A Linear-Time Nominal μ-Calculus with Name Allocation

Daniel Hausmann
Gothenburg University, Göteborg, Sweden

Stefan Milius
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Lutz Schröder
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

Logics and automata models for languages over infinite alphabets, such as Freeze LTL and register automata, serve the verification of processes or documents with data. They relate tightly to formalisms over nominal sets, such as nondeterminininistic orbit-finite automata (NOFAs), where names play the role of data. Reasoning problems in such formalisms tend to be computationally hard. *Name-binding* nominal automata models such as *regular nondeterministic nominal automata (RNNAs)* have been shown to be computationally more tractable. In the present paper, we introduce a linear-time fixpoint logic Bar-μTL for finite words over an infinite alphabet, which features full negation and freeze quantification via name binding. We show by a nontrivial reduction to *extended regular nondeterministic nominal automata* that even though Bar-μTL allows unrestricted nondeterminism and unboundedly many registers, model checking Bar-μTL over RNNAs and satisfiability checking both have elementary complexity. For example, model checking is in 2ExpSpace, more precisely in parametrized ExpSpace, effectively with the number of registers as the parameter.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking

Keywords and phrases Model checking, linear-time logic, nominal sets

Digital Object Identifier 10.4230/LIPIcs.MFCS.2021.58

Related Version *Extended Version*: `https://arxiv.org/abs/2010.10912`

Funding *Daniel Hausmann*: Supported by Deutsche Forschungsgemeinschaft (DFG) under project MI 717/7-1 and by the European Research Council (ERC) under project 772459.
Stefan Milius: Supported by Deutsche Forschungsgemeinschaft (DFG) under project MI 717/7-1.
Lutz Schröder: Supported by Deutsche Forschungsgemeinschaft (DFG) under project SCHR 1118/15-1.

1 Introduction

There has been longstanding interest in logics and automata models over infinite alphabets, such as the classical register automaton model [24] and Freeze LTL (e.g. [11,31]), or automata models over nominal sets [36] such as nondeterministic orbit-finite automata [2], in which *names* play the role of letters. Infinite alphabets may be seen as representing data. For example, nonces in cryptographic protocols [30], data values in XML documents [35], object identities [18], or parameters of method calls [23] can all be usefully understood as letters in infinite alphabets. A central challenge in dealing with infinite alphabets is that key decision problems in many logics and automata models are either undecidable or of prohibitively high complexity unless drastic restrictions are imposed (see the related work section). In a nutshell, the contribution of the present work is the identification of a linear-time fixpoint logic Bar-μTL for *finite* words over infinite alphabets that

- allows both safety and liveness constraints (via fixpoints, thus in particular going beyond the expressiveness of LTL [17]) as well as full nondeterminism, and e.g. expresses the language "some letter occurs twice" (which cannot be accepted by deterministic or unambigous register automata [2,37]);
- imposes no restriction on the number of registers; and
- is closed under complement;

and nevertheless allows model checking and satisfiability checking in elementary complexity: Bar-μTL model checking over regular nondeterministic nominal automata [38] (in the sense of checking that all words accepted by a given automaton satisfy a given formula) is in 2ExpSpace and more precisely in parametrized ExpSpace, with the maximal size of the support of states as the parameter (in the translation of nominal automata to register automata [2,38], this corresponds to the number of registers); and satisfiability checking is in ExpSpace (and in parametrized PSpace).

The tradeoff that buys this comparatively low complexity is a mild recalibration of the notion of freshness, which we base on explicit binding of names in strings in a nominal language model; depending on the exact discipline of α-renaming of bound names, one obtains either global freshness (w.r.t. all previous letters, as in session automata [3]) or local freshness (w.r.t. currently stored letters, as in register automata). This principle has been previously employed in the semantics of nominal Kleene algebra [16,28] and in regular non-deterministic nominal automata (RNNAs) [38]. It carries the limitation that in the local freshness variant, letters can be required to be distinct only from such previous letters that are expected to be seen again – a restriction that seems reasonable in applications; that is, in many situations, one presumably would not need to insist on an object identifier, nonce, or process name b to be distinct from a previous name a if a is never going to be used again.

We introduce a dedicated finite-word automaton model, *extended regular nondeterministic nominal automata (ERNNAs)*, which extend RNNAs by $\top$*-states*, i.e. deadlocked universal states. We then base our results on mutual translations between Bar-μTL and ERNNAs; the tableau-style logic-to-automata translation turns out to be quite nontrivial as it needs to avoid the accumulation of renamed copies of subformulae in automata states. Since RNNAs are, under global freshness, essentially equivalent to session automata [3], one consequence of our results is that session automata (which as such are only closed under *resource-bounded* complementation [3]) can be made closed under complement simply by allowing $\top$-states.

Proofs are mostly omitted or only sketched; full proofs are in the extended version [22].

Related work. *Automata:* Over infinite alphabets, the expressive power of automata models generally increases with the power of control (deterministic/nondeterministic/alternating) [25]. In deterministic models, language inclusion can often be decided in reasonable complexity; this remains true for unambiguous register automata [5,34]. For nondeterministic register automata and the equivalent nondeterministic orbit-finite automata [2], emptiness is decidable but inclusion is undecidable unless one restricts to at most two registers [24]. Similarly, language inclusion (equivalently nonemptiness) of alternating register automata is undecidable unless one restricts to at most one register, and even then is not primitive recursive [11]. Automata models for infinite words outside the register paradigm include data walking automata [33], whose inclusion problem is decidable even under nondeterminism but at least as hard as reachability in Petri nets (equivalently vector addition systems) [7], as well as the highly expressive data automata [1], whose nonemptiness problem is decidable but, again, at least as hard as Petri net reachability. Note that by recent results [9], Petri net reachability is not elementary, and in fact Ackermann-complete [10,32].

Logics: Bar-μTL is incomparable to Freeze LTL. Satisfiability checking of Freeze LTL is by reduction to alternating register automata, and has the same high complexity even for the one-register case [11]. Satisfiability in the *safety* fragment of Freeze LTL over infinite words [31] is ExpSpace-complete if the number of registers is restricted to at most one, while the refinement and model checking problems are decidable but not primitive recursive (all three problems become undecidable in presence of more than one register). The *μ-calculus with atoms* [26] is interpreted over Kripke models with atoms; its satisfiability problem is undecidable while its model checking problem is decidable, with the complexity analysis currently remaining open. It is related to the very expressive *first-order μ-calculus* [19,20], for which model checking is only known to be semidecidable. The *μ-calculus over data words* [6,8] works in the data walking paradigm. The satisfiability problem of the full calculus is undecidable; that of its ν-fragment, which translates into data automata, is decidable but elementarily equivalent to Petri net reachability. *Variable LTL* [21] extends LTL with a form of first-order quantification over data domains. The full language is very expressive and in particular contains full Freeze LTL [39]. Some fragments have decidable satisfiability or model checking problems (typically not both) [21,39], occasionally in elementary complexity; these impose prenex normal form (reducing expressiveness) and restrict to quantifier prefixes that fail to be stable under negation. Decidable fragments will, of course, no longer contain full Freeze LTL; how their expressiveness compares to Bar-μTL needs to be left open at present. *Flat* Freeze LTL [12] interdicts usage of the freeze quantifier in safety positions (e.g. the freeze quantifier can occur under F but not under G); its existential model checking problem over (infinite runs of) one-counter automata is NExpTime-complete, while the universal model checking problem is undecidable [4].

2 Preliminaries: Nominal Sets

Nominal sets offer a convenient formalism for dealing with names and freshness; for our present purposes, names play the role of data. We briefly recall basic notions and facts (see [36] for more details).

Fix a countably infinite set $\mathbb{A}$ of *names*, and let G denote the group of finite permutations on $\mathbb{A}$, which is generated by the *transpositions* $(a\,b)$ for $a \neq b \in \mathbb{A}$ (recall that $(a\,b)$ just swaps a and b); we write 1 for the neutral element of G. A (left) *action* of G on a set X is a map $(-)\cdot(-)\colon G \times X \to X$ such that $1 \cdot x = x$ and $\pi \cdot (\pi' \cdot x) = (\pi\pi') \cdot x$ for all $x \in X$, $\pi, \pi' \in G$. For instance, G acts on $\mathbb{A}$ by $\pi \cdot a = \pi(a)$. A set $S \subseteq \mathbb{A}$ is a *support* of $x \in X$ if $\pi(x) = x$ for all $\pi \in G$ such that $\pi(a) = a$ for all $a \in S$. We say that x is *finitely supported* if x has some finite support, and *equivariant* if the empty set is a support of x, i.e. $\pi \cdot x = x$ for all $\pi \in G$. The *orbit* of $x \in X$ is the set $\{\pi \cdot x \mid \pi \in G\}$. The set of all orbits forms a partition of X, and X is *orbit-finite* if there are finitely many orbits.

A *nominal set* is a set X equipped with an action of G such that every element of X is finitely supported; e.g. $\mathbb{A}$ itself is a nominal set. An early motivating example of a nominal set is the set of λ-terms with variable names taken from $\mathbb{A}$, and with the action of $\pi \in G$ given by replacing every variable name a with $\pi(a)$ as expected, e.g. $(a\,b) \cdot \lambda a.ba = \lambda b.ab$. Every element x of a nominal set has a least finite support, denoted $\mathsf{supp}(x)$, which one may roughly think of as the set of names occurring ("freely", see below) in x. A name $a \in \mathbb{A}$ is *fresh* for x if $a \notin \mathsf{supp}(x)$. On subsets $A \subseteq X$ of a nominal set X, G acts by $\pi \cdot A = \{\pi \cdot x \mid x \in A\}$. Thus, $A \subseteq X$ is equivariant iff $\pi \cdot A \subseteq A$ for all $\pi \in G$. Similarly, A has support S iff $\pi \cdot A \subseteq A$ whenever $\pi(a) = a$ for all $a \in S$. We say that A is *uniformly finitely supported* if $\bigcup_{x \in A} \mathsf{supp}(x)$ is finite [40], in which case A is also finitely supported [14, Thm. 2.29].

(The converse does not hold, e.g. the set $\mathbb{A}$ is finitely supported but not uniformly finitely supported.) Uniformly finitely supported subsets of orbit-finite sets are always finite but in general, uniformly finitely supported sets can be infinite; e.g. for finite $B \subseteq \mathbb{A}$, the set $B^* \subseteq \mathbb{A}^*$ is uniformly finitely supported.

The Cartesian product $X \times Y$ of nominal sets X, Y is a nominal set under the componentwise group action; then, $\mathsf{supp}(x, y) = \mathsf{supp}(x) \cup \mathsf{supp}(y)$. Given a nominal set X equipped with an equivalence relation $\sim$ that is equivariant as a subset of $X \times X$, the quotient $X/\!\sim$ is a nominal set under the group action $\pi \cdot [x]_\sim = [\pi \cdot x]_\sim$. A key role in the technical development is played by *abstraction sets*, which provide a semantics for binding mechanisms [15]:

▶ **Definition 2.1** (Abstraction set)**.** Given a nominal set X, an equivariant equivalence relation $\sim$ on $\mathbb{A} \times X$ is defined by

$$(a, x) \sim (b, y) \qquad \text{iff} \qquad (a\,c) \cdot x = (b\,c) \cdot y \text{ for some } c \in \mathbb{A} \text{ that is fresh for } (a, x, b, y)$$

(equivalently for all such c). The *abstraction set* $[\mathbb{A}]X$ is the quotient set $(\mathbb{A} \times X)/\!\sim$. The $\sim$-equivalence class of $(a, x) \in \mathbb{A} \times X$ is denoted by $\langle a \rangle x \in [\mathbb{A}]X$.

We may think of $\sim$ as an abstract notion of α-equivalence, and of $\langle a \rangle$ as binding the name a. Indeed we have $\mathsf{supp}(\langle a \rangle x) = \mathsf{supp}(x) \setminus \{a\}$, as expected in binding constructs.

3 Data Languages and Bar Languages

As indicated, we use the set $\mathbb{A}$ of names as the data domain, and capture freshness of data values via α-equivalence, roughly as follows. We work with words over $\mathbb{A}$ where names may be preceded by the bar symbol "|", which indicates that the next letter is bound until the end of the word (cf. Remark 3.2); such words are called *bar strings* [38]. Bar strings may be seen as patterns that govern how letters are read from the input word; broadly speaking, an occurrence of $|a$ corresponds to reading a letter from the input word, and binding this letter to the name a, while an undecorated occurrence of a means that the letter referred to by a occurs in the input word. (We loosely speak of the input word as consisting of letters, and of bar strings as consisting of names; formally, however, letters and names are the same, viz., elements of $\mathbb{A}$.) Bound names can be renamed, giving rise to a notion of α-equivalence; as usual, the new name needs to be sufficiently *fresh*, i.e. cannot already occur freely in the scope of the binding. For instance, in $a|bab$, the | binds the letter b in $|bab$. The bar string $a|bab$ is α-equivalent to $a|cac$ but not, of course, to $a|aaa$. That is, we can rename the bound name b into c but not into a, as a already occurs freely in $|bab$; we say that renaming b into a is *blocked*.

We will see that bar strings modulo α-equivalence relate to formalisms for *global* freshness (a name is globally fresh if it has never been seen before), such as session automata [3]. Contrastingly, if we regard a bar string as representing all words over $\mathbb{A}$ that arise by performing some α-equivalent renaming and then removing the bars, we arrive at a notion of *local* freshness, similar to freshness w.r.t. currently stored names as in register automata; precise definitions are given later in this section. For instance, the bar string $|a|ba$ represents the set of words $\{cdc \mid c, d \in \mathbb{A}, c \neq d\} \subseteq \mathbb{A}^*$ under both local and global freshness semantics – in $|a|ba$, a and b cannot be renamed into the same letter, since a occurs freely in the scope of $|b$. Contrastingly, $|a|b$ represents the set $\{cd \mid c \neq d\} \subseteq \mathbb{A}^*$ under global freshness semantics, but under local freshness semantics it just represents the set $\mathbb{A}^2$ of all two-letter words, since $|a|b$ is α-equivalent to $|a|a$. The impossibility of expressing the language $\{cd \mid c \neq d\}$ under local freshness is thus hardwired into our language model. We emphasize again that this

restriction seems reasonable in practice, since one may expect that freshness of new letters is often relevant only w.r.t. letters that are intended to be seen again later, e.g. in deallocation statements or message acknowledgements. Formal definitions are as follows.

▶ **Definition 3.1** (Bar strings)**.** We put $\overline{\mathbb{A}} = \mathbb{A} \cup \{\mathfrak{l}a \mid a \in \mathbb{A}\}$; we refer to elements $\mathfrak{l}a \in \overline{\mathbb{A}}$ as *bar names*, and to elements $a \in \overline{\mathbb{A}}$ as *plain names*. A *bar string* is a word $w = \sigma_1\sigma_2 \cdots \sigma_n \in \overline{\mathbb{A}}^*$, with *length* $|w| = n$; we denote the empty string by ϵ. We turn $\overline{\mathbb{A}}$ into a nominal set by putting $\pi \cdot a = \pi(a)$ and $\pi \cdot \mathfrak{l}a = \mathfrak{l}\pi(a)$; then, $\overline{\mathbb{A}}^*$ is a nominal set under the pointwise action of G. We define *α-equivalence* on bar strings to be the least equivalence $\equiv_\alpha$ such that

$$w\mathfrak{l}av \equiv_\alpha w\mathfrak{l}bu \quad \text{whenever} \quad \langle a\rangle v = \langle b\rangle u \text{ in } [\mathbb{A}]\,\overline{\mathbb{A}}^*$$

(Definition 2.1) for $w, v, u \in \overline{\mathbb{A}}^*$, $a \in \mathbb{A}$. Thus, $\mathfrak{l}a$ binds a, with scope extending to the end of the word. Correspondingly, a name a is *free* in a bar string w if there is an occurrence of a in w that is to the left of any occurrence of $\mathfrak{l}a$. We write $[w]_\alpha$ for the α-equivalence class of $w \in \overline{\mathbb{A}}^*$ and $\mathsf{FN}(w) = \{a \in \mathbb{A} \mid a \text{ is free in } w\}$ $(= \mathsf{supp}([w]_\alpha))$ for the set of free names of w. If $\mathsf{FN}(w) = \emptyset$, then w is *closed*. A bar string w is *clean* if all bar names $\mathfrak{l}a$ in w are pairwise distinct and have $a \notin \mathsf{FN}(w)$. For a set $S \subseteq \mathbb{A}$ of names, we write $\mathsf{bs}(S) = \{w \in \overline{\mathbb{A}}^* \mid \mathsf{FN}(w) \subseteq S\}$.

▶ Remark 3.2. Closed bar strings are essentially the same as the *well-formed symbolic words* that appear in the analysis of session automata [3]. Indeed, symbolic words consist of operations that read a letter into a register, corresponding to bar names, and operations that require seeing the content of some register in the input, corresponding to plain names. Symbolic words are normalized by a register allocation procedure similar to α-renaming. Well-formedness of symbolic words corresponds to closedness of bar strings.

Moreover, modulo the respective equational laws, bar strings coincide with the *ν-strings* [28, 29] that appear in the semantics of *Nominal Kleene Algebra (NKA)* [16]; cf. [38]. These are constructed from names in $\mathbb{A}$, sequential composition, and a binding construct $\nu a.\, w$, which binds the name a in the word w. In particular, the equational laws of ν-strings allow extruding the scope of every ν to the end of the word after suitable α-renaming. We note that Bar-μTL and its associated automata models are more expressive than NKA as they express languages with unbounded nesting of binders [38].

We will work with three different types of languages:

▶ **Definition 3.3.**

1. *Data languages* are subsets of $\mathbb{A}^*$.
2. *Literal languages* are subsets of $\overline{\mathbb{A}}^*$, i.e. sets of bar strings.
3. *Bar languages* are subsets of $\overline{\mathbb{A}}^*/\equiv_\alpha$, i.e. sets of α-equivalence classes of bar strings.

A bar language L is *closed* if $\mathsf{supp}(L) = \emptyset$.

Bar languages are the natural semantic domain of our formalims, and relate tightly to data languages as discussed next. A key factor in the good computational properties of regular nominal nondeterministic automata (RNNA) [38] is that the bar languages they accept (cf. Section 5) are uniformly finitely supported, and we will design Bar-μTL to ensure the same property. Note that a uniformly supported bar language is closed iff it consists of (equivalence classes of) closed bar strings. For brevity, we will focus the exposition on target formulae (in model checking) and automata that denote or accept, respectively, closed bar languages, with free names appearing only in languages accepted by non-initial states or denoted by proper subformulae of the target formula. (The treatment is easily extended to

bar languages with free names; indeed, such globally free names are best seen as a separate finite alphabet of constant symbols [29].) We will occasionally describe example bar languages as regular expressions over $\overline{\mathbb{A}}$ (i.e. as *regular bar expressions* [38]), meaning the set of all α-equivalence classes of instances of the expression.

To convert bar strings into data words, we define $\mathsf{ub}(a) = \mathsf{ub}(|a) = a$ and extend ub to bar strings letterwise; i.e. $\mathsf{ub}(w)$ is the data word obtained by erasing all bars "|" from w. We then define two ways to convert a bar language L into a data language:

$$N(L) = \{\mathsf{ub}(w) \mid [w]_\alpha \in L, w \text{ clean}\} \qquad \text{and} \qquad D(L) = \{\mathsf{ub}(w) \mid [w]_\alpha \in L\}.$$

That is, N is a global freshness interpretation of |, while D provides a local freshness interpretation as exemplified above; e.g. as indicated above we have $D(|a|ba) = N(|a|ba) = \{aba \mid a, b \in \mathbb{A}, a \neq b\}$, while $N(|a|b) = \{ab \mid a, b \in \mathbb{A}, a \neq b\}$ but $D(|a|b) = \{ab \mid a, b \in \mathbb{A}\}$.

▶ Remark 3.4. In fact, the operator N is injective on closed bar languages, because ub is injective on closed clean bar strings [37, 38]. This means that bar language semantics and global freshness semantics are essentially the same, while local freshness semantics is a quotient of the other semantics. It is immediate from [37, Lemma A.4] that N preserves intersection and complement of closed bar languages, the latter in the sense that $N(\mathsf{bs}(\emptyset) \setminus L) = \mathbb{A}^* \setminus N(L)$ for closed bar languages L. Both properties fail for the local freshness interpretation D; the semantics of formulae should therefore be understood first in terms of bar languages, with D subsequently applied globally.

4 Syntax and Semantics of Bar-μTL

We proceed to introduce a variant Bar-μTL of linear temporal logic whose formulae define bar languages. This logic relates, via its local freshness semantics, to Freeze LTL. It replaces freeze quantification with name binding modalities, and features fixpoints, for increased expressiveness in comparison to the temporal connectives of LTL [17]. Via global freshness semantics, Bar-μTL may moreover be seen as a logic for session automata [3].

Syntax. We fix a countably infinite set V of *(fixpoint) variables*. The set Bar of *bar formulae* $\phi, \psi, \ldots$ (in negation normal form) is generated by the grammar

$$\phi, \psi := \epsilon \mid \neg\epsilon \mid \phi \wedge \psi \mid \phi \vee \psi \mid \heartsuit_\sigma \phi \mid X \mid \mu X.\, \phi,$$

where $\heartsuit \in \{\Diamond, \Box\}$, $\sigma \in \overline{\mathbb{A}}$ and $X \in \mathsf{V}$. We define $\top = \epsilon \vee \neg\epsilon$ and $\bot = \epsilon \wedge \neg\epsilon$. We refer to $\Diamond_\sigma$ and $\Box_\sigma$ as *σ-modalities*. The meaning of the Boolean operators is standard; the fixpoint construct μ denotes unique fixpoints, with uniqueness guaranteed by a guardedness restriction to be made precise in a moment. The other constructs are informally described as follows. The constant ϵ states that the input word is empty, and $\neg\epsilon$ that the input word is nonempty. A formula $\Diamond_a \phi$ is read "the first letter is a, and the remaining word satisfies ϕ", and $\Box_a \phi$ is read dually as "if the first letter is a, then the remaining word satisfies ϕ". The reading of $|a$-modalities is similar but involves α-renaming as detailed later in this section; as indicated in Section 3, this means that $|a$-modalities effectively read fresh letters. They thus replace the freeze quantifier; one important difference with the latter is that $\Diamond_{|a}$ consumes the letter it reads, i.e. advances by one step in the input word. A name a is *free* in a formula ϕ if ϕ contains an a-modality at a position that is not in the scope of any $|a$-modality; that is, $|a$-modalities bind the name a. We write $\mathsf{FN}(\phi)$ for the set of free names in ϕ, and $\mathsf{BN}(\phi)$ for the set of *bound names* in ϕ, i.e. those names a such that ϕ

mentions $\mathsf{I}a$; we put $\mathsf{N}(\phi) = \mathsf{FN}(\phi) \cup \mathsf{BN}(\phi)$, and (slightly generously) define the *degree* of ϕ to be $\mathsf{deg}(\phi) = |N(\phi)|$. We write $\mathsf{FV}(\phi)$ for the set of *free* fixpoint variables in ϕ, defined in the standard way by letting μX bind X; a formula ϕ is *closed* if $\mathsf{FV}(\phi) = \emptyset$. (We refrain from introducing terminology for formulae without free *names*.) As indicated above we require that all fixpoints $\mu X.\,\phi$ are *guarded*, that is, all free occurrences of X lie within the scope of some σ-modality in ϕ. We denote by $\mathsf{cl}(\phi)$ the *closure* of ϕ in the standard sense [27], i.e. the least set of formulae that contains ϕ and is closed under taking immediate subformulae and unfolding top-level fixpoints; this set is finite. We define the *size* of ϕ as $|\phi| = |\mathsf{cl}(\phi)|$.

For purposes of making Bar a nominal set, we regard every fixpoint variable X with enclosing fixpoint expression $\mu X.\,\phi$ as being annotated with the set $A = \mathsf{FN}(\mu X.\,\phi)$; that is, we identify X with the pair (X, A). We then let G act by replacing names in the obvious way; i.e. $\pi \cdot \phi$ is obtained from ϕ by replacing a with $\pi(a)$, $\mathsf{I}a$ with $\mathsf{I}\pi(a)$, and (X, A) with $(X, \pi \cdot A)$ everywhere. Otherwise, the definition is as expected:

▶ **Definition 4.1.** *α-Equivalence* $\equiv_\alpha$ on formulae is the congruence relation generated by

$$\Diamond_{\mathsf{I}a}\phi \equiv_\alpha \Diamond_{\mathsf{I}b}\psi \text{ and } \Box_{\mathsf{I}a}\phi \equiv_\alpha \Box_{\mathsf{I}b}\psi \quad \text{whenever } \langle a\rangle\phi = \langle b\rangle\psi$$

(cf. Definition 2.1).

▶ Remark 4.2. The point of implicitly annotating fixpoint variables with the free names of the enclosing μ-expression is to block unsound α-renamings: It ensures that, e.g., $\Diamond_{\mathsf{I}a}(\mu X.\,(\Diamond_a\epsilon \vee \Diamond_{\mathsf{I}b}X))$ is *not* α-equivalent to $\Diamond_{\mathsf{I}a}(\mu X.\,(\Diamond_a\epsilon \vee \Diamond_{\mathsf{I}a}X))$ (as X is actually $(X, \{a\})$), and is required to ensure stability of α-equivalence under fixpoint expansion, recorded next. We note that fixpoint expansion does *not* avoid capture of names; e.g. the expansion of $\mu X.\,(\Diamond_a\epsilon \vee \Diamond_{\mathsf{I}a}X)$ is $\Diamond_a\epsilon \vee \Diamond_{\mathsf{I}a}(\mu X.\,(\Diamond_a\epsilon \vee \Diamond_{\mathsf{I}a}X))$.

▶ **Lemma 4.3.** *Let $\mu X.\,\phi \equiv_\alpha \mu X.\,\phi'$. Then $\phi[\mu X.\,\phi/X] \equiv_\alpha \phi'[\mu X.\,\phi'/X]$.*

Proof. Immediate from the fact that by the convention that fixpoint variables are annotated with the free names of their defining formulae, X, $\mu X.\,\phi$, and $\mu X.\,\phi'$ have the same free names and hence allow the same α-renamings in the outer contexts ϕ and ϕ', respectively. ◀

Semantics. We interpret each bar formula ϕ as denoting a uniformly finitely supported bar language, depending on a *context*, i.e. a finite set $S \subseteq \mathbb{A}$ such that $\mathsf{FN}(\phi) \subseteq S$, which specifies names that are allowed to occur freely; at the outermost level, S will be empty (cf. Section 3). The context grows when we traverse modalities $\Diamond_{\mathsf{I}a}$ or $\Box_{\mathsf{I}a}$. Correspondingly, we define satisfaction $S, w \models \phi$ of a formula ϕ by a bar string $w \in \mathsf{bs}(S)$ recursively by the usual clauses for the Boolean connectives, and

$$\begin{array}{lcl}
S, w \models \neg\epsilon & \Leftrightarrow & w \neq \epsilon \\
S, w \models \epsilon & \Leftrightarrow & w = \epsilon \\
S, w \models \mu X.\,\phi & \Leftrightarrow & S, w \models \phi[\mu X.\,\phi/X] \\
S, w \models \Diamond_a\phi & \Leftrightarrow & \exists v.\, w = av \text{ and } S, v \models \phi \\
S, w \models \Box_a\phi & \Leftrightarrow & \forall v.\, \text{if } w = av \text{ then } S, v \models \phi \\
S, w \models \Diamond_{\mathsf{I}a}\phi & \Leftrightarrow & \exists \psi \in \mathsf{Bar}, v \in \overline{\mathbb{A}}^*, b \in \mathbb{A}. \\
 & & w \equiv_\alpha \mathsf{I}bv \text{ and } \langle a\rangle\phi = \langle b\rangle\psi \text{ and } S \cup \{b\}, v \models \psi \\
S, w \models \Box_{\mathsf{I}a}\phi & \Leftrightarrow & \forall \psi \in \mathsf{Bar}, v \in \overline{\mathbb{A}}^*, b \in \mathbb{A}. \\
 & & \text{if } w \equiv_\alpha \mathsf{I}bv \text{ and } \langle a\rangle\phi = \langle b\rangle\psi \text{ then } S \cup \{b\}, v \models \psi.
\end{array}$$

Guardedness of fixpoint variables guarantees that on the right hand side of the fixpoint clause, $\mu X.\,\phi$ is evaluated only on words that are strictly shorter than w, so the given clause uniquely defines the semantics. Notice that $\Diamond_{|a}$ and $\Box_{|a}$ allow α-renaming of both the input word and the formula; we comment on this point in Remark 4.7. For a formula ϕ such that $\mathsf{FN}(\phi) = \emptyset$, we briefly write

$$[\![\phi]\!]_0 = \{w \in \mathsf{bs}(\emptyset) \mid \emptyset, w \models \phi\} \qquad \text{and} \qquad [\![\phi]\!] = [\![\phi]\!]_0/{\equiv_\alpha},$$

referring to $[\![\phi]\!]_0$ as the *literal language* and to $[\![\phi]\!]$ as the *bar language* of ϕ (variants with non-empty context and $\mathsf{FN}(\phi) \neq \emptyset$ are technically unproblematic but require more notation). In particular, $[\![\phi]\!]$ is closed by construction. The *global* and *local freshness semantics* of ϕ are $N([\![\phi]\!])$ and $D([\![\phi]\!])$, respectively, where N and D are the operations converting bar languages into data languages described in Section 3.

▶ Remark 4.4. In Bar-μTL, fixpoints take on the role played by the temporal operators in Freeze LTL. In bar language semantics, the overall mode of expression in Bar-μTL, illustrated in Example 4.8, is slightly different from that of Freeze LTL, as in Bar-μTL the input is traversed using modalities tied to specific letters rather than using a *next* operator $\bigcirc$. In local freshness semantics, the effect of $\bigcirc$ is included in the name binding modality $\Diamond_{|a}$. For instance, in local freshness semantics we can express LTL-style formulae $\phi\, U\, \psi$ ('ϕ until ψ') as $\mu X.\,\psi \vee (\phi \wedge \Diamond_{|a} X)$. In particular, $\mu X.\,\epsilon \vee \Diamond_{|a} X$ defines the universal data language, so $\top$ is not actually needed in local freshness semantics. Overall, Freeze LTL and Bar-μTL (with local freshness semantics) intersect as indicated but are incomparable: On the one hand, Freeze LTL can express the language "the first two letters are different", which as indicated in Section 3 is not induced by a bar language. On the other hand, Bar-μTL features fixpoints, which capture properties that generally fail to be expressible using LTL operators, e.g. the language of all even-length words. The latter point relates to the fact that even over finite alphabets, LTL on finite words is only as expressive as first-order logic, equivalently star-free regular expressions (cf. [17]). Constrastingly, thanks to the fixpoint operators, Bar-μTL is as expressive as its corresponding automata model (Theorem 6.6).

▶ Remark 4.5. As indicated previously, Bar-μTL is closed under complement: By taking negation normal forms, we can define $\neg\phi$ so that $S, w \models \neg\phi$ iff $S, w \not\models \phi$.

We note next that literal languages of formulae are closed under α-equivalence, and that α-equivalent renaming of formulae indeed does not affect the semantics (cf. Remark 4.2):

▶ **Lemma 4.6.** *For $\phi, \psi \in \mathsf{Bar}$, $a \in \mathbb{A}$, $S \subseteq \mathbb{A}$, and $w, w' \in \overline{\mathbb{A}}^*$, we have:*
1. *If $S, w \models \psi$ and $w \equiv_\alpha w'$, then $S, w' \models \psi$.*
2. *If $S, w \models \psi$ and $\phi \equiv_\alpha \phi'$, then $S, w' \models \phi'$.*

The proof is by induction along the recursive definition of the semantics; the case for fixpoints in Claim 2 is by Lemma 4.3.

▶ Remark 4.7. We have noted above that the semantics allows α-renaming of both words and formulae. Let us refer to an alternative semantics where the definition of $S, w \models \Diamond_{|a}\phi$ is modified to require that there exists $w \equiv_\alpha |av$ such that $S \cup \{a\}, v \models \phi$ (without allowing α-renaming of $\Diamond_{|a}\phi$), similarly for $\Box_{|a}$, as the *rigid* semantics, and to the semantics defined above as the *actual* semantics. The rigid semantics is not equivalent to the actual semantics, and has several flaws. First off, claim 2 of the above Lemma 4.6 fails under the rigid semantics, in which, for example,

$$\emptyset, |b|ab \models \Diamond_{|b}\Diamond_{|a}\top \qquad \text{but} \qquad \emptyset, |b|ab \not\models \Diamond_{|b}\Diamond_{|b}\top$$

(the latter because $\{b\}, \mathsf{|}ab \not\models \Diamond_{\mathsf{|}b}\top$, as α-renaming of $\mathsf{|}a$ into $\mathsf{|}b$ is blocked in $\mathsf{|}ab$). More importantly, the rigid semantics has undesirable effects in connection with fixpoints. For instance, in the actual semantics, the formula $\phi = \mu X.\,((\neg\epsilon \wedge \Box_{\mathsf{|}a}\bot) \vee \Diamond_{\mathsf{|}a} X)$ has the intuitively intended meaning: A bar string satisfies ϕ iff it contains some plain name. In the rigid semantics, however, we unexpectedly have $\emptyset, \mathsf{|}a\mathsf{|}bab \not\models \phi$; to see this, note that $\{a\}, \mathsf{|}bab \not\models \phi$ in the rigid semantics, since α-renaming of $\mathsf{|}b$ into $\mathsf{|}a$ is blocked in $\mathsf{|}bab$.

▶ **Example 4.8.** We consider some Bar-μTL formulae and their respective semantics under local and global freshness. (The local freshness versions are expressible in Freeze LTL in each case; recall however Remark 4.4.)

1. The bar language $[\![\top]\!]$ is the set of all closed bar strings (modulo α-equivalence, a qualification that we omit henceforth). Under both global and local freshness semantics, this becomes the set of all data words.
2. The bar language $[\![\Diamond_{\mathsf{|}a}\Box_a\epsilon]\!]$ is the language of all closed bar strings that start with a bar name $\mathsf{|}a$, and stop after the second letter if that letter exists and is the plain name a (e.g. $[\![\Diamond_{\mathsf{|}a}\Box_a\epsilon]\!]$ contains $\mathsf{|}a$, $\mathsf{|}aa$, $\mathsf{|}a\mathsf{|}bab$ but not $\mathsf{|}aaa$). In both global and local freshness semantics, this becomes the language of all words that stop after the second letter if that letter exists and coincides with the first letter.
3. In context $\{a\}$, a bar string satisfies $\mu Y.\,((\Diamond_{\mathsf{|}b} Y) \vee \Diamond_a \top)$ iff it contains a free occurrence of a preceded only by bar names distinct from $\mathsf{|}a$. Thus, the bar language of

   $$\mu X.\,(\Diamond_{\mathsf{|}a}(X \vee \mu Y.\,((\Diamond_{\mathsf{|}b} Y) \vee \Diamond_a \top)))$$

   consists of all closed bar strings that start with a prefix of bar names and eventually mention a plain name corresponding to one of these bar names. Under both local and global freshness semantics, this becomes the data language of all words mentioning some letter twice (which is not acceptable by deterministic or even unambiguous register automata [2, 37]). Notice that during the evaluation of the formula, the context can become unboundedly large, as it grows every time a bar name is read.
4. The bar language of the similar formula

   $$\mu X.\,((\Diamond_{\mathsf{|}a} X) \vee (\Diamond_{\mathsf{|}a} \mu Y.\,((\Diamond_{\mathsf{|}b} Y) \vee \Diamond_a \epsilon)))$$

   consists of all closed bar strings where all names except the last one are bound names. Under global freshness semantics, this becomes the data language where the last letter occurs *precisely* twice in the word, and all other names only once. Under local freshness semantics, the induced data language is that of all words where the last letter occurs *at least* twice, with no restrictions on the other letters.
5. To illustrate both the mechanism of local freshness via α-equivalence and, once again, the use of $\top$, we consider the bar language of

   $$\Diamond_{\mathsf{|}a}\Diamond_{\mathsf{|}b}\mu X.\,((\Diamond_{\mathsf{|}b} X) \vee \Diamond_a\Diamond_b \top),$$

   which consists of all closed bar strings that start with a bar name $\mathsf{|}a$, at some later point contain a substring $\mathsf{|}bab$, and have only bar names distinct from $\mathsf{|}a$ in between. Under global freshness semantics, this becomes the data language of all words where the first name a occurs a second time at the third position or later, all letters are mutually distinct until that second occurrence, and the letter preceding that occurrence is repeated immediately after. The local freshness semantics is similar but only requires the letters between the first and second occurrence of a to be distinct from a (rather than mutually distinct), that is, the substring bab is required to contain precisely the second occurrence of a.

5 Extended Regular Nondeterministic Nominal Automata

We proceed to introduce the nominal automaton model we use in model checking, *extended regular nondeterministic nominal automata* (ERNNAs), a generalized version of RNNAs [38] that allow for limited alternation in the form of deadlocked universal states.

Nominal automata models [2] generally feature nominal sets of states; these are infinite as sets but typically required to be orbit-finite. RNNAs are distinguished from other nominal automata models (such as *nondeterministic orbit-finite automata* [2]) in that they impose finite branching but feature *name-binding* transitions; that is, they have *free* transitions $q \xrightarrow{a} q'$ for $a \in \mathbb{A}$ as well as *bound* transitions $q \xrightarrow{!a} q'$, both consuming the respective type of letter in the input bar string w. Bound transitions may be understood as reading fresh letters. RNNAs are a nondeterministic model, i.e. accept w if there *exists* a run on w ending in an accepting state. ERNNAs additionally feature $\top$*-states* that accept the current word even if it has not been read completely, and thus behave like the formula $\top$; these states may be seen as universal states without outgoing transitions. Formal definitions are as follows.

▶ **Definition 5.1.** An *extended regular nondeterministic nominal automaton (ERNNA)* is a four-tuple $A = (Q, \to, s, f)$ that consists of

- an orbit-finite nominal set Q of *states* (whose orbits we also refer to as the *orbits of* A);
- an *initial state* $s \in Q$ such that $\mathsf{supp}(s) = \emptyset$;
- an equivariant *transition* relation $\to \subseteq Q \times (\overline{\mathbb{A}} \cup \{\epsilon\}) \times Q$, with $(q, \sigma, q') \in \to$ denoted by $q \xrightarrow{\sigma} q'$; and
- an equivariant *acceptance* function $f \colon Q \to \{0, 1, \top\}$

such that $\to$ is α*-invariant* (that is, $q \xrightarrow{!a} q'$ and $\langle a \rangle q' = \langle b \rangle q''$ imply $q \xrightarrow{!b} q''$) and finitely branching up to α-equivalence (i.e. for each q, the sets $\{(a, q') \mid q \xrightarrow{a} q'\}$, $\{(\epsilon, q') \mid q \xrightarrow{\epsilon} q'\}$, and $\{\langle a \rangle q' \mid q \xrightarrow{!a} q'\}$ are finite). Whenever $f(q) = \top$, we require $\mathsf{supp}(q) = \emptyset$ and moreover that q is a deadlock, i.e. there are no transitions of the form $q \xrightarrow{\sigma} q'$. The *degree* $\mathsf{deg}(A)$ of A is the maximal size of the support of a state in Q (in the translation of nominal automata into register automata, the degree corresponds to the number of registers [2, 38]). A state q is *accepting* if $f(q) = 1$, *non-accepting* if $f(q) = 0$, and a $\top$*-state* if $f(q) = \top$.

We extend the transition relation to words w over $\overline{\mathbb{A}}$, i.e. to bar strings, as usual; that is, $q \xrightarrow{w} q'$ iff there exist states $q = q_0, q_1, \dots, q_k = q'$ and transitions $q_i \xrightarrow{\sigma_{i+1}} q_{i+1}$ for $i = 0, \dots, k-1$ such that w is the concatenation $\sigma_1 \cdots \sigma_k$, where σ_i is regarded as a one-letter word if $\sigma_i \in \overline{\mathbb{A}}$, and as the empty word if $\sigma_i = \epsilon$. We define $L_{\mathsf{pre}}(A) \subseteq \overline{\mathbb{A}}^* \times \{1, \top\}$ (for *prelanguage*) as

$$L_{\mathsf{pre}}(A) = \{(w, f(q)) \mid s \xrightarrow{w} q, f(q) \in \{1, \top\}\}.$$

The *literal language* $L_0(A) \subseteq \overline{\mathbb{A}}^*$ accepted by an ERNNA A then is defined by

$$L_0(A) = \mathsf{bs}(\emptyset) \cap \big(\{w \mid (w, 1) \in L_{\mathsf{pre}}(A)\} \cup \{vu \mid (v, \top) \in L_{\mathsf{pre}}(A), u \in \overline{\mathbb{A}}^*\}\big);$$

that is, a closed bar string w is literally accepted if either w has a run ending in an accepting state or a prefix of w has a run ending in a $\top$-state. (Again, extending the treatment to bar strings with free names is technically unproblematic but heavier on notation.) The *bar language accepted by* A is the quotient

$$L_\alpha(A) = L_0(A)/{\equiv_\alpha}.$$

We say that A is ϵ*-free* if A contains no ϵ-transitions. If A is ϵ-free and contains no $\top$-states, then A is a *regular nondeterministic nominal automaton (RNNA)*.

▶ Remark 5.2. The presence of $\top$-states makes ERNNAs strictly more expressive than RNNAs under bar language semantics (equivalently, under global freshness semantics). Indeed, the ERNNA consisting of a single $\top$-state accepts the universal bar language, which is not acceptable by an RNNA [38]. On the other hand, under local freshness semantics, an accepting state with a $|a$-self-loop accepts the universal data language (in analogy to the expressibility of $\top$ by $\mu X.\, \epsilon \vee \Diamond_{|a} X$ in Bar-μTL under local freshness semantics, cf. Remark 4.4), so RNNAs are as expressive as ERNNAs under local freshness semantics.

Name dropping. Like for RNNAs, the literal language accepted by an ERNNA is not in general closed under α-equivalence. However, one can adapt the notion of name dropping [38] to ERNNA: Roughly speaking, an ERNNA is *name-dropping* if all its transitions may nondeterministically lose any number of names from the support of states (which corresponds to losing register contents in a register automaton). The literal language of a name-dropping ERNNA is closed under α-equivalence, and every ERNNA A can be transformed into a name-dropping ERNNA $\mathsf{nd}(A)$, preserving the bar language. This transformation is central to the inclusion checking algorithm (see additional remarks in Section 7).

Representing ERNNAs. ERNNAs are, prima facie, infinite objects; we next discuss a finite representation of ERNNAs as *extended bar NFAs*, generalizing the representation of RNNAs as bar NFAs [38]. The intuition behind extended bar NFAs is similar to that of ERNNAs, except that extended bar NFAs are not closed under name permutation. In particular, extended bar NFAs feature deadlocked universal states:

▶ **Definition 5.3.** An *extended bar NFA* $A = (Q, \to, s, f)$ consists of
- a finite set Q of *states*;
- a *transition relation* $\to \subseteq (Q \times \overline{\mathbb{A}} \times Q)$, with $(q, \sigma, q') \in \to$ denoted by $q \xrightarrow{\sigma} q'$;
- an *initial* state $s \in Q$; and
- an *acceptance* function $f \colon Q \to \{0, 1, \top\}$

such that whenever $f(q) = \top$, then q is a deadlock, i.e. has no outgoing transitions. We extend the transition relation to words over $\overline{\mathbb{A}}$ (including the empty word) as usual. Similarly as for ERNNAs, we define $L_{\mathsf{pre}}(A) \subseteq \overline{\mathbb{A}}^* \times \{1, \top\}$ by

$$L_{\mathsf{pre}}(A) = \{(w, f(q)) \mid s \xrightarrow{w} q, f(q) \in \{1, \top\}\}.$$

The *literal language* $L_0(A)$ accepted by A is

$$L_0(A) = \{w \mid (w, 1) \in L_{\mathsf{pre}}(A)\} \cup \{vu \mid (v, \top) \in L_{\mathsf{pre}}(A), u \in \overline{\mathbb{A}}^*\}.$$

The *bar language* of A is then defined as the quotient $L_\alpha(A) = L_0(A)/{\equiv_\alpha}$. For ease of presentation, we only consider the case where $L_\alpha(A)$ is closed, which is easily checked syntactically (no a may be reached in A without passing $|a$). We generally write (A, q) for the extended bar NFA that arises by making $q \in Q$ the initial state of A, dropping however the requirement that the bar language accepted by (A, q) is closed. The set $\mathsf{FN}(A, q)$ of *free names* of $q \in Q$ is then $\mathsf{FN}(A, q) = \bigcup_{(w,b) \in L_{\mathsf{pre}}(A,q)} \mathsf{FN}(w)$. Slightly sharpening the original definition [38], we take the *degree* of A to be $\mathsf{deg}(A) := \max_{q \in Q} |\mathsf{FN}(A, q)|$.

▶ **Theorem 5.4.** *ERNNAs and extended bar NFAs accept the same bar languages; that is:*
1. *For a given extended bar NFA with n states and degree k, there exists a name-dropping ERNNA of degree k with $n \cdot 2^k$ orbits that accepts the same bar language.*
2. *For a given ERNNA with n orbits and degree k, there exists an extended bar NFA of degree k with $n \cdot k!$ states that accepts the same bar language.*

The key algorithmic task on ERNNAs is inclusion checking; we generalize the inclusion algorithm for RNNAs [38] to obtain

▶ **Theorem 5.5.** *Given a bar NFA A_1 and an extended bar NFA A_2, the inclusion $L_\alpha(A_1) \subseteq L_\alpha(A_2)$ can be checked using space polynomial in the number of orbits of A_1 and A_2, and exponential in $\mathsf{deg}(A_1)$ and $\mathsf{deg}(A_2)$. The same holds under local freshness semantics.*

6 Equivalence of Bar-μTL and ERNNA

Our model checking algorithm will be based on translation of closed formulae into ERNNAs, in what amounts to a tableau construction that follows a similar spirit as the standard automata-theoretic translation of LTL, but requires a special treatment of $\Box$-formulae and $\neg\epsilon$, and moreover uses nondeterminism to bound the number of free names in automata states (which may be thought of as the number of registers) by guessing certain names, as explained in the following example.

▶ **Example 6.1.** Consider the formulae $\phi(b) = \mu Y.\,(\Box_b \bot \wedge \Box_{|c} Y)$ and $\psi = \mu X.\,(\Box_{|a} X \wedge \Box_{|b}\phi(b))$. The formula $\phi(b)$ states that the first plain name that occurs is not a free occurrence of b, and ψ thus states that none of the bar names have a free occurrence later on. When evaluating ψ over a bar string $w = |a_1|a_2 \ldots |a_n a_i v$ consisting of n bar names $|a_i$ followed by the plain name a_i $(1 \leq i \leq n)$ and a remaining bar string v (so w does not satisfy ψ), one eventually has to evaluate all the formulae $\phi(a_1), \ldots, \phi(a_n)$ over the bar string $a_i v$. Thus, the number of copies of formulae that a naively constructed ERRNA for ψ needs to keep track of can in principle grow indefinitely. At a first glance, this seems to prohibit a translation of formulae into orbit-finite automata. However, we observe that when the letter a_i is read, all $\phi(a_j)$ for $i \neq j$ immediately evaluate to $\top$ (since the conjuncts $\Box_{a_j}\bot$ and $\Box_{|c}\phi(a_j)$ of their fixpoint unfolding both hold vacuously), and only the evaluation of $\phi(a_i)$ becomes relevant (since the argument of $\Box_{a_i}\bot$ is actually evaluated). In fact, it is possible to let the ERRNA for ψ nondeterministically guess the first plain name a_i that occurs in the input bar string. Then it suffices to let the ERNNA keep track of $\phi(a_i)$ since as discussed, all other copies of $\phi(b)$ become irrelevant.

The idea from the previous example can be generalized to work for all formulae. To this end we introduce a recursive manipulation of formulae that uses annotations to explicitly restrict the support of formulae and to guess and enforce so-called distinguishing letters. When constructing an ERNNA from a formula, we will use such manipulated formulae to avoid the problem described in Example 6.1 by bounding the number of formulae that the constructed ERNNA has to track.

▶ **Definition 6.2.** Fix a marker element $* \notin \mathbb{A}$ (indicating absence of a name). Let ϕ be a formula, let B and C be sets of letters such that $B \subseteq C$, and let $a \in (C \setminus B) \cup \{*\}$. For $n \in \mathbb{N}$, we define $\phi^B_C(a)_n$ recursively (as termination measure of the recursive definition we use tuples $(|n|, \mathsf{u}(\phi), |\phi|)$, ordered by lexicographic ordering, where $\mathsf{u}(\phi)$ denotes the number of unguarded fixpoint operators in ϕ) by putting $\phi^B_C(a)_0 = \top$ and, for $n > 0$,

$$\epsilon^B_C(a)_n = \epsilon \qquad \neg\epsilon^B_C(a)_n = \neg\epsilon$$
$$(\psi \wedge \chi)^B_C(a)_n = \psi^B_C(a)_n \wedge \chi^B_C(a)_n \qquad (\psi \vee \chi)^B_C(a)_n = \psi^B_C(a)_n \vee \chi^B_C(a)_n$$
$$(\Diamond_{|b}\psi)^B_C(a)_n = \Diamond_{|b}(\psi^{B\cup\{b\}}_{C\cup\{b\}}(a)_{n-1}) \qquad (\Box_{|b}\psi)^B_C(a)_n = \Box_{|b}(\psi^{B\cup\{b\}}_{C\cup\{b\}}(a)_{n-1})$$
$$(\mu X.\,\psi)^B_C(a)_n = (\psi[\mu X.\,\psi/X])^B_C(a)_n$$

and

$$(\Diamond_b\psi)^B_C(a)_n = \begin{cases} \bot & b \notin C \\ \Diamond_b(\psi^B_C(a)_{n-1}) & b \in C, b \in B \\ \Diamond_b(\psi^{\emptyset}_{\mathsf{FN}(\psi)}(*)_{n-1}) & b \in C, b \notin B \end{cases}$$

$$(\Box_b\psi)^B_C(a)_n = \begin{cases} \top & b \notin C \\ \Box_b(\psi^B_C(a)_{n-1}) & b \in C, b \in B \\ \chi(b)^B_C(a)_{n-1} & b \in C, b \notin B \end{cases}$$

where

$$\chi(b)^B_C(a)_{n-1} = \begin{cases} \Box_b(\psi^{\emptyset}_{\mathsf{FN}(\psi)}(*)_{n-1}) & a = * \\ \epsilon \vee \Diamond_{\mathsf{lc}}\top \vee \bigvee_{d\in B} \Diamond_d\top \vee \Diamond_a(\psi^{\emptyset}_{\mathsf{FN}(\psi)}(*)_{n-1}) & a = b \\ \epsilon \vee \Diamond_{\mathsf{lc}}\top \vee \bigvee_{d\in B\cup\{a\}} \Diamond_d\top & * \neq a \neq b. \end{cases}$$

During this process, fixpoint formulae are unfolded before replacing any free modalities within their arguments; by guardedness of fixpoint variables, this happens at most n times. (We intend the number n as as a strict upper bound on the length of bar strings over which $\phi^B_C(a)_n$ is meant to be evaluated; cf. Lemma 6.3.) The process replaces just the first freely occurring boxes whose index is in C but not in B; hence, modal operators whose index comes from B are left unchanged. Intuitively, $\phi^B_C(a)_n$ is a formula that behaves like the restriction of ϕ to the support C on bar strings w such that $|w| < n$ and in which a is the first free name that is not contained in B (if any such name occurs in w; in this case we refer to the letter a as *distinguishing letter*). Hence $\phi^B_C(a)_n$ is like the restriction of ϕ to support C, assuming that the distinguishing letter is a. Formally:

▶ **Lemma 6.3.** *Let B and C be sets of names such that $B \subseteq C$, let $a \in (C \setminus B) \cup \{*\}$, let ϕ be a formula, and let S be a context. Let v be a bar string such that if $a \neq *$, then each letter that has a free occurrence in v before the first free occurrence of a in v is contained in B. Also, suppose that if there is some freely occurring letter in v that is not contained in B and the first such letter d is in $\mathsf{FN}(\phi)$, then $d \in C$. Under these assumptions, we have*

$$S, v \models \phi \iff S, v \models \phi^B_C(a)_n \qquad \text{for all } n \text{ such that } |v| < n.$$

Proof sketch. Induction along the recursive definition of $\phi^B_C(a)_n$. ◀

Let ϕ be a formula, let B, C, and D be sets of names such that $B \subseteq C$, and let $a \in (C \setminus B) \cup D \cup \{*\}$. We put

$$\phi^B_C(a)^D_n = \begin{cases} \bot & \text{if } a \in D \\ \phi^B_C(a)_n & \text{if } a \notin D, \end{cases}$$

the intuition being that $\phi^B_C(a)^D_n$ encodes ϕ (restricted to support C) together with the guess that a is the distinguishing letter (and that all names freely occurring before a are from the set B), where guessing a letter from the set D is not allowed. This rules out the situation that a letter from D is used to satisfy a distinguishing box formula in ϕ. Using Lemma 6.3, we obtain:

▶ **Lemma 6.4.** *Let π, π' be permutations, let ϕ be a formula, let w be a bar string, let S be a context, and put $A := \mathsf{FN}(\pi \cdot \phi)$, $A' := \mathsf{FN}(\pi' \cdot \phi)$ and $B := A \cap A'$. Furthermore, let a be either the first freely occuring letter in w that is not in B if that letter exists and is in $A \cup A'$ (in which case we say that a is the* distinguishing letter *w.r.t B and w), and $a = *$ otherwise. Then $a \notin B$ and we have*

$$S, w \models (\pi \cdot \phi) \wedge (\pi' \cdot \phi) \iff S, w \models ((\pi \cdot \phi)^B_A(a)^{A'}_{|w|+1} \wedge (\pi' \cdot \phi)^{\emptyset}_B(*)^{\emptyset}_{|w|+1}) \vee ((\pi' \cdot \phi)^B_{A'}(a)^{A}_{|w|+1} \wedge (\pi \cdot \phi)^{\emptyset}_B(*)^{\emptyset}_{|w|+1}).$$

Relying on name restriction as in Lemma 6.4, we are able to translate formulae to ERRNAs.

▶ **Theorem 6.5.** *For every closed formula ϕ of size m and degree k, there is an ERNNA $A(\phi)$ with degree bounded exponentially in k and polynomially in m, and with number of orbits bounded doubly exponentially in k and singly exponentially in m, that accepts the bar language of ϕ, i.e. $L_\alpha(A(\phi)) = [\![\phi]\!]$.*

We sketch the construction of $A(\phi)$. Let $\mathsf{cl}(\phi)$ denote the closure of ϕ, defined in the standard way. We put

$$\mathsf{Cl} = \mathsf{cl}(\phi) \cup \{\Diamond_\sigma \psi \mid \Box_\sigma \psi \in \mathsf{cl}(\phi)\} \cup \{\epsilon, \bot\} \cup \{\Diamond_b \top \mid b \in \mathsf{BN}(\phi)\},$$

noting $|\mathsf{Cl}| \leq 4m$ (recall that $\bot$ and $\top$ abbreviate $\epsilon \wedge \neg\epsilon$ and $\epsilon \vee \neg\epsilon$, respectively). Furthermore, we put

$$\mathsf{formulae} = \{\psi^B_C(a) \mid \psi \in \mathsf{Cl}, B \subseteq \mathsf{N}(\phi), C \subseteq \mathsf{FN}(\psi), a \in C \cup \{*\}\},$$

noting that the cardinality of $\mathsf{formulae}$ is linear in m and exponential in k; specifically, $|\mathsf{formulae}| \leq |\mathsf{Cl}| \cdot (|\mathsf{N}(\phi)| + 1) \cdot 2^{2|\mathsf{N}(\phi)|} \leq 2^{2k} \cdot 4m(k+1) \leq 2^{2m} \cdot 5m^2$ since we have $|\mathsf{FN}(\psi)| \leq |\mathsf{N}(\phi)|$ for all $\psi \in \mathsf{Cl}$ and since $k \leq m$. The set $\mathsf{formulae}$ contains formulae $\psi \in \mathsf{Cl}$ that are annotated with a single name a (or $*$) and two sets B and C of names. The annotation with a is used to encode a guessed name (with $*$ denoting the situation that no name has been guessed yet) which we call *distinguishing letter*, while the set C encodes the restriction of the support of ψ to C and the set B denotes the names that are allowed to occur freely before the distinguishing letter does. This data will be used to bound the number of copies of subformulae that can occur in nodes of the constructed tableau. We construct an ERRNA $A(\phi) = (Q, \to, s, f)$ with carrier set

$$Q = \big\{\{(\{\pi_1 \cdot \phi_1, \ldots, \pi_n \cdot \phi_n\}, a) \mid \pi_1, \ldots, \pi_n \in G, \{\phi_1, \ldots, \phi_n\} \subseteq \mathsf{formulae}, a \in \mathbb{A} \cup \{*\}\big\},$$

where the π_i act on names as usual and we define $\pi_i(*) = *$. The bound on the number of orbits follows since each combination of a subset Φ of $\mathsf{formulae}$, a name a and an equivalence relation on the set of free names of Φ and a gives rise to at most one orbit. We put $s = (\{\phi^{\emptyset}_{\mathsf{FN}(\phi)}(*)\}, *)$. Given a state $(\Gamma, b) \in Q$, formulae $\pi \cdot (\psi^B_C(a)) \in \Gamma$ stand for instances of ψ in which the distinguishing letter is fixed to be $\pi \cdot a$, $\pi \cdot B$ is the set of free names that are allowed to occur before $\pi \cdot a$ does, and the support of ψ is restricted to $\pi \cdot C$. The shape of a formula $\pi \cdot (\psi^B_C(a))$ is just the shape of ψ; for brevity, we omit the annotations with C, B, a when they are not relevant. To deal with box operators and $\neg\epsilon$, states also contain a separate name component b (which may be $*$) that denotes a guess of the last relevant free name after which the evaluation of formulae from Γ will stop. A state $(\Gamma, b) \in Q$ is *propositional* if Γ contains some formula of the shape $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$, or $\mu X.\, \psi_1$; *quasimodal* if (Γ, b) is not propositional but Γ contains some formula of the form $\neg\epsilon$ or $\Box_\sigma \phi$; and *modal* if (Γ, b) is

neither propositional nor quasimodal, i.e. if all elements of Γ are either of the shape ϵ or $\Diamond_\sigma\phi$. We define the acceptance function $f\colon Q \to \{0, 1, \top\}$ by $f(\emptyset, b) = \top$, $f(\Gamma, b) = 1$ if $\Gamma \neq \emptyset$ and all elements of Γ are of the shape ϵ, and $f(\Gamma, b) = 0$ for all other states.

Transitions work roughly as follows. The component Γ of a state (Γ, b) plays the usual role: It records formulae that the automaton requires the remaining input word to satisfy. To bound the number of free names that have to be tracked, formulae are annotated with guesses for distinguishing letters. The transitions from propositional and modal states follow the standard tableau rules; e.g. given a propositional state $q = (\Gamma \cup \{\psi \wedge \chi\}, b)$, we have $q \xrightarrow{\epsilon} (\Gamma \cup \{\psi, \chi\}, b)$; given a modal state $q = (\{\Diamond_a\psi_1, \dots, \Diamond_a\psi_n\}, b)$, we have $q \xrightarrow{a} (\{\psi_1, \dots, \psi_n\}, b)$; and a modal state containing $\Diamond_a\psi$ and $\Diamond_b\chi$ for $a \neq b$ is a deadlock (the treatment of $\Diamond_{|a}$ is more involved). $\Box$-formulae and $\neg\epsilon$ are dealt with by ϵ-transitions from quasimodal states (Γ, b) to modal states. This process is straightforward if Γ contains some formula $\Diamond_\sigma\psi$. The critical case is the remaining one: A formula $\Box_\sigma\psi$ in Γ can be satisfied by in fact satisfying $\Diamond_\sigma\psi$, by ending the word, or by reading either a plain name c other than σ or a bar name (if $\sigma \in \mathbb{A}$), or by reading any plain name (if $\sigma \in \overline{\mathbb{A}}$). This is where the second component b of states comes in: The letter c must have previously appeared as a bar name; $A(\phi)$ guesses when this happens (in bound transitions from modal states), and records its guess in b, with $*$ representing the situation that no guess has yet been made.

When constructing transitions in $A(\phi)$ as explained above, the set of formulae to which a naively constructed transition leads may contain two instances $\pi \cdot (\psi^B_C(a))$ and $\pi' \cdot (\psi^B_C(a))$ of an annotated formula ψ. In this situation, the rest of the word has to satisfy both formulae, but the set Q can only contain one instance of $\psi^B_C(a)$. Here we use the above restriction technique and repeated application of Lemma 6.4 to ensure that only a single copy needs to be kept.

We also have a converse translation which goes via the equivalence of ERNNAs and extended bar NFAs, and associates a fixpoint variable to each state.

▶ **Theorem 6.6.** *For every ERNNA A there exists a formula ϕ such that $[\![\phi]\!] = L_\alpha(A)$.*

Notably, by Remark 4.5, the results of this section imply

▶ **Corollary 6.7.** *The class of closed bar languages definable by ERNNAs is closed under complement: for each ERNNA A there exists an ERNNA $\bar{A}$ such that $L_\alpha(\bar{A}) = \mathsf{bs}(\emptyset) \setminus L_\alpha(A)$.*

As mentioned previously, RNNAs are essentially equivalent to session automata [38], which are only closed under *resource-bounded* complement [3] for some resource bound k; in our present terminology, this corresponds to complement within the set of closed bar strings, modulo α-equivalence, that can be written with at most k different bound names. It is maybe surprising that full complementation (of session automata or RNNAs) is enabled by simply allowing $\top$-states.

7 Reasoning for Bar-μTL

Using Theorem 6.5, we reduce reasoning problems for Bar-μTL to ERNNAs:

▶ **Definition 7.1** (Reasoning problems)**.** A closed formula ϕ is *satisfiable* if $[\![\phi]\!] \neq \emptyset$, and *valid* if $[\![\phi]\!] = \mathsf{bs}(\emptyset)$. A formula ψ *refines* ϕ if $[\![\psi]\!] \subseteq [\![\phi]\!]$. An RNNA A *satisfies* ϕ (written $A \models \phi$) if $L_\alpha(A) \subseteq [\![\phi]\!]$. The *model checking problem* is to decide whether $A \models \phi$. We sometimes emphasize that these problems refer to *bar languages* or, equivalently, *global freshness*. The corresponding problems for *local freshness* arise by applying the D operator (Section 3) to all bar languages involved; e.g. *A satisfies ϕ under local freshness* if $D(L_\alpha(A)) \subseteq D([\![\phi]\!])$.

The complexity of Bar-μTL reasoning problems, obtained using in particular Theorem 6.5 and Theorem 5.5, is summed up as follows.

▶ **Theorem 7.2.**

1. *Model checking Bar-μTL over RNNAs is in 2ExpSpace, more precisely in para-ExpSpace with the degree of the formula as the parameter, under both bar language semantics (equivalently global freshness) and under local freshness. The same holds for validity under local freshness.*
2. *The satisfiability problem for Bar-μTL is in ExpSpace, more precisely in para-PSpace with the degree of the formula as the parameter, under both bar language semantics / global freshness and under local freshness. The same holds for validty and refinement under bar language semantics / global freshness.*

We leave the complexity (and in fact the decidability) of refinement checking under local freshness semantics as an open problem.

8 Conclusions

We have defined a specification logic Bar-μTL for finite words over infinite alphabets, modelled in the framework of nominal sets, which covers both local and global freshness. Bar-μTL features freeze quantification in the shape of name-binding modalities, and as such relates to Freeze LTL. It combines comparatively low complexity of the main reasoning problems with reasonable expressiveness, in particular unboundedly many registers, full nondeterminism, and closure under complement. Freshness is based on α-equivalence in nominal words; this entails certain expressive limitations on local freshness, which however seem acceptable in relation to the mentioned good computational properties.

An important issue for future work is the behaviour of Bar-μTL over infinite words; also, we will investigate whether our methods extend to languages of data trees (e.g. [13]).

References

1 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011. doi:10.1145/1970398.1970403.

2 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014. doi:10.2168/LMCS-10(3:4)2014.

3 Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A robust class of data languages and an application to learning. *Log. Meth. Comput. Sci.*, 10(4), 2014. doi:10.2168/LMCS-10(4:19)2014.

4 Benedikt Bollig, Karin Quaas, and Arnaud Sangnier. The complexity of flat freeze LTL. *Log. Methods Comput. Sci.*, 15(3), 2019. doi:10.23638/LMCS-15(3:33)2019.

5 Thomas Colcombet. Unambiguity in automata theory. In *Descriptional Complexity of Formal Systems, DCFS 2015*, volume 9118 of *LNCS*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19225-3.

6 Thomas Colcombet and Amaldev Manuel. Generalized data automata and fixpoint logic. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2014*, volume 29 of *LIPIcs*, pages 267–278. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.267.

7 Thomas Colcombet and Amaldev Manuel. μ-calculus on data words. *CoRR*, 2014. arXiv:1404.4827.

8 Thomas Colcombet and Amaldev Manuel. Fragments of fixpoint logic on data words. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2015*, volume 45 of *LIPIcs*, pages 98–111. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.98.

9 Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. *J. ACM*, 68(1):7:1–7:28, 2021. doi:10.1145/3422822.

10 Wojciech Czerwiński and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. *CoRR*, 2021. arXiv:2104.13866.

11 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. doi:10.1145/1507244.1507246.

12 Stéphane Demri and Arnaud Sangnier. When model-checking freeze LTL over counter machines becomes decidable. In *Foundations of Software Science and Computation Structures, FOSSACS 2010*, volume 6014 of *LNCS*, pages 176–190. Springer, 2010. doi:10.1007/978-3-642-12032-9_13.

13 Diego Figueira and Luc Segoufin. Future-looking logics on data words and trees. In *Mathematical Foundations of Computer Science, MFCS 2009*, volume 5734 of *LNCS*, pages 331–343. Springer, 2009. doi:10.1007/978-3-642-03816-7_29.

14 Murdoch J. Gabbay. Foundations of nominal techniques: logic and semantics of variables in abstract syntax. *Bull. Symb. Log.*, 17(2):161–229, 2011. doi:10.2178/bsl/1305810911.

15 Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *Logic in Computer Science, LICS 1999*, pages 214–224. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782617.

16 Murdoch James Gabbay and Vincenzo Ciancia. Freshness and name-restriction in sets of traces with names. In *Foundations of Software Science and Computation Structures, FOSSACS 2011*, volume 6604 of *LNCS*, pages 365–380. Springer, 2011. doi:10.1007/978-3-642-19805-2.

17 Giuseppe De Giacomo and Moshe Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 854–860. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

18 Radu Grigore, Dino Distefano, Rasmus Petersen, and Nikos Tzevelekos. Runtime verification based on register automata. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2013*, volume 7795 of *LNCS*, pages 260–276. Springer, 2013. doi:10.1007/978-3-642-36742-7_19.

19 Jan Groote and Radu Mateescu. Verification of temporal properties of processes in a setting with data. In *Algebraic Methodology and Software Technology, AMAST 1998*, volume 1548 of *LNCS*, pages 74–90. Springer, 1998. doi:10.1007/3-540-49253-4_8.

20 Jan Groote and Tim Willemse. Model-checking processes with data. *Sci. Comput. Prog.*, 56(3):251–273, 2005. doi:10.1016/j.scico.2004.08.002.

21 Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. Model checking systems and specifications with parameterized atomic apropositions. In *Automated Technology for Verification and Analysis, ATVA 2012*, volume 7561 of *LNCS*, pages 122–136. Springer, 2012. doi:10.1007/978-3-642-33386-6_11.

22 Daniel Hausmann, Stefan Milius, and Lutz Schröder. Harnessing LTL with freeze quantification. *CoRR*, 2020. arXiv:2010.10912.

23 Falk Howar, Bengt Jonsson, and Frits Vaandrager. Combining black-box and white-box techniques for learning register automata. In *Computing and Software Science – State of the Art and Perspectives*, volume 10000 of *LNCS*, pages 563–588. Springer, 2019. doi:10.1007/978-3-319-91908-9_26.

24 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

25 Bartek Klin, Sławomir Lasota, and Szymon Toruńczyk. Nondeterministic and co-nondeterministic implies deterministic, for data languages. In *Foundations of Software Science and Computation Structures, FOSSACS 2021*, volume 12650 of *LNCS*, pages 365–384. Springer, 2021. doi:10.1007/978-3-030-71995-1_19.

26 Bartek Klin and Mateusz Łełyk. Scalar and vectorial μ-calculus with atoms. *Log. Methods Comput. Sci.*, 15(4), 2019. doi:10.23638/LMCS-15(4:5)2019.

27 Dexter Kozen. Results on the propositional μ-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

28 Dexter Kozen, Konstantinos Mamouras, Daniela Petrisan, and Alexandra Silva. Nominal Kleene coalgebra. In *Automata, Languages, and Programming, ICALP 2015*, volume 9135 of *LNCS*, pages 286–298. Springer, 2015. doi:10.1007/978-3-662-47666-6.

29 Dexter Kozen, Konstantinos Mamouras, and Alexandra Silva. Completeness and incompleteness in nominal kleene algebra. In *Relational and Algebraic Methods in Computer Science, RAMiCS 2015*, volume 9348 of *LNCS*, pages 51–66. Springer, 2015. doi:10.1007/978-3-319-24704-5.

30 Klaas Kürtz, Ralf Küsters, and Thomas Wilke. Selecting theories and nonce generation for recursive protocols. In *Formal methods in security engineering, FMSE 2007*, pages 61–70. ACM, 2007. doi:10.1145/1314436.1314445.

31 Ranko Lazić. Safely freezing LTL. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2006*, volume 4337 of *LNCS*, pages 381–392. Springer, 2006. doi:10.1007/11944836_35.

32 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. *CoRR*, 2021. arXiv:2104.12695.

33 Amaldev Manuel, Anca Muscholl, and Gabriele Puppis. Walking on data words. *Theory Comput. Sys.*, 59(2):180–208, 2016. doi:10.1007/s00224-014-9603-3.

34 Antoine Mottet and Karin Quaas. The containment problem for unambiguous register automata. In *Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPIcs*, pages 53:1–53:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.53.

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

36 Andrew Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013.

37 Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal automata with name binding. *CoRR*, 2016. arXiv:1603.01455.

38 Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal automata with name binding. In *Foundations of Software Science and Computation Structures, FOSSACS 2017*, volume 10203 of *LNCS*, pages 124–142, 2017. doi:10.1007/978-3-662-54458-7_8.

39 Fu Song and Zhilin Wu. On temporal logics with data variable quantifications: Decidability and complexity. *Inf. Comput.*, 251:104–139, 2016. doi:10.1016/j.ic.2016.08.002.

40 David Turner and Glynn Winskel. Nominal domain theory for concurrency. In *Computer Science Logic, CSL 2009*, pages 546–560, 2009. doi:10.1007/978-3-642-04027-6_39.