}, \overset{\triangleleft}{y}, \triangledown) \wedge \overset{\triangleleft}{x} = \overset{\triangleright}{x} \wedge \overset{\triangleleft}{y} = \overset{\triangleright}{y}$$
$$O_x \equiv R(\overset{\triangleleft}{x}, \overset{\triangleleft}{y}, \triangledown) \wedge \overset{\triangleleft}{x} = \overset{\triangleright}{x} \wedge \overset{\triangleleft}{y} = \overset{\triangleright}{y} \wedge \overset{\triangleleft}{x} = \triangledown$$
$$O_y \equiv R(\overset{\triangleleft}{x}, \overset{\triangleleft}{y}, \triangledown) \wedge \overset{\triangleleft}{x} = \overset{\triangleright}{x} \wedge \overset{\triangleleft}{y} = \overset{\triangleright}{y} \wedge \overset{\triangleleft}{y} = \triangledown$$
$$G_x \equiv R(\overset{\triangleleft}{x}, \overset{\triangleleft}{y}, \triangledown) \qquad \wedge \overset{\triangleright}{y} = \overset{\triangleleft}{y}$$
$$G_y \equiv R(\overset{\triangleleft}{x}, \overset{\triangleleft}{y}, \triangledown) \wedge \overset{\triangleright}{x} = \overset{\triangleleft}{x}$$



Observe that $\mathcal{A}$ does not compare the current letter $\triangledown$ with register values $\overset{\triangleright}{x}$ and $\overset{\triangleright}{y}$ after taking the transition. Any NRA$^{\mathrm{W}}$ can be brought to such form at the cost of doubling the number of registers: we only need to maintain the invariant $\overset{\triangleleft}{x'} = \overset{\triangleright}{x}$ between the values of register x and its copy x', a comparison such as $R(\overset{\triangleright}{x}, \triangledown, \overset{\triangleleft}{y})$ translates to $R(\overset{\triangleleft}{x'}, \triangledown, \overset{\triangleleft}{y})$.

Live intervals of registers and their witnesses. Below, there is an accepting run π of $\mathcal{A}$ over word $w = 4, 5, 4, 8, \ldots$, with the register values specified for each inter-letter position. Live intervals are marked with thick coloured lines. As $\mathcal{A}$ is weakly-guessing, every live interval features at least one *witness*: a position on which the register value is equal to the data word letter. Witnesses are marked with coloured triangles and underlines.

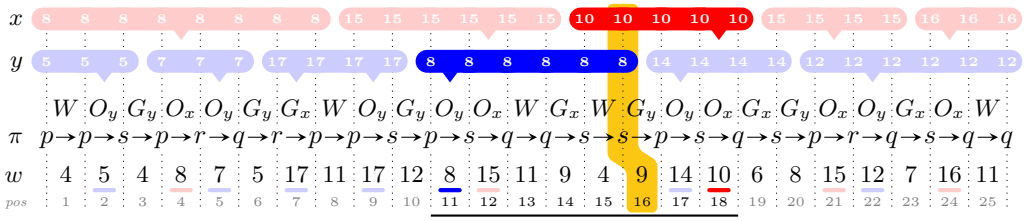


Defining the existence of an accepting run. In $\mathrm{MSO}_{\mathrm{s}}[<, \Sigma, \mathbb{A}]$ we can define a path in the graph of $\mathcal{A}$ from initial to final state in the standard way, as a sequence of transition rules: for each of the twelve rules, we existentially quantify a set of positions P_i over which that rule is applied. The language of $\mathcal{A}$ is then defined by

$$\exists P_1 \ \ldots \ \exists P_{12}.\ \mathrm{Run}_{\mathcal{A}}(\bar{P}) \wedge \mathrm{Data}_{\mathcal{A}}(\bar{P})\,,$$

where $\mathrm{Run}_{\mathcal{A}}(\bar{P})$ says that (i) every position has exactly one associated transition rule, (ii) consecutive transition rules have matching states, and (iii) the first rule starts in Q_{ini}, and the last ends in Q_{fin}. This is identical to how automata runs are defined in standard MSO[<]. It remains to define a formula $\mathrm{Data}_{\mathcal{A}}(\bar{P})$ to ensure that the run encoding $\bar{P}$ satisfies the data relations at every position.

Verifying data relations for a single position. Let us first consider a simpler task of verifying data relations at a single fixed position p. For instance, for $p = 16$, we need to verify that R holds between values 9 (current letter) and 10, 8 (register values before position 16, coming from positions 11 and 18).



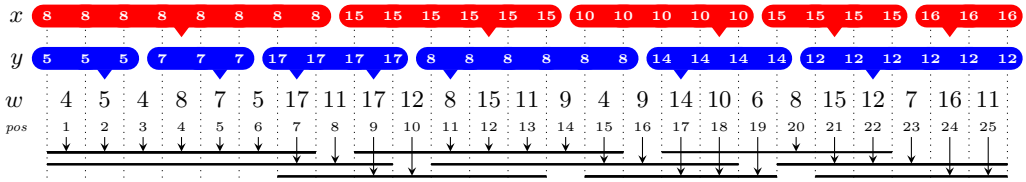
Encoding that in $\mathrm{MSO_s}[<,\mathbb{A}]$ is straightforward if both live intervals are within the scope (in our example: if the scope contains at least $[11,18]$). Indeed, we can write the following formula $\mathrm{DataLoc}_{\mathcal{A}}(p,\bar{P})$:

$$\exists p_x, a_x, b_x,\ p_y, a_y, b_y.\ \mathrm{Live}_{\mathcal{A},x}(a_x,p_x,b_x,\bar{P}) \wedge \mathrm{Live}_{\mathcal{A},y}(a_y,p_y,b_y,\bar{P}) \wedge p \in [a_x, b_x+1] \wedge p \in [a_y, b_y+1] \wedge R(p_x,p_y,p)\,,$$

where $\mathrm{Live}_{\mathcal{A},x}(a_x,p_x,b_x,\bar{P})$ is an $\mathrm{MSO}[<]$-definable formula asserting that a_x and b_x are the endpoints of a live interval of register x, and the register value is found at the position $p_x \in [a_x,b_x]$, given the run encoding $\bar{P}$. For instance, in the above run, $\mathrm{Live}_{\mathcal{A},x}$ holds for $a_x = 15$, $b_x = 18$ and $p_x = 18$. With the register value witnesses extracted, verifying register consistency amounts to asserting the data value relation $R(p_x,p_y,p)$.

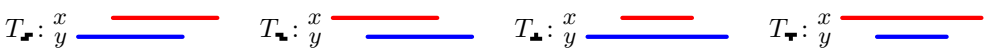
Verifying data relations for all positions: a challenge. Recall that our goal is to define $\mathrm{Data}_{\mathcal{A}}(\bar{P})$ that verifies data relations for all positions. We cannot simply define $\mathrm{Data}_{\mathcal{A}}(\bar{P}) \equiv \forall p\ \mathrm{DataLoc}_{\mathcal{A}}(p,\bar{P})$, as this is outside $\mathrm{MSO_s}[<,\Sigma,\mathbb{A}]$: variables p_x, p_y become nested and the atomic formula $R(p_x,p_y,p)$ does not satisfy Condition 8. Solving this requires a careful use of the modality $\wr$ instead of the universal quantifier.

Operating intervals. Above, verifying register relations for position 16 required having the interval $[11,18]$ contained within the current scope. In the same vein, we can determine ranges required for all word positions; we call them *operating intervals*. Every operating interval is a union of two overlapping live intervals.



A crucial observation is that the number of operating intervals incident with a word position p is bounded, so a fixed number of $\wr$ modalities is enough to assert the correctness of register relations within every operating interval. We proceed by showing that property in a systematic way.

Types of operating intervals. There are four ways the beginnings and the ends of two overlapping live intervals can be ordered: order types $T_{\lrcorner}, T_{\llcorner}, T_{\urcorner}$ and $T_{\ulcorner}$:

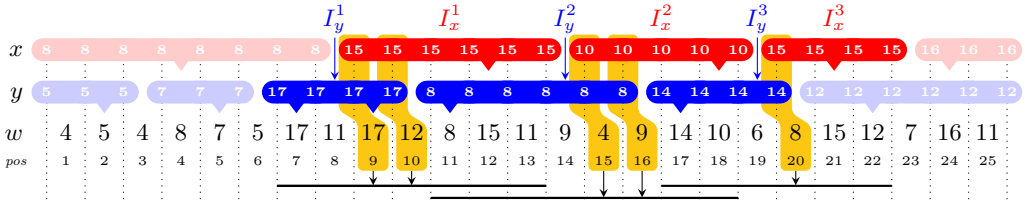


We construct the formula $\mathrm{Data}_{\mathcal{A}}(\bar{P})$ using a case disjunction on active interval shapes

$$\mathrm{Data}_{\mathcal{A}}(\bar{P}) \equiv \mathrm{Data}_{\mathcal{A},\lrcorner}(\bar{P}) \wedge \mathrm{Data}_{\mathcal{A},\llcorner}(\bar{P}) \wedge \mathrm{Data}_{\mathcal{A},\ulcorner}(\bar{P}) \wedge \mathrm{Data}_{\mathcal{A},\urcorner}(\bar{P})\,,$$

where formula $\mathrm{Data}_{\mathcal{A},t}(\bar{P})$ verifies register relations for all positions whose two live intervals combining to the operating interval have the order type T_t. We focus on $\mathrm{Data}_{\mathcal{A},\lrcorner}(\bar{P})$ and $\mathrm{Data}_{\mathcal{A},\urcorner}(\bar{P})$, as the remaining two cases are symmetrical.

Constructing the formula $\mathrm{Data}_{\mathcal{A},\blacktriangleright}(\bar{P})$. Live intervals contributing to type $T_{\blacktriangleright}$ operating intervals are highlighted below:



There are three type-$T_{\blacktriangleright}$ operating intervals: $J_1 = I_x^1 \cup I_y^1$, $J_2 = I_x^2 \cup I_y^2$ and $J_3 = I_x^3 \cup I_y^3$. Observe that J_1 and J_3 have an empty intersection. This property is not accidental: let $(J_i)_{i\in[L]}$ be a sequence of operating intervals of type $T_{\blacktriangleright}$, where each $J_i = I_x^i \cup I_y^i$. For every J_i, J_{i+1}, J_{i+2}, the intersection $J_i \cap J_{i+2}$ is empty, because

$$\max J_i = \max I_x^i < \min I_x^{i+1} - 1 \leq \max I_y^{i+1} < \min I_y^{i+2} = \min J_{i+2}\,.$$

Thus, the sequence decomposes into two subsequences of pairwise disjoint operating intervals: $S_{\blacktriangleright}^0 = (J_{2i})_{i\in[\lfloor L/2\rfloor]}$ and $S_{\blacktriangleright}^1 = (J_{2i+1})_{i\in[\lceil L/2\rceil]}$, of even and odd indices of (J_i). These sequences are definable in $\mathrm{MSO_S}[<]$. Let $\mathrm{EvenM}_{\mathcal{A},\blacktriangleright}(X,\bar{P})$ be a formula that verifies that X is the set of minimal elements of intervals from $S_{\blacktriangleright}^0$, given a run encoding $\bar{P}$, and similarly $\mathrm{OddM}_{\mathcal{A},\blacktriangleright}(Y,\bar{P})$ for $S_{\blacktriangleright}^1$. We define $\mathrm{Data}_{\mathcal{A},\blacktriangleright}(\bar{P})$ as follows

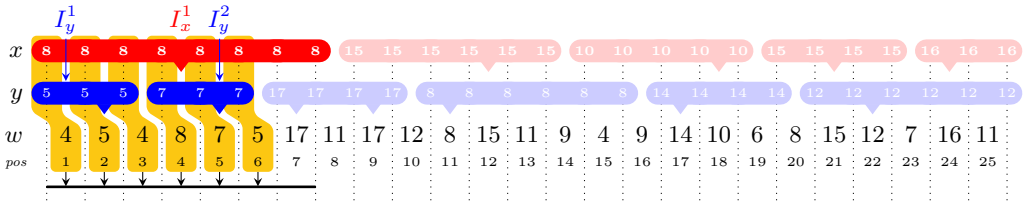
$$\exists X\ \exists Y.\ \mathrm{EvenM}_{\mathcal{A},\blacktriangleright}(X,\bar{P}) \wedge \mathrm{OddM}_{\mathcal{A},\blacktriangleright}(Y,\bar{P}) \wedge (\partial_X\ \mathrm{Data}'_{\mathcal{A},\blacktriangleright}(\bar{P})) \wedge (\partial_Y\ \mathrm{Data}'_{\mathcal{A},\blacktriangleright}(\bar{P}))\,,$$

where $\mathrm{Data}'_{\mathcal{A},\blacktriangleright}(\bar{P})$ verifies data consistency for all positions in the intersection of live intervals:

$$\exists p_x, a_x, b_x,\ p_y, a_y, b_y.\ \mathrm{Live}_{\mathcal{A},x}(a_x,p_x,b_x,\bar{P}) \wedge \mathrm{Live}_{\mathcal{A},y}(a_y,p_y,b_y,\bar{P}) \wedge \mathrm{Type}_{\blacktriangleright}(a_x,b_x,a_y,b_y) \wedge \forall p.\ p \in [a_x, b_y+1] \to R(p_x,p_y,p)\,,$$

where, $\mathrm{Type}_{\blacktriangleright}$ verifies the order type. Note that p_x and p_y are not nested in $\mathrm{Data}_{\mathcal{A}}(\bar{P})$.

Constructing the formula $\mathrm{Data}_{\mathcal{A},\blacktriangledown}(\bar{P})$. The case of $T_{\blacktriangledown}$ is slightly different than one of $T_{\blacktriangleright}$. Within one operating interval, there can be more than one live interval of y. Below, the sole operating interval $J = I_x^1 = [1,7]$ of type $T_{\blacktriangledown}$ contains two different live intervals of y.



Unlike the case of $T_{\blacktriangleright}$, operating intervals of type $T_{\blacktriangledown}$ are pairwise disjoint. Let $\mathrm{M}_{\mathcal{A},\blacktriangledown}(X,\bar{P})$ be a formula verifying that X is the set of minimal elements of all type-$T_{\blacktriangledown}$ operating intervals. We define $\mathrm{Data}_{\mathcal{A},\blacktriangledown}(\bar{P})$ as follows:

$$\exists X.\ \mathrm{M}_{\mathcal{A},\blacktriangledown}(X,\bar{P}) \wedge (\partial_X\ \mathrm{Data}'_{\mathcal{A},\blacktriangledown}(X,\bar{P}))\,,$$

where $\mathrm{Data}'_{\mathcal{A},\blacktriangledown}(X,\bar{P})$ finds the value of x and recursively uses modality ∂ to verify data value consistency within every live interval of y separately, assuming we have a formula $\mathrm{M}_{\mathcal{A},y}$ to select their minimal endpoints:

$$\exists p_x, a_x, b_x,\ p_y, a_y, b_y\ \exists Y.\ \mathrm{Live}_{\mathcal{A},x}(a_x, p_x, b_x, \bar{P}) \wedge \mathrm{Live}_{\mathcal{A},y}(a_y, p_y, b_y, \bar{P}) \wedge \mathrm{Type}_{\blacktriangledown}(a_x, b_x, a_y, b_y) \wedge \mathrm{M}_{\mathcal{A},y}(Y, a_x, b_x, \bar{P}) \wedge (\eth_Y\ \mathrm{Live}_{\mathcal{A},y}(a'_y, p'_y, b'_y) \wedge \forall p.\ p \in [a'_y, b'_y + 1] \to R(p_x, p'_y, p))\,.$$

In the above formula, a_y and b_y are endpoints of *some* live interval of y combining with the unique live interval of x that yields the order type $T_{\blacktriangledown}$. We need to verify R *within all* such intervals separately, hence the use of $\eth$ and a'_y and b'_y. This completes our example.

Generalisation. The recursion becomes more convoluted when the input register automaton has more than two registers, but the analysis of order types $T_{\blacktriangledown}$ and $T_{\blacktriangledown}$ illustrates most of the conceptual difficulties. In the k-register general case, operating intervals are still formed from two live intervals: one extending the most to the left and the other the most to the right. Each live interval can belong to one of k registers, which increases the number of types beyond four. Using modality $\eth$, we essentially express "for all operating intervals corresponding to registers r_i and r_j", which allows us to properly define positions p_{r_i} and p_{r_j} on which r_i and r_j have their value. This way, we have dealt with two registers and can recursively deal with the remaining ones. At the bottom of the recursion, we have access to top level variables (p_r) holding correct values for every register r, and we can verify the transition constraints.

5 Extensions and restrictions of Scoped MSO

5.1 A logic equivalent to register automata over infinite words

Fix a finite alphabet Σ and atoms $\mathbb{A}$. An *infinite* word over $\Sigma \times \mathbb{A}$ is a sequence $(a_i)_{i \in \mathbb{N}_+}$ with $a_i \in \Sigma \times \mathbb{A}$. We write $(\Sigma \times \mathbb{A})^\omega$ for the set of *infinite data words*. The notion of a data word structure extends verbatim to this setting by taking the set of positions to be $\mathbb{N}_+$.

Register automata require no syntactic changes for ω-words; only the acceptance condition is replaced by the standard *Büchi* condition. For $\mathcal{A} \in \mathrm{NRA}^{\mathrm{W}}[\Sigma, \mathbb{A}]$, we define $\mathcal{L}^\omega(\mathcal{A})$ as the set of ω-words admitting an infinite run that starts in an initial configuration and visits accepting states from Q_{fin} infinitely often. The semantics of $\mathrm{MSO_s}[<, \Sigma, \mathbb{A}]$ is unchanged over infinite data word structures: formulas are interpreted over positions $\mathbb{N}_+$, with scopes now being intervals of $\mathbb{N}_+$ as before.

▷ Claim 15. For every finite Σ and atoms $\mathbb{A}$, $\mathcal{L}^\omega(\mathrm{MSO_s}[<, \Sigma, \mathbb{A}]) = \mathcal{L}^\omega\big(\mathrm{NRA}^{\mathrm{W}}[\Sigma, \mathbb{A}]\big)$.

Indeed, accepting runs of $\mathrm{NRA}^{\mathrm{W}}$ over infinite words and MSO-definable as in Section 4.2.2; only $\mathrm{Run}(\bar{P})$ needs to be adapted to express the Büchi acceptance condition.

For the converse direction, as in Section 4.2.1, we can translate φ to formulas $\varphi_{\updownarrow}$ and $\varphi_{\leftrightarrow}$ such that $\mathcal{L}^\omega(\varphi) = \mathrm{proj}_{\varpi}\big(\mathcal{L}^\omega(\varphi_{\updownarrow}) \cap \mathcal{L}^\omega(\varphi_{\leftrightarrow})\big)$. Hence, by the ω-regular analogue of the Büchi–Elgot–Trakhtenbrot theorem [17], $\mathcal{L}^\omega\big(\varphi_{\updownarrow}\big)$ is recognised by a register automaton with the Büchi acceptance condition.

5.2 A logic equivalent to register automata with strong guessing

A modified logic $\mathrm{MSO_s^G}[<, \Sigma, \mathbb{A}]$ can capture $\mathrm{NRA^G}$ languages even for atoms $\mathbb{A}$ without elimination of strong guessing. Weak guessing is essential in our main logical characterisation: $\mathrm{MSO_s}[<, \Sigma, \mathbb{A}]$ can only refer to data values that occur in the input word, whereas a strongly guessing $\mathrm{NRA^G}$ may introduce fresh data values that never appear on the input track. To accommodate such behaviour, one can enrich the multi-track variant of $\mathrm{MSO_s}[<, \Sigma, \mathbb{A}]$

(described in Section 4.2.1) with top-level existential quantification over *data tracks*. Formally, we extend the syntax with quantifiers of the form $\exists t[\cdot]\ \varphi$, where $t[\cdot]$ is a new track-accessor term ranging over $\mathbb{A}$.

Semantics. Let $u_1, \ldots, u_k \in \Sigma^L$ and $v_1, \ldots, v_\ell \in \mathbb{A}^L$ be length-L tracks. Then we have $\mathbb{W}(u_1, \ldots, u_k, v_1, \ldots, v_\ell), \nu, I \models \exists t[\cdot]\ \varphi$ whenever $\mathbb{W}(u_1, \ldots, u_k, v_1, \ldots, v_\ell, v'), \nu, I \models \varphi$ for some $v' \in \mathbb{A}^L$, where inside φ the term $t[x]$ is interpreted as $v'[\nu(x)]$ for each first-order variable x. Intuitively, $\exists t[\cdot]$ equips the logic with a source of fresh data values, mirroring strong guesses performed by the automaton.

▷ Claim 16. $\mathcal{L}\big(\mathrm{MSO}^{\mathrm{G}}_{\mathrm{S}}[<, \Sigma, \mathbb{A}]\big) = \mathcal{L}\big(\mathrm{NRA}^{\mathrm{G}}[\Sigma, \mathbb{A}]\big)$ for any finite Σ and arbitrary atoms $\mathbb{A}$.

5.3 A logic equivalent to register automata without guessing

A simple refinement of Scoped MSO suffices to capture the formalism of $\mathrm{NRA}^{\mathrm{N}}$. We write $\mathrm{MSO}^{\mathrm{N}}_{\mathrm{S}}[<, \Sigma, \mathbb{A}]$ for the variant obtained by replacing the scope modality with a *scoping quantifier* $\partial_{X,y}$ and adapting the notion of top-level variables accordingly:

- $\mathbb{W}, \nu, I \models \partial_{X,y}\ \varphi$ iff for every subscope $I' = [a, b] \in \mathrm{subs}(I, \nu(X))$ we have $\mathbb{W}, \nu[y \leftarrow a], I' \models \varphi$. In the logic-to-automata translation, scopes correspond to live intervals of registers in the run. The variable y is bound by $\partial_{X,y}$ as the initial position in every sub-interval. This corresponds to the beginning of a live interval of an $\mathrm{NRA}^{\mathrm{N}}$, enforcing the "no-guessing" discipline: register values are always read from the input each time a new live interval begins.
- a first-order variable is *top-level* if it is bound by some $\partial_{X,y}$ that does not occur under negation in the syntax tree, and it is *nested* otherwise;
- the well-formedness condition on data atoms is unchanged: every occurrence of $R(\bar{x})$ may contain at most one nested variable.

▷ Claim 17. $\mathcal{L}\big(\mathrm{MSO}^{\mathrm{N}}_{\mathrm{S}}[<, \Sigma, \mathbb{A}]\big) = \mathcal{L}\big(\mathrm{NRA}^{\mathrm{N}}[\Sigma, \mathbb{A}]\big)$ for any finite Σ and arbitrary atoms $\mathbb{A}$.

References

1. Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4):702–715, 2010. Fundamentals of Computation Theory. doi:10.1016/j.tcs.2009.10.009.
2. Mikołaj Bojańczyk. Slightly infinite sets. Draft of a book, version of September 11, 2019, 2019. URL: https://www.mimuw.edu.pl/~bojan/paper/atom-book.
3. Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Logic*, 12(4), 2011. doi:10.1145/1970398.1970403.
4. Mikołaj Bojańczyk and Rafał Stefański. Single-use automata and transducers for infinite alphabets. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 113:1–113:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.113.
5. Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, Volume 10, Issue 3, August 2014. doi:10.2168/LMCS-10(3:4)2014.

6 Paul Brunet and Alexandra Silva. A kleene theorem for nominal automata. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 107:1–107:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2019.107.

7 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960. doi:10.1002/malq.19600060105.

8 Thomas Colcombet, Clemens Ley, and Gabriele Puppis. Logics with rigidly guarded data tests. *Log. Methods Comput. Sci.*, 11(3), 2015. doi:10.2168/LMCS-11(3:10)2015.

9 Calvin C Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, 1961.

10 Michael Kaminski and Nissim Francez. Finite-memory automata. *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 683–688 vol.2, 1990. URL: https://api.semanticscholar.org/CorpusID:205095872.

11 Michael Kaminski and Tony Tan. Regular expressions for languages over infinite alphabets. *Fundamenta Informaticae*, 69:301–318, 2006. URL: https://api.semanticscholar.org/CorpusID:1411996.

12 Stephen Cole Kleene. *Representation of events in nerve nets and finite automata*, volume 34. Princeton University Press Princeton, 1956.

13 Bartek Klin, Sławomir Lasota, and Szymon Toruńczyk. Nondeterministic and co-nondeterministic implies deterministic, for data languages. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures*, pages 365–384, Cham, 2021. Springer International Publishing.

14 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004. doi:10.1145/1013560.1013562.

15 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2013.

16 Radosław Piórkowski. Scoped MSO, register automata, and expressions: Equivalence over data words, 2026. doi:10.48550/arXiv.2602.13120.

17 J. Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In Ernest Nagel, Patrick Suppes, and Alfred Tarski, editors, *Logic, Methodology and Philosophy of Science*, volume 44 of *Studies in Logic and the Foundations of Mathematics*, pages 1–11. Elsevier, 1966. doi:10.1016/S0049-237X(09)70564-6.

18 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In Zoltán Ésik, editor, *Computer Science Logic*, pages 41–57, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11874683_3.

19 B. A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *Sib. Mat. Zh.*, 3:103–131, 1962.