

Harnessing LTL With Freeze Quantification

Daniel Hausmann, Stefan Milius, and 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, respectively, serve the verification of processes or documents with data. They relate tightly to formalisms over nominal sets, where names play the role of data. For example, *regular nondeterministic nominal automata (RNNA)* are equivalent to a subclass of the standard register automata model, characterized by a lossiness condition referred to as name dropping. This subclass generally enjoys better computational properties than the full class of register automata, for which, e.g., inclusion checking is undecidable. Similarly, satisfiability in full Freeze LTL is undecidable, and decidable but not primitive recursive if the number of registers is limited to at most one. In the present paper, we introduce a name-dropping variant $\text{Bar-}\mu\text{TL}$ of Freeze LTL for finite words over an infinite alphabet. We show by reduction to *extended regular nondeterministic nominal automata (ERNNA)* that even with unboundedly many registers, model checking for $\text{Bar-}\mu\text{TL}$ over RNNA is elementary, in fact in EXPSPACE , more precisely in parametrized PSPACE , effectively with the number of registers as the parameter.

1 Introduction

While formal languages and automata models classically work with finite alphabets, there has also been longstanding interest in infinite alphabets, which may be seen as modelling data. For example, nonces in cryptographic protocols [22], data values in XML documents [25], object identities [15], or parameters of method calls [19] can all be usefully understood as letters in infinite alphabets. Briefly, the contributions of the present paper are a linear-time temporal logic and an associated automaton model for finite words over infinite alphabets, distinguished by comparatively low complexity of their main decision problems (such as model checking, satisfiability, validity, language inclusion).

There is by now a wide variety of automata models for languages over infinite alphabets; for our present purposes, we are most interested in models descending from Kaminski and Francez’ original register automaton model [20], in which letters encountered in words over infinite alphabets can be stored in registers for equality or inequality comparison with letters encountered later (we briefly review further models further below). Register automata are equivalent to automata models over nominal sets, specifically to nondeterministic orbit-finite automata (NOFA) [2]. A central challenge in dealing with such models is that

key decision problems are either undecidable or of prohibitively high complexity. For example, language inclusion of nondeterministic register automata, or equivalently NOFAs, is undecidable [20] (it becomes decidable only if the number of registers is restricted to at most two); and language inclusion (equivalently nonemptiness) of alternating one-register automata is decidable but not primitive recursive [7]. The situation is better for deterministic or unambiguous [24] models, which however have markedly lower expressiveness; e.g. the language ‘some letter appears twice’ can be accepted by a non-deterministic register automaton but not by any deterministic or unambiguous one [2]. Specification logics over infinite alphabets often include a *freeze quantifier* which binds letters from infinite alphabets encountered in words to logical registers, e.g. in *Freeze LTL*. The algorithmics of such logics is similarly challenging, e.g. satisfiability checking of Freeze LTL over finite words is undecidable, and decidable but not primitive recursive if the number of registers is restricted to at most one [7].

These computational issues can be ameliorated by a mild recalibration of the notion of freshness: While standard register automata allow requiring a newly seen data value to be fresh w.r.t. the data values currently stored in the registers, *regular non-deterministic nominal automata (RNNA)* [28] enforce *local freshness* of data values via a mechanism based on the principle of α -equivalence as known from λ -calculus and other name-binding formalisms, exploiting the fact that renaming of bound names needs to avoid clashes with free names. The main effect of this mechanism, applied to words with name binding referred to as *bar strings*, is that the automaton can insist on new data values being distinct from a stored value only if the stored value is expected to be seen again later in the word. RNNA can be identified with a subclass of NOFAs (equivalently register automata) characterized by a lossiness condition referred to as *name dropping*. Name dropping implies a decrease in expressiveness; e.g. while the language $\{aba \mid a \neq b\}$ can be accepted by an RNNA (which can insist that b is distinct from a because a is expected to be seen again), the language $\{ab \mid a \neq b\}$ cannot. The restrictions imposed by name dropping seem 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 if a is never going to be used again; note also that, e.g., the mentioned language ‘some letter occurs twice’ can be accepted by an RNNA), and buy a substantial decrease in complexity. Indeed, language inclusion of RNNAs is in EXPSpace (more precisely parametrized PSPACE [30], the parameter being the number of registers), with no need to restrict nondeterminism or the number of registers.

In the present paper, we apply these principles in the logical setting, specifically to Freeze LTL. We develop a variant of Freeze LTL, or more precisely its extension with a (guarded) fixpoint construct, over finite words whose semantics is based on α -equivalence, and hence exhibits name dropping. This logic **Bar- μ TL** retains a reasonable level of expressiveness slightly above that of RNNAs. Nevertheless, even without bounding the number of registers, **Bar- μ TL** model checking over the above-mentioned RNNA is decidable in elementary complexity, specifically, again, in parametrized PSPACE (compare this to the mentioned high

complexity of Freeze LTL with one register). We establish this by translating $\text{Bar-}\mu\text{TL}$ into *extended regular alternating nominal automata (ERNNA)*, which slightly extend RNNAs (by what amounts to a very restricted form of alternation where all universal nodes are deadlocks), and showing that language inclusion of RNNAs in ERNNA is in parametrized PSPACE. The satisfiability problem of $\text{Bar-}\mu\text{TL}$ reduces to non-emptiness checking for ERNNA, which we show to be in PSPACE, while the validity problem of $\text{Bar-}\mu\text{TL}$ is in EXPSpace by reduction to the model checking problem. Like RNNAs, $\text{Bar-}\mu\text{TL}$ in fact leads a second life as a formalism for global freshness (indeed, the difference between global and local freshness is determined simply by whether one does or does not insist on clean α -renaming of bar strings). The above results on model checking and satisfiability checking (although not the one for validity checking) hold in essentially the same way also for this global freshness interpretation, so that $\text{Bar-}\mu\text{TL}$ may also be seen as a logical language for session automata [3].

Organization We recall basic notions and concepts on nominal sets and data languages in Sections 2 and 3, and introduce the logic $\text{Bar-}\mu\text{TL}$ in Section 4. In Section 5, we define ERNNA, show how $\text{Bar-}\mu\text{TL}$ formulae can be transformed into ERNNA and present a name-dropping construction that transforms ERNNA to bar language equivalent ERNNA whose literal language is closed under α -equivalence. In Section 6 we obtain our main result, a model checking algorithm for $\text{Bar-}\mu\text{TL}$ that realizes inclusion checking of RNNAs in ERNNA.

Related work *Automata:* As indicated above, over infinite alphabets, the expressive power of automata models generally increases with the power of control (deterministic/nondeterministic/alternating). Deterministic models often allow deciding language inclusion in reasonable complexity; this remains true for unambiguous register automata [24]. Nondeterministic automaton models in the vicinity of register automata tend to have decidable nonemptiness problems but undecidable inclusion problems; decidability of inclusion is typically regained only by drastic restrictions on the number of registers as indicated above. The decidability of the nonemptiness problem of alternating one-register automata [7] (in the mentioned high complexity) is preserved under extensions with nondeterministic update and a form of quantification over data values [9]. Automata models for infinite words outside the register paradigm include data walking automata [23], whose inclusion problem is decidable even under nondeterminism but at least as hard as Petri net reachability, and the highly expressive data automata [1], whose nonemptiness problem is decidable but, again, at least as hard as Petri net reachability. The principles of name binding automata and name dropping that we employ in the present work go back to work on regular nondeterministic nominal automata (RNNAs) [28]. RNNAs have an alternative global freshness semantics, and under this semantics are essentially equivalent to session automata [3]. ERNNAs with global semantics are slightly more expressive than session automata, as they can express the universal language. (E)RNNAs natively accept strings with name binding, so-called bar strings, which are related both to dynamic sequences [14] and to linear expressions in nominal Kleene algebra [13].

Logics: The μ -calculus with atoms [21] has an infinitary but orbit-finite syntax and 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 [16, 17], for which model checking is only known to be semidecidable. The μ -calculus over data words [5, 6] 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 reachability in vector addition systems. *Variable LTL* [18] 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 [29]. Some fragments have decidable satisfiability or model checking problems (typically not both) [18, 29], 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- μ TL needs to be left open at present. Flat Freeze LTL [8] interdicts usage of the freeze quantifier in safety positions (e.g. the freeze quantifier can occur under F but not under G); its model checking problem over one-counter automata is NEXPTIME-complete [4].

2 Preliminaries: Nominal Sets

Nominal sets form 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 [26] 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* (ab) for $a \neq b \in \mathbb{A}$ (recall that (ab) just swaps a and b). A *nominal set* is a set X equipped with a (left) action $(-)\cdot(-)$ of G such that every element $x \in X$ has a finite *support* $S \subseteq \mathbb{A}$, i.e. $\pi \cdot x = x$ for every $\pi \in G$ such that $\pi(a) = a$ for all $a \in S$. Every element x of a nominal set has a least finite support, denoted $\text{supp}(x)$; one may roughly think of $\text{supp}(x)$ as consisting of the names needed to describe x . A name $a \in \mathbb{A}$ is *fresh* for x if $a \notin \text{supp}(x)$. Putting $\pi \cdot a = \pi(a)$ makes \mathbb{A} into a nominal set. Moreover, G acts on subsets $A \subseteq X$ of a nominal set X by $\pi \cdot A = \{\pi \cdot x \mid x \in A\}$, and on functions $f: X \rightarrow Y$ between nominal sets by $(\pi \star f)(x) = \pi \cdot (f(\pi^{-1} \cdot x))$ (we denote the action on functions by \star to avoid misreadings). A subset of a nominal set or a function between nominal sets is *finitely supported* if it has finite support w.r.t. the respective actions of G just described, and *equivariant* if it has empty support. That is, $A \subseteq X$ is equivariant (has support S) if $\pi \cdot A \subseteq A$ for all $\pi \in G$ (with $\pi(s) = s$ for all $s \in S$), and $f: X \rightarrow Y$ is equivariant (has support S) if $f(\pi \cdot x) = \pi \cdot f(x)$ for all $x \in X$ and $\pi \in G$ (with $\pi(s) = s$ for all $s \in S$). We denote the set of finitely supported subsets of a nominal set X by $\mathcal{P}_{\text{fs}}(X)$. For equivariant f , we have $\text{supp}(f(x)) \subseteq \text{supp}(x)$. The function supp itself is equivariant, i.e. $\text{supp}(\pi \cdot x) = \pi \cdot (\text{supp}(x))$ for $\pi \in G$. Hence

$|\text{supp}(x_1)| = |\text{supp}(x_2)|$ whenever x_1, x_2 are in the same orbit of a nominal set; recall that the *orbit* of $x \in X$ is the set $\{\pi \cdot x \mid \pi \in G\}$.

The Cartesian product $X \times Y$ of nominal sets X and Y is a nominal set under the componentwise group action; we have $\text{supp}(x, y) = \text{supp}(x) \cup \text{supp}(y)$. Given a nominal set X equipped with an equivariant relation, that is an equivalence relation \sim which is an equivariant subset $\sim \subseteq X \times X$, the quotient X/\sim is a nominal set under the expected group action defined by $\pi \cdot [x]_\sim = [\pi \cdot x]_\sim$.

A subset A of a nominal set X is *uniformly finitely supported* if $\bigcup_{x \in A} \text{supp}(x)$ is finite [31], in which case A is also finitely supported [12, Theorem 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 a finite subset $B \subseteq \mathbb{A}$, the set $B^* \subseteq \mathbb{A}^*$ is uniformly finitely supported. The complement of a uniformly finitely supported set is typically not uniformly finitely supported, so we consider a form of relative complement.

Definition 2.1 (Support-relative complement). Let $S \subseteq \mathbb{A}$. The *S-relative complement* (or, omitting mention of S , *support-relative complement*) of a uniformly finitely supported subset A of a nominal set X is the uniformly finitely supported set $\{x \in X \setminus A \mid \text{supp}(x) \subseteq S\}$.

A key role in the technical development is played by *abstraction sets*, which provide a semantics for binding mechanisms [11].

Definition 2.2 (Abstraction set). Given a nominal set X , an equivalence relation \sim on $\mathbb{A} \times X$ is defined by $(a, x) \sim (b, y)$ iff $(a\ c) \cdot x = (b\ c) \cdot y$ for some (equivalently, all) fresh 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 $\text{supp}(\langle a \rangle x) = \text{supp}(x) \setminus \{a\}$ (while $\text{supp}(a, x) = \{a\} \cup \text{supp}(x)$), as expected in binding constructs.

3 Data Languages and Bar Languages

As indicated above, 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} that additionally may contain occurrences of the bar symbol ‘|’, which indicates that the next letter is bound until the end of the word; such words are called *bar strings* [28]. 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. We will see that bar strings modulo α -equivalence relate closely to formalisms for *global* freshness (which require that a name 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. Returning to the examples in the introduction, we find that the bar string $\mathbf{la} \mathbf{b} a$ then represents the set of words

$$\{cdc \mid c, d \in \mathbb{A}, c \neq d\} \subseteq \mathbb{A}^*$$

while $\mathbf{la} \mathbf{b}$ just represents the set \mathbb{A}^2 of all two-letter words – in $\mathbf{la} \mathbf{b} a$, a and b cannot be renamed into the same letter, since a occurs freely in the scope of b , while $\mathbf{la} \mathbf{b}$ is α -equivalent to $\mathbf{la} \mathbf{a}$. The impossibility of expressing the language $\{cd \mid c \neq d\}$ under local freshness is thus hardwired into our model of in data words. We emphasize again that this restriction seems reasonable in practice, since one may expect that freshness of new letters is only relevant 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 $\overline{\mathbb{A}} = \mathbb{A} \cup \{\mathbf{la} \mid a \in \mathbb{A}\}$; we refer to elements $\mathbf{la} \in \overline{\mathbb{A}}$ as *bar names*, and occasionally call the other elements of $\overline{\mathbb{A}}$ *plain names* for distinction. A *bar string* is a word $w = a_1 a_2 \dots a_n \in \overline{\mathbb{A}}^*$, with *length* denoted $|w| = n$; we denote the *empty bar string* (with $n = 0$) by ϵ . We turn $\overline{\mathbb{A}}$ into a nominal set by $\pi \cdot a = \pi(a)$ and $\pi \cdot \mathbf{la} = \mathbf{l}\pi(a)$; then, $\overline{\mathbb{A}}^*$ is a nominal set under the pointwise action of G . We define α -equivalence on bar strings to be the equivalence generated by $w \mathbf{la} v \equiv_\alpha w \mathbf{b} u$ if $\langle a \rangle v = \langle b \rangle u$ in $[\mathbb{A}] \overline{\mathbb{A}}^*$ (cf. Definition 2.2). Thus, \mathbf{la} 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 \mathbf{la} . We write $\text{FN}(w) = \{a \in \mathbb{A} \mid a \text{ is free in } w\}$ for the set of free names of w . A bar string w is *clean* if all occurrences of bar names \mathbf{la} in w have a distinct from all free names of w , and moreover have pairwise distinct a . We write $[w]_\alpha$ for the α -equivalence class of $w \in \overline{\mathbb{A}}^*$. A set of bar strings is α -closed if it is closed under α -equivalence. For a set $S \subseteq \mathbb{A}$ of names, we let $\text{free}(S) = \{w \in \overline{\mathbb{A}}^* \mid \text{FN}(w) \subseteq S\}$ denote the set of bar strings that only use free names from S .

Remark 3.2. Note that α -equivalence is not a congruence w.r.t. concatenation from the right; e.g. $\mathbf{la} \equiv_\alpha \mathbf{lb}$ but $\mathbf{la} a \not\equiv_\alpha \mathbf{lb} a$.

We will work with three different types of languages:

- *Data languages* are subsets of \mathbb{A}^* .
- *Literal languages* are subsets of $\overline{\mathbb{A}}^*$, i.e. sets of bar strings.
- *Bar languages* are subsets of $\overline{\mathbb{A}}^* / \equiv_\alpha$, i.e. sets of α -equivalence classes of bar strings; these are the natural languages underlying our models, and relate tightly to data languages as discussed next. We may represent a bar language as an α -closed literal language. Bar languages accepted by regular nominal nondeterministic automata (RNNA) [28] are always uniformly finitely supported; this appears to be a key aspect contributing to the good computational properties of RNNA, and we will explicitly design $\text{Bar-}\mu\text{TL}$ to work only with uniformly finitely supported bar languages. A bar language is *closed* if

it has empty support, equivalently consists of closed bar strings. We will occasionally describe example bar languages as regular expressions over $\overline{\mathbb{A}}$ (i.e. as *regular bar expressions* [28]), meaning the set of all bar strings that are α -equivalent to some instance of the expression.

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

$$N(L) = \{\mathbf{ub}(w) \mid [w]_\alpha \in L, w \text{ clean}\}$$

defines a global freshness interpretation of \mathbf{l} (in fact, the operator N is injective on closed bar languages [28]), while

$$D(L) = \{\mathbf{ub}(w) \mid [w]_\alpha \in L\}$$

defines a local freshness interpretation as exemplified above; e.g. $D(\mathbf{la}|\mathbf{ba}) = \{aba \mid a, b \in \mathbb{A}, a \neq b\}$ (where we use regular expressions over $\overline{\mathbb{A}}$ to describe bar languages as indicated above), while $D(\mathbf{la}|\mathbf{b}) = \{ab \mid a, b \in \mathbb{A}\}$; on the other hand, $N(\mathbf{la}|\mathbf{b}) = \{ab \mid a, b \in \mathbb{A}, a \neq b\}$ (the previous comments on the inexpressibility of this language thus applies *only* to the local freshness interpretation).

Remark 3.3. We will equip $\mathbf{Bar}\text{-}\mu\mathbf{TL}$ with a primary bar language semantics, from which we will derive a global freshness semantics and a local freshness semantics using N and D , respectively. Injectivity of N on closed bar languages 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 the fact that \mathbf{ub} is injective on clean closed bar strings [27, Lemma A.4] that N preserves intersection and \emptyset -relative complement (Definition 2.1) of closed bar languages, the latter in the sense that $N(\mathbf{free}(\emptyset) \setminus L) = \overline{\mathbb{A}}^* \setminus N(L)$ for closed bar languages L . On the other hand, the local freshness interpretation D does not preserve intersection or support-relative complement: E.g. the bar languages \mathbf{laa} and $\mathbf{la}|\mathbf{a}$ have empty intersection, but $D(\mathbf{laa}) \cap D(\mathbf{la}|\mathbf{a}) = \{aa \mid a \in \mathbb{A}\}$. Also, the \emptyset -relative complement of the bar language $\mathbf{la}|\mathbf{b}$ consists of \mathbf{laa} and all closed words of length other than 2, while $D(\mathbf{la}|\mathbf{b})$ contains all words of length 2, so any form of (relative) complement of $D(\mathbf{la}|\mathbf{b})$ will not contain any words of length 2. Conjunction (and a definable support-relative negation) in $\mathbf{Bar}\text{-}\mu\mathbf{TL}$ will refer to bar languages; generally, the understanding of a $\mathbf{Bar}\text{-}\mu\mathbf{TL}$ formula should be based on understanding its global freshness semantics, from which the local freshness semantics is understood by giving up freshness of names that do not appear again later in the word.

Remark 3.4. Data words are often defined as being words over $(\Sigma \times D)$ where Σ is a finite alphabet and D an infinite alphabet (\mathbb{A} , in the present setting). The additional finite alphabet Σ is easily coded into bar strings via free names. For example, if \mathbb{A} is seen as a set of messages, and $\Sigma = \{\mathbf{send}, \mathbf{receive}\}$, then the two-letter data word $(\mathbf{send}, a)(\mathbf{receive}, a)$ over $\Sigma \times \mathbb{A}$ can be encoded as the four-letter word $\tau(\mathbf{send})a\tau(\mathbf{receive})a$ where τ injects Σ into $\mathbb{A} \setminus \{a\}$.

4 Syntax and Semantics of Bar- μ TL

We proceed to introduce a variant Bar- μ TL of linear temporal logic whose formulae define bar languages. This logic relates to a fragment of Freeze LTL, whose operator for reading a letter from the input word corresponds to name binding modalities in Bar- μ TL.

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

$$\phi, \psi := \epsilon \mid \neg\epsilon \mid \top \mid \perp \mid \phi \wedge \psi \mid \phi \vee \psi \mid \Diamond_\beta \phi \mid \Box_\beta \phi \mid X \mid \mu X. \phi,$$

where $\beta \in \overline{\mathbb{A}}$ and $X \in V$. We refer to \Diamond_β and \Box_β as β -*operators*. 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 (both \top and $\neg\epsilon$ additionally impose that the free names of the remaining word come from a given context; this will be made precise in the formal definition of the semantics). 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 $\mathbf{l}a$ -operators is similar but involves α -renaming as detailed below; as indicated in Section 3, this means that $\mathbf{l}a$ -operators effectively read fresh letters. They thus replace the freeze quantifier; one important difference with the latter is that $\Diamond_{\mathbf{l}a}$ consumes a letter, while the freeze quantifier standardly does not. A name a is *free* in a formula ϕ if ϕ contains an a -operator at a position that is not in the scope of any $\mathbf{l}a$ -operator; that is, $\mathbf{l}a$ -operators bind the name a . We write $\text{FN}(\phi)$ for the set of free names in ϕ .

We define a notion of free (fixpoint) variable, determined as usual by letting μ bind variables. We write $\text{FV}(\phi)$ for the set of free variables in ϕ ; a formula ϕ is *closed* if $\text{FV}(\phi) = \emptyset$. We assume that formulae ϕ are *clean*, i.e. bound fixpoint variables in ϕ are mutually distinct and distinct from all free variables, and then denote by $\theta(X)$ the subformula $\mu X. \psi$ of ϕ that binds a bound variable X in ϕ . Moreover, we require that all fixpoint variables are *guarded*, that is, occur only within the scope of at least one modal operator within their binding fixpoint expressions.

For purposes of making Bar a nominal set, we regard every fixpoint variable X as being annotated with the set $\text{FN}(\theta(X))$ of free names in the binding formula $\theta(X)$. We then let G act by replacing names (including bound ones!) in the obvious way; i.e. $\pi \cdot \phi$ is the formula obtained from ϕ by replacing every plain name a with $\pi(a)$, also in annotations of fixpoint variables, and every bar name $\mathbf{l}a$ with $\mathbf{l}\pi(a)$. We define α -*equivalence* \equiv_α on formulae to be the congruence relation generated by $\Diamond_{\mathbf{l}a} \phi \equiv_\alpha \Diamond_{\mathbf{l}b} \psi$ and $\Box_{\mathbf{l}a} \phi \equiv_\alpha \Box_{\mathbf{l}b} \psi$ if $\langle a \rangle \phi = \langle b \rangle \psi$ (cf. Definition 2.2). As usual, this is equivalent to the expected recursive definition, e.g. $\Diamond_{\mathbf{l}a} \phi \equiv_\alpha \Diamond_{\mathbf{l}b} \psi$ iff there is ψ' such that $\langle a \rangle \phi = \langle b \rangle \psi'$ and $\psi' \equiv_\alpha \psi$.

Remark 4.1. The point of annotating fixpoint variables with the free names of their binding formulae is to block unsound α -renamings. E.g. the formulae $\phi_1 = \mu X. \Diamond_a \epsilon \vee \Diamond_{\mathbf{I}a} X$ and $\phi_2 = \mu X. \Diamond_a \epsilon \vee \Diamond_{\mathbf{I}b} X$ are *not* α -equivalent because X is, for purposes of the action of G , in fact $(X, \{a\})$ so that α -renaming of $\Diamond_{\mathbf{I}b} X$ into $\Diamond_{\mathbf{I}a} X$ is blocked. We have yet to introduce the formal semantics but it should be clear that ϕ_1 is meant to define the bar language of bar strings (modulo α -equivalence) consisting of any number of occurrences of $\mathbf{I}a$ followed by a final a , while ϕ_2 defines the set of bar strings consisting of any number of occurrences of $\mathbf{I}b$ followed by a final a , so ϕ_1 and ϕ_2 will define different bar languages. The key closure condition of α -equivalence enabled by the above definition is compatibility with fixpoint unfolding, recorded in the next lemma.

Lemma 4.2. *Let $\mu X. \phi \equiv_\alpha \mu X. \phi'$. Then $\phi[\mu X. \phi/X] \equiv_\alpha \phi'[\mu X. \phi'/X]$*

Semantics We interpret each bar formula ϕ as denoting a uniformly finitely supported bar language, depending on a *context* $\text{FN}(\phi) \subseteq S \subseteq \mathbb{A}$, which specifies names that are allowed to occur freely. We define the satisfaction relation \models that relates bar strings w in context S with formulae ϕ recursively by

$S, w \models \perp$	\Leftrightarrow	never
$S, w \models \top$	\Leftrightarrow	$w \in \text{free}(S)$
$S, w \models \neg \epsilon$	\Leftrightarrow	$w \neq \epsilon$ and $w \in \text{free}(S)$
$S, w \models \epsilon$	\Leftrightarrow	$w = \epsilon$
$S, w \models \phi \wedge \psi$	\Leftrightarrow	$S, w \models \phi$ and $S, w \models \psi$
$S, w \models \phi \vee \psi$	\Leftrightarrow	$S, w \models \phi$ or $S, w \models \psi$
$S, w \models \mu X. \phi$	\Leftrightarrow	$S, w \models \phi[\mu X. \phi/X]$
$S, w \models \Diamond_a \phi$	\Leftrightarrow	$\exists v. w = av$ and $S, v \models \phi$
$S, w \models \Box_a \phi$	\Leftrightarrow	$\forall v. \text{if } w = av \text{ then } S, v \models \phi$
$S, w \models \Diamond_{\mathbf{I}a} \phi$	\Leftrightarrow	$\exists \psi \in \text{Bar}, v \in \overline{\mathbb{A}}^*, b \in \mathbb{A}.$ $w \equiv_\alpha \mathbf{I}bv$ and $\langle a \rangle \phi = \langle b \rangle \psi$ and $S \cup \{b\}, v \models \psi$
$S, w \models \Box_{\mathbf{I}a} \phi$	\Leftrightarrow	$\forall \psi \in \text{Bar}, v \in \overline{\mathbb{A}}^*, b \in \mathbb{A}.$ if $w \equiv_\alpha \mathbf{I}bv$ and $\langle a \rangle \phi = \langle b \rangle \psi$ then $S \cup \{b\}, v \models \psi$

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; in other words, μ takes unique guarded fixpoints. Notice that $\Diamond_{\mathbf{I}a}$ and $\Box_{\mathbf{I}a}$ allow α -renaming of both the input word and the formula. We denote the *language* of ϕ in context S by

$$\llbracket S \vdash \phi \rrbracket = \{w \in \overline{\mathbb{A}}^* \mid S, w \models \phi\}$$

and note that $\llbracket S \vdash \phi \rrbracket$ is uniformly finitely supported by construction; specifically, $\llbracket S \vdash \phi \rrbracket \subseteq \text{free}(S)$.

The *global* and *local freshness semantics* of a formula ϕ in context S are given by

$$N(\llbracket S \vdash \phi \rrbracket) \quad \text{and} \quad D(\llbracket S \vdash \phi \rrbracket),$$

respectively, where N and D are the operations converting bar languages into data languages described in Section 3. For distinction, we sometimes refer to $\llbracket S \vdash \phi \rrbracket$ itself as the *bar language semantics* of ϕ .

Example 4.3. To illustrate the semantics, let $\psi = \Diamond_{\mathbf{I}a}(\Diamond_b \top \vee (\epsilon \wedge \Diamond_a \epsilon))$. In context S , ψ specifies bar strings in which the first letter is some bound letter, in which the second letter is b and in which all free names are contained in S . The disjunct $\epsilon \wedge \Diamond_c \epsilon$ is contradictory and not satisfied by any barstring. E.g. we have $\{b\}, \mathbf{I}cbclcb \models \psi$ since $\{b, c\}, bclcb \models \Diamond_b \top$: we have $\{b, c\}, clcb \models \top$, since all free names in $clcb$ are contained in $\{b, c\}$. The semantics of further formulae is discussed in Example 4.7.

Remark 4.4. The fixpoint operator of $\text{Bar-}\mu\text{TL}$ takes the role that is played by the temporal operators in Freeze LTL. In bar language semantics, the overall mode of expression in $\text{Bar-}\mu\text{TL}$ is slightly different from that of Freeze LTL, due to the fact that the word needs to be traversed using modalities tied to specific letters in $\text{Bar-}\mu\text{TL}$ – that is, one part of the expressive restrictions guaranteeing the good computational properties of $\text{Bar-}\mu\text{TL}$ is the absence of a *next* operator \bigcirc that would accept unrestricted letters in $\overline{\mathbf{A}}$. Correspondingly, in bar language semantics, $\text{Bar-}\mu\text{TL}$ also has no direct equivalent of, say, the F operator (‘eventually’), which using \bigcirc would be expressed as $F\phi = \mu X. \phi \vee \bigcirc X$. Technically, this restriction relates to the policy of keeping the bar language denoted by a $\text{Bar-}\mu\text{TL}$ formula uniformly finitely supported; e.g. $\bigcirc \epsilon$ would denote the language of all single-letter bar strings, which fails to be uniformly finitely supported. Indeed, the evaluation of formulae in a context similarly serves the purpose of keeping the bar languages defined by formulae uniformly supported, as the semantics of both \top and $\neg \epsilon$ would fail to be uniformly finitely supported without restriction to the context.

In local freshness semantics, the intended way of specifying an unrestricted next letter is via name binding modalities; e.g. the formula $\Diamond_{\mathbf{I}a} \epsilon$ specifies the bar language consisting of all bar strings consisting of a single bar name, which under local (and also global) freshness semantics becomes the data language consisting of all single-letter words. Generally, $\Diamond_{\mathbf{I}a}$, for fresh a , acts like a *next* operator in local freshness semantics. Indeed, in local freshness semantics, the formula $\mu X. \phi \vee \Diamond_{\mathbf{I}a} X$ specifies that ϕ is satisfied at some point in the word, thus implementing F ; in particular, $\mu X. \epsilon \vee \Diamond_{\mathbf{I}a} X$ defines the universal data language, so \top is not actually needed in local freshness semantics. This paradigm is further illustrated in Example 4.7.

Remark 4.5. By taking negation normal forms, we can define support-relative negation \neg , i.e. $\llbracket S \vdash \neg \phi \rrbracket = \text{free}(S) \setminus \llbracket S \vdash \phi \rrbracket$.

We note next that languages of formulae are closed under α -equivalence of bar strings, i.e. actually represent bar languages, and that α -equivalent renaming of formulae indeed does not affect the semantics (cf. Remark 4.1):

Lemma 4.6. *For $\phi, \psi \in \text{Bar}$, $a \in \mathbb{A}$, $S \subseteq \mathbb{A}$, 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 $\phi \equiv_\alpha \psi$, then $\llbracket S \vdash \phi \rrbracket = \llbracket S \vdash \psi \rrbracket$.*

The proof is by induction along the recursive definition of the semantics; the case for fixpoints in Claim (2) is by Lemma 4.2.

Example 4.7. We consider some Bar- μTL formulae and their respective semantics under local and global freshness (corresponding to application of the D and N operators from Section 3).

- The formula \top , when evaluated in the empty context, denotes the set of all closed bar strings. Under both global and local freshness semantics, this becomes the set of all data words.

- The formula $\Diamond_{\mathbf{la}} \Box_a \epsilon$, when evaluated in the empty context, describes the language of all closed bar strings that start with a bar name \mathbf{la} and which stop after the second letter if there is a second letter and that letter is the plain name a . 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.

- The formula

$$\mu X. ((\Diamond_{\mathbf{la}} X) \vee (\Diamond_{\mathbf{la}} \mu Y. ((\Diamond_{\mathbf{lb}} Y) \vee \Diamond_a \top)))$$

describes, when evaluated in the empty context, the language 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. Notice that during the evaluation of the formula, the context can become unboundedly large, as it grows every time a bar name is read.

- The similar formula

$$\mu X. ((\Diamond_{\mathbf{la}} X) \vee (\Diamond_{\mathbf{la}} \mu Y. ((\Diamond_{\mathbf{lb}} Y) \vee \Diamond_a \epsilon)))$$

describes, when evaluated in the empty context, the language of all closed bar strings where all names except the last one are bar 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.

- To illustrate both the mechanism of local freshness via α -equivalence and, once again, the use of \top , we consider the formula

$$\Diamond_{\mathbf{la}} \Diamond_{\mathbf{lb}} \mu X. ((\Diamond_{\mathbf{lb}} X) \vee \Diamond_a \Diamond_b \top).$$

It describes, again in the empty context, the language of all closed bar strings that start with a bar name \mathbf{la} , at some later point contain a substring \mathbf{lbab} ,

and have only bar names distinct from $|a$ 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 .

We have discussed only the empty global context in these examples; in general, the global context should be thought of as defining a finite alphabet of actions as exemplified in Remark 3.4.

We conclude this section by introducing syntactic notions needed later in the automata translation and the complexity estimates and showing that the operators $\neg\epsilon$, \Box_a and $\Box_{|a}$ can be eliminated from formulae:

Definition 4.8 (Closure, degree). We define the *closure* $\text{cl}(\phi)$ of a formula $\phi \in \text{Bar}$ recursively by

$$\begin{aligned} \text{cl}(\top) &= \{\top\} & \text{cl}(\perp) &= \{\perp\} \\ \text{cl}(\epsilon) &= \{\epsilon\} & \text{cl}(\neg\epsilon) &= \{\neg\epsilon\} \\ \text{cl}(\psi_1 \vee \psi_2) &= \{\theta^*(\psi_1 \vee \psi_2)\} \cup \text{cl}(\psi_1) \cup \text{cl}(\psi_2) & \text{cl}(\mu X. \psi) &= \{\theta^*(\mu X. \psi)\} \cup \text{cl}(\psi) \\ \text{cl}(\psi_1 \wedge \psi_2) &= \{\theta^*(\psi_1 \wedge \psi_2)\} \cup \text{cl}(\psi_1) \cup \text{cl}(\psi_2) & \text{cl}(X) &= \emptyset \\ \text{cl}(\Diamond_a \psi) &= \{\theta^*(\Diamond_a \psi)\} \cup \text{cl}(\psi) & \text{cl}(\Box_a \psi) &= \{\theta^*(\Box_a \psi)\} \cup \text{cl}(\psi) \\ \text{cl}(\Diamond_{|a} \psi) &= \{\theta^*(\Diamond_{|a} \psi)\} \cup \text{cl}(\psi) & \text{cl}(\Box_{|a} \psi) &= \{\theta^*(\Box_{|a} \psi)\} \cup \text{cl}(\psi) \end{aligned}$$

where $\theta^*(\phi)$ denotes the formula that is obtained from ϕ by repeatedly replacing free fixpoint variables X with $\theta(X)$. Furthermore, we define $\text{deg}(\phi)$ to be least number k such that for all subformulae ψ of ϕ , we have $|\text{FN}(\phi) \cup \text{scope}(\psi)| \leq k$, where $\text{scope}(\psi) = \{a \in \mathbb{A} \mid \psi \text{ occurs in the scope of } \Diamond_{|a} \text{ or } \Box_{|a} \text{ within } \phi\}$.

Fact 4.9. We have $|\text{cl}(\phi)| \leq |\phi|$.

Lemma 4.10. For all $S \subseteq \mathbb{A}$, all $a \in S$ and $c \in \mathbb{A}$, all $w \in \overline{\mathbb{A}}^*$ and all $\psi \in \text{Bar}$, we have the following:

$$\begin{aligned} S, w \models \neg\epsilon &\Leftrightarrow S, w \models \bigvee_{b \in S} \Diamond_b \top \vee \Diamond_{|c} \top \\ S, w \models \Box_a \psi &\Leftrightarrow S, w \models \bigvee_{b \in S \setminus \{a\}} \Diamond_b \top \vee \Diamond_a \psi \vee \Diamond_{|c} \top \vee \epsilon \\ S, w \models \Box_{|c} \psi &\Leftrightarrow S, w \models \bigvee_{b \in S} \Diamond_b \top \vee \Diamond_{|c} \psi \vee \epsilon \end{aligned}$$

It follows that the operators $\neg\epsilon$, \Box_a and $\Box_{|a}$ can be eliminated from formulae while preserving semantics and increasing formula size only linearly; hence we assume from now on that *all formulae are $\neg\epsilon$ -free, \Box_a -free and $\Box_{|a}$ -free*.

5 Extended Regular Nondeterministic Nominal Automata

We proceed to introduce the automaton model we use in model checking, extended regular nondeterministic nominal automata (ERNNA), a slightly generalized version of RNNA [28]. We extend the *name-dropping* construction from RNNA to ERNNA; the point of this construction is that unlike for (E)RNNA in general, the literal languages (Section 3) accepted by the resulting name-dropping automata are closed under α -equivalence. Finally, we show how Bar- μ TL formulae can be transformed into ERNNA.

Definition 5.1 (Extended regular nominal automata). An *extended regular nondeterministic nominal automaton* $A = (Q, \rightarrow, s, f)$ consists of

- an orbit-finite nominal set Q of *states* with *initial state* $s \in Q$,
- an equivariant subset $\rightarrow \subseteq Q \times (\overline{\mathbb{A}} \cup \{\epsilon\}) \times Q$,
- an equivariant function $f : Q \rightarrow \{\perp, \top, \epsilon\}$ such that for all $q \in Q$, $f(q) = \perp$ whenever q has some outgoing transition and $\text{supp}(q) = \emptyset$ whenever $f(q) = \top$,

such that \rightarrow is α -invariant (that is, $q \xrightarrow{la} q'$ and $\langle a \rangle q' = \langle b \rangle q''$ imply $q \xrightarrow{lb} q''$) and finitely branching up to α -equivalence (that is, the sets $\{(a, q') \mid q \xrightarrow{a} q'\}$, $\{(\epsilon, q') \mid q \xrightarrow{\epsilon} q'\}$ and $\{\langle a \rangle q' \mid q \xrightarrow{la} q'\}$ are finite). The *degree* 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, 28]).

A *run-in-context* of A on a bar string $w = \alpha_0 \alpha_1 \dots \alpha_m$ and a context $S_0 \subseteq \mathbb{A}$ is a sequence $\tau = (S_0, q_0), (S_1, q_1), \dots, (S_n, q_n)$ such that for all $0 \leq i < n$, we have $q_i \xrightarrow{\beta_i} q_{i+1}$ for some $\beta_i \in \overline{\mathbb{A}} \cup \{\epsilon\}$; we also require that $S_{i+1} = S_i$ if $\beta_i \in \mathbb{A}$ and $S_{i+1} = S_i \cup \{a\}$ if $\beta_i = la$, and that $\beta_0 \beta_1 \dots \beta_m|_{\overline{\mathbb{A}}}$ is a prefix of w , where $v|_{\overline{\mathbb{A}}}$ denotes bar string that is obtained from the word $v \in (\overline{\mathbb{A}} \cup \{\epsilon\})^*$ by dropping all occurrences of ϵ . In this case we refer to $\beta_0 \beta_1 \dots \beta_m|_{\overline{\mathbb{A}}}$ as the *run word* of τ . A run-in-context τ of length n on a bar string w of length m and a context S is *accepting* if either the run word is w and $f(q_n) = \epsilon$, or $f(q_n) = \top$ and $v \in \text{free}(S_n)$, where v is obtained by taking the postfix of w that remains after the run word has been read. Given a context $\text{supp}(s) \subseteq S \subseteq \mathbb{A}$, the automaton A *literally accepts* the language

$$L_0(A, S) = \{w \in \overline{\mathbb{A}}^* \mid \text{there is an accepting run of } A \text{ on } w \text{ in context } S\}$$

in context S . The *bar language accepted by A in context S* is the quotient

$$L_\alpha(A, S) = L_0(A, S) / \equiv_\alpha.$$

We say that A is ϵ -free if A contains no ϵ -transitions. If A is ϵ -free and contains no state q such that $f(q) = \top$, then A is just an RNNA and the accepted language does not depend on the context S .

Lemma 5.2. *Let A be an ERNNA with set Q of states, let $q, q' \in Q$, let $a \in \mathbb{A}$. Then the following hold.*

- (1) If $q \xrightarrow{\epsilon} q'$, then $\text{supp}(q') \subseteq \text{supp}(q)$.
- (2) If $q \xrightarrow{a} q'$, then $\text{supp}(q') \cup \{a\} \subseteq \text{supp}(q)$.
- (3) If $q \xrightarrow{la} q'$, then $\text{supp}(q') \subseteq \text{supp}(q) \cup \{a\}$.

Remark 5.3. It is easy to see that ϵ -transitions can be eliminated from an ERNNA in the usual fashion; the key point to note is that by the above lemma, the support of states cannot increase along ϵ -transitions, so that ϵ -elimination preserves the conditions on finite branching.

Example 5.4. The literal languages of ERNNA are in general not closed under α -equivalence, as can be seen on, e.g., the ERNNA (similar to the corresponding example RNNA in [28]) denoted by

$$s \xrightarrow{la} t(a) \xrightarrow{lb} u(a, b) \xrightarrow{b} v,$$

where $f(v) = \epsilon$, i.e. v is an accepting state; by $t(a)$, $u(a, b)$ we mean to denote orbits of the state space arising by renaming the indicated names a, b in accordance with the equivariance and α -invariance conditions in the definition of ERNNA. The automaton accepts the bar string $|a|bb$ since we have transitions $s \xrightarrow{la} t(a)$, $t(a) \xrightarrow{lb} u(a, b)$ and $u(a, b) \xrightarrow{b} v$. On the other hand, the automaton does not accept the α -equivalent bar string $|b|bb$ since after the transition $s \xrightarrow{lb} t(b)$, we do have a transition $t(b) \xrightarrow{lc} u(b, c)$, in which however c cannot be renamed to b as b occurs freely in the target state.

We next adapt the name-dropping construction for RNNA [28], which is aimed at regaining closure of the literal language under α -equivalence, to ERNNA. E.g. in terms of the above example, the name dropping construction will just offer the option of forgetting the name b in $u(b, c)$, thus unblocking the desired α -renaming.

Definition 5.5. Given an ϵ -free ERNNA $A = (Q, \rightarrow, s, f)$ of degree k with n orbits, we define the *name-dropping* ERNNA $\text{nd}(A) = (Q', \rightarrow', s', f')$ by putting

$$Q' = \{q|_N \mid q \in Q, N \subseteq \text{supp}(q)\},$$

where $q|_N := \text{Fix}(N)q$; here, $\text{Fix}(N)q$ denotes the orbit of q under $\text{Fix}(N)$. We put $s' = s|_{\text{supp}(s)}$ and $f(q|_N) = f(q)$. As transitions, we take

- $q|_N \xrightarrow{a'} q'|_{N'}$, whenever $q \xrightarrow{a} q'$, $N' \subseteq N$, and $a \in N$, and
- $q|_N \xrightarrow{la'} q'|_{N'}$, whenever $q \xrightarrow{lb} q''$, $N'' \subseteq \text{supp}(q'') \cap (N \cup \{b\})$, and $\langle a \rangle(q'|_{N'}) = \langle b \rangle(q''|_{N''})$.

As in [28], the name-dropping construction does not change the bar languages of automata, but it closes the accepted literal language under α -equivalence.

Lemma 5.6. *Given an ϵ -free ERNNA A of degree k and with n orbits, a bar string $w \in \overline{\mathbb{A}}^*$, and a context $S \subseteq \mathbb{A}$, we have*

- (1) $L_\alpha(A, S) = L_\alpha(\text{nd}(A), S)$
 - (2) $[w]_\alpha \in L_\alpha(\text{nd}(A), S)$ if and only if $w \in L_0(A, S)$.
- Furthermore, $\text{nd}(A)$ is of degree k and has at most $n2^k$ orbits.

We now translate formulae ϕ to equivalent extended regular nondeterministic nominal automata $A(\phi)$, in what amounts to a tableau construction that follows a similar spirit as the standard automata-theoretic translation of LTL. Alternatively, one may morally view $A(\phi)$ as a nondeterminization of an alternating automaton whose states are subformulae of ϕ . In either view, states in $A(\phi)$ are sets of α -renamings of formulae from $\text{cl}(\phi)$, read conjunctively.

Definition 5.7. Let $\phi \in \text{Bar}$ be a formula and put $\text{cl} = \text{cl}(\phi)$. We define $A(\phi) = (Q, \rightarrow, s, f)$ by putting $Q = \{\pi \cdot \Gamma \mid \Gamma \in \mathcal{P}(\text{cl})\}$ where $\pi \cdot \Gamma$ denotes pointwise application of π to the elements of Γ , and $s = \{\phi\}$. We define **prestates** to be the set of all sets $\Gamma \in Q$ such that Γ contains some formula of the shape $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$, or $\mu X. \psi_1$ and put **states** $= Q \setminus \text{prestates}$. For $\Gamma \in \text{prestates}$, we put $f(\Gamma) = \perp$. Let $\psi_1, \psi_2 \in \text{cl}$. If $\psi_1 \vee \psi_2 \in \Gamma$, then we add transitions $\Gamma \xrightarrow{\epsilon} (\Gamma \setminus \{\phi\}) \cup \{\psi_i\}$, for $i \in \{1, 2\}$; if $\psi_1 \wedge \psi_2 \in \Gamma$, then we add a transition $\Gamma \xrightarrow{\epsilon} (\Gamma \setminus \{\phi\}) \cup \{\psi_1, \psi_2\}$; and if $\mu X. \psi_1 \in \Gamma$, then we add a transition $\Gamma \xrightarrow{\epsilon} (\Gamma \setminus \{\phi\}) \cup \{\psi_1[X \mapsto \mu X. \psi_1]\}$. For $\Gamma \in \text{states}$, we put

$$f(\Gamma) = \begin{cases} \top & \text{if } \Gamma = \{\top\}, \\ \epsilon & \text{if } \Gamma = \{\epsilon\} \text{ or } \Gamma = \{\epsilon, \top\}, \\ \perp & \text{otherwise.} \end{cases}$$

Let $\Gamma \setminus \{\top\} = \{\psi_1, \dots, \psi_o\}$. If there is $a \in \mathbb{A}$ such that $\psi_i = \Diamond_a \chi_i$ for all $1 \leq i \leq o$, then we add a transition $\Gamma \xrightarrow{a} \{\chi_1, \dots, \chi_o\}$. On the other hand, if $\psi_i = \Diamond_{b_i} \chi_i$ for all $1 \leq i \leq o$, then we add, for each $a \in \mathbb{A}$ and all formulae $\theta_1, \dots, \theta_o$ such that $\langle b_i \rangle \chi_i = \langle a \rangle \theta_i$ for all $1 \leq i \leq o$, a transition $\Gamma \xrightarrow{a} \{\theta_1, \dots, \theta_o\}$.

We first show that this construction, illustrated in Example 6.4, does the intended job, i.e. produces an ERNNA accepting precisely the bar language of ϕ :

Lemma 5.8. *The structure $A(\phi)$ is an ERNNA of degree $\deg(\phi)$ and with at most $2^{|\text{cl}(\phi)|}$ orbits.*

Lemma 5.9. *For all bar strings $w \in \overline{\mathbb{A}}^*$, all closed formulae $\phi \in \text{Bar}$ and all contexts $S \subseteq \mathbb{A}$, we have*

$$[w]_\alpha \in L_\alpha(A(\phi), S) \text{ if and only if } w \in \llbracket S \vdash \phi \rrbracket.$$

In one direction, the proof of this result proceeds by induction over accepting runs and shows that for macro-nodes q in $A(\phi)$ that accept some bar string v in context S , every formula that is contained within q is satisfied by S, v . For the converse direction, the proof inductively constructs an accepting run of $A(\phi)$ in context S on w , relying on $w \in \llbracket S \vdash \phi \rrbracket$ to ensure that suitable transitions exist and that the run ends in an accepting state.

Remark 5.10. The above lemma implies in particular that under local freshness semantics, $\text{Bar-}\mu\text{TL}$ is indeed a fragment of Freeze LTL, via the translation of RNNA into register automata [28]. Details are in the appendix.

By Lemma 5.9 and Lemma 5.6 we have

Corollary 5.11. *For all closed formulae $\phi \in \text{Bar}$ and contexts $S \subseteq \mathbb{A}$, we have*

$$\llbracket S \vdash \phi \rrbracket = L_0(\text{nd}(A(\phi)), S),$$

where $\text{nd}(A(\phi))$ is of degree $\deg(\phi)$ and has at most $2^{|\text{cl}(\phi)| + \deg(\phi)}$ orbits.

6 Model Checking for $\text{Bar-}\mu\text{TL}$

Next, we reduce model checking for $\text{Bar-}\mu\text{TL}$ to inclusion checking between RNNA and ERNNA and show that the latter problem is in EXPSpace.

Definition 6.1 (Model checking problem). Let A be an RNNA with initial state s , and let $\phi \in \text{Bar}$. We say that A *satisfies* ϕ (denoted by $A \models \phi$) if $L_\alpha(A) \subseteq \llbracket S \vdash \phi \rrbracket_\alpha$ where $S = \text{supp}(s) \cup \text{FN}(\phi)$ and $\llbracket S \vdash \phi \rrbracket_\alpha = \{[w]_\alpha \mid w \in \llbracket S \vdash \phi \rrbracket\}$. The *model checking problem* (for emphasis: *under bar language semantics*) consists in deciding whether $A \models \phi$ for given A, ϕ . Similarly, *model checking under local freshness* is the problem to decide whether $D(L_\alpha(A)) \subseteq D(\llbracket S \vdash \phi \rrbracket_\alpha)$.

(We know that the RNNA A accepts only words whose free names are contained in $\text{supp}(s)$ [28], so it suffices to consider the case where the context of ϕ is $\text{supp}(s) \cup \text{FN}(\phi)$.) By Lemma 5.9 we have

$$A \models \phi \text{ if and only if } L_\alpha(A) \subseteq L_\alpha(A(\phi), S).$$

Hence model checking a formula ϕ against an RNNA A reduces to inclusion checking between an RNNA and an ERNNA.

Theorem 6.2. *The inclusion problem between RNNA and ERNNA is in EXPSpace, and more precisely in para-PSPACE: Given an RNNA A_1 with initial state s_1 and an ERNNA A_2 with initial state s_2 , the inclusion $L_\alpha(A_1) \subseteq L_\alpha(A_2, S)$, where $S = \text{supp}(s_1) \cup \text{supp}(s_2)$, can be checked using space polynomial in the number of orbits of A_1 and A_2 , and exponential in $\deg(A_1)$ and $\deg(A_2)$. The same holds under local freshness semantics, i.e. for checking whether $D(L_\alpha(A_1)) \subseteq D(L_\alpha(A_2, S))$.*

The proof of this result first transforms A_1 to an equivalent *bar NFA* A'_1 (see [28] for more details) and then exhibits an NEXPSpace-algorithm that checks for *non-inclusion*, that is, that checks $L_\alpha(A'_1) \not\subseteq L_\alpha(A_2, S)$. In order to restrict the search space in A_2 , the algorithm checks for non-inclusion in the equivalent automaton $\text{nd}(A_2)$ whose bar language is closed under α -equivalence so that the algorithm only has to explore those states in $\text{nd}(A_2)$ that can be reached by literally following transitions in A'_1 , yielding the stated upper bound on memory

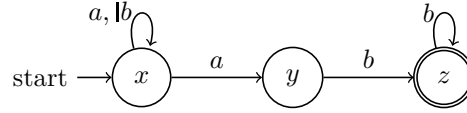
usage. The algorithm for local freshness semantics differs only by allowing a plain name a (read by A'_1) to be matched by either an a -transition or a $|a$ -transition in A_2 .

Since $A(\phi)$ is, by Lemma 5.8, of degree $\deg(\phi)$ and has at most 2^n orbits, where $n = |\text{cl}(\phi)|$, we obtain a model checking algorithm for $\text{Bar-}\mu\text{TL}$ witnessing an upper bound EXPSpace , or again more precisely para-PSpace :

Corollary 6.3. *The model checking problem for $\text{Bar-}\mu\text{TL}$, under both local freshness and bar language semantics, is in EXPSpace , more precisely in para-PSpace with formula size and the degrees of the automaton and the formula as the parameter; more precisely, $A \models \phi$ can be checked in space polynomial in the number of orbits of A , and exponential in $|\text{cl}(\phi)|$ and $\deg(A)$.*

The same bounds follow for model checking under global freshness semantics (cf. Section 3), restricted to closed formulae and RNNAs whose initial states have empty support.

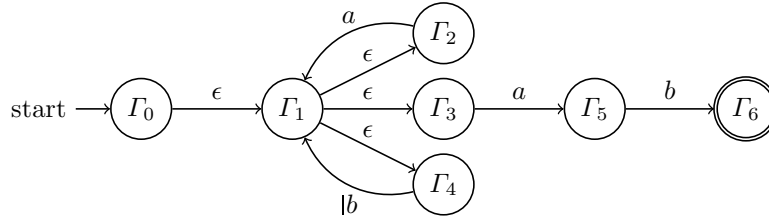
Example 6.4. Let A be the RNA



with depicted states to be read as representing orbits that arise by the action of G on formulae; A accepts bar strings of the abstract shape $(a + |b)^*ab^+$, for instance $|cacc$, $aal|b|cacc$ and $alaaa$. Furthermore, let

$$\phi = \mu X. (\Diamond_a \Diamond_b \top \vee \Diamond_a X \vee \Diamond_{|b} X)$$

Interpreted over A , the formula ϕ expresses that the substring ab eventually occurs. We translate ϕ to the ERRNA $A(\phi)$ depicted by



where $\Gamma_0 = \{\phi_1\}$ and

$$\begin{aligned} \Gamma_1 &= \{\Diamond_a \Diamond_b \top \vee \Diamond_a \phi_1 \vee \Diamond_{|b} \phi_1\} & \Gamma_2 &= \{\Diamond_a \phi_1\} & \Gamma_3 &= \{\Diamond_a \Diamond_b \top\} \\ \Gamma_4 &= \{\Diamond_{|b} \phi_1\} & \Gamma_5 &= \{\Diamond_b \top\} & \Gamma_6 &= \{\top\} \end{aligned}$$

and $f(\Gamma_6) = \top$, that is, in context S , Γ_6 accepts any bar string from $\text{free}(S)$. To check $A \models \phi$, our algorithm then transforms A to an equivalent NFA A' with alphabet $\{a, |b, b\}$ (i.e. a *bar NFA* as mentioned above [28]), takes the name-dropping variant $\text{nd}(A(\phi))$ of $A(\phi)$ and then nondeterministically searches for

an accepting run of A' that cannot be literally replicated in $\text{nd}(A(\phi))$. Since all bar strings that are accepted by A are of the shape $(a + |b)^*ab^+$, $\text{nd}(A(\phi))$ has accepting literal runs for all words that are accepted by A' . E.g. for the bar string $|bab$, the automaton $\text{nd}(A(\phi))$ can – since it is name-dropping – reach the accepting state $\{\top\}$ after having read $|bab$.

Remark 6.5. The *satisfiability problem* for input $\phi \in \text{Bar}$ and $S \subseteq \mathbb{A}$ consists in checking whether $\llbracket S \vdash \phi \rrbracket \neq \emptyset$. As shown above, we have $\llbracket S \vdash \phi \rrbracket \neq \emptyset$ if and only if $L_0(\text{nd}(A(\phi)), S) \neq \emptyset$. Since $L_0(\text{nd}(A(\phi)), S)$ is closed under α -equivalence, non-emptiness checking for this automaton amounts to checking whether some accepting state is reachable from its initial state. This can be done by a depth-first search using a polynomial amount of memory, which shows that the satisfiability problem for $\text{Bar-}\mu\text{TL}$ is in PSPACE.

Since $\text{Bar-}\mu\text{TL}$ encodes only support-relative negation, validity checking does not reduce to satisfiability checking; rather, it reduces to model checking over an RNNA accepting the bar language $(|a)^*$, whose local freshness semantics is the universal language. Thus, the validity problem of $\text{Bar-}\mu\text{TL}$ under local freshness semantics is in EXPSpace. Under bar language semantics, formulae can only be valid relative to a given support (in the sense that $\llbracket S \vdash \phi \rrbracket = \text{free}(S)$); we leave decidability of relative validity in bar language semantics as an open problem.

7 Conclusions

We have defined a specification logic $\text{Bar-}\mu\text{TL}$ for finite words over infinite alphabets; $\text{Bar-}\mu\text{TL}$ is a variant of Freeze LTL distinguished by enforcing a lossiness property called name dropping, in analogy to regular nondeterministic nominal automata (RNNA) [28]. We have moreover introduced a slight modification of RNNA called *extended regular nondeterministic nominal automata (ERNNA)*, whose non-emptiness problem is in PSPACE; the inclusion problem for RNNA in ERNNA is in parametrized PSPACE. By a translation of $\text{Bar-}\mu\text{TL}$ into ERNNA, the same bounds then follow for the $\text{Bar-}\mu\text{TL}$ satisfiability and for $\text{Bar-}\mu\text{TL}$ model checking over RNNA. By comparison, full Freeze LTL is undecidable, and the fragment with at most one register (and only future operators) is decidable but not primitive recursive [7]. Name dropping implies that $\text{Bar-}\mu\text{TL}$ is less expressive than Freeze LTL; nevertheless, $\text{Bar-}\mu\text{TL}$ retains a reasonable level of expressiveness, and in particular can express many languages that fail to be expressible using only one register (such as the language ‘the first two letters appear again, adjacent and in the same order’ [27]).

An important issue for future work is the behaviour of $\text{Bar-}\mu\text{TL}$ over infinite words; also, we will investigate whether name dropping can be usefully applied to languages of data trees (e.g. [10]).

References

1. Bojanczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data words. *ACM Trans. Comput. Log.* **12**(4), 27:1–27:26 (2011). <https://doi.org/10.1145/1970398.1970403>
2. Bojanczyk, M., Klin, B., Lasota, S.: Automata theory in nominal sets. *Log. Methods Comput. Sci.* **10** (2014). [https://doi.org/10.2168/LMCS-10\(3:4\)2014](https://doi.org/10.2168/LMCS-10(3:4)2014)
3. Bollig, B., Habermehl, P., Leucker, M., Monmege, B.: A robust class of data languages and an application to learning. *Log. Meth. Comput. Sci.* **10** (2014). [https://doi.org/10.2168/LMCS-10\(4:19\)2014](https://doi.org/10.2168/LMCS-10(4:19)2014)
4. Bollig, B., Quaas, K., Sangnier, A.: The complexity of flat freeze LTL. *Log. Methods Comput. Sci.* **15**(3) (2019). [https://doi.org/10.23638/LMCS-15\(3:33\)2019](https://doi.org/10.23638/LMCS-15(3:33)2019)
5. Colcombet, T., Manuel, A.: Generalized data automata and fixpoint logic. In: 34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15–17, 2014, New Delhi, India. LIPIcs, vol. 29, pp. 267–278. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2014). <https://doi.org/10.4230/LIPIcs.FSTTCS.2014.267>
6. Colcombet, T., Manuel, A.: Fragments of fixpoint logic on data words. In: Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015. LIPIcs, vol. 45, pp. 98–111. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2015). <https://doi.org/10.4230/LIPIcs.FSTTCS.2015.98>
7. Demri, S., Lazic, R.: LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.* **10** (2009). <https://doi.org/10.1145/1507244.1507246>
8. Demri, S., Sangnier, A.: When model-checking freeze LTL over counter machines becomes decidable. In: Foundations of Software Science and Computational Structures, FOSSACS 2010. LNCS, vol. 6014, pp. 176–190. Springer (2010). https://doi.org/10.1007/978-3-642-12032-9_13
9. Figueira, D.: Alternating register automata on finite words and trees. *Log. Methods Comput. Sci.* **8**(1) (2012). [https://doi.org/10.2168/LMCS-8\(1:22\)2012](https://doi.org/10.2168/LMCS-8(1:22)2012)
10. Figueira, D., Segoufin, L.: Future-looking logics on data words and trees. In: Mathematical Foundations of Computer Science, MFCS 2009. LNCS, vol. 5734, pp. 331–343. Springer (2009). https://doi.org/10.1007/978-3-642-03816-7_29
11. Gabbay, M., Pitts, A.: A new approach to abstract syntax involving binders. In: Logic in Computer Science, LICS 1999. pp. 214–224. IEEE Computer Society (1999)
12. Gabbay, M.J.: Foundations of nominal techniques: logic and semantics of variables in abstract syntax. *Bull. Symb. Log.* **17**(2), 161–229 (2011). <https://doi.org/10.2178/bsl/1305810911>
13. Gabbay, M.J., Ciancia, V.: Freshness and name-restriction in sets of traces with names. In: Foundations of Software Science and Computational Structures, FOSSACS 2011. LNCS, vol. 6604, pp. 365–380. Springer (2011). <https://doi.org/10.1007/978-3-642-19805-2>
14. Gabbay, M.J., Ghica, D.R., Petrisan, D.: Leaving the nest: Nominal techniques for variables with interleaving scopes. In: Computer Science Logic, CSL 2015. LIPIcs, vol. 41, pp. 374–389. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
15. Grigore, R., Distefano, D., Petersen, R., Tzevelekos, N.: Runtime verification based on register automata. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2013. LNCS, vol. 7795, pp. 260–276. Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_19

16. Groote, J., Mateescu, R.: Verification of temporal properties of processes in a setting with data. In: Algebraic Methodology and Software Technology, AMAST 1998. LNCS, vol. 1548, pp. 74–90. Springer (1998). https://doi.org/10.1007/3-540-49253-4_8
17. Groote, J., Willemse, T.: Model-checking processes with data. *Sci. Comput. Prog.* **56**(3), 251–273 (2005). <https://doi.org/10.1016/j.scico.2004.08.002>
18. Grumberg, O., Kupferman, O., Sheinvald, S.: Model checking systems and specifications with parameterized atomic propositions. In: Chakraborty, S., Mukund, M. (eds.) Automated Technology for Verification and Analysis. LNCS, vol. 7561, pp. 122–136. Springer (2012). https://doi.org/10.1007/978-3-642-33386-6_11
19. Howar, F., Jonsson, B., Vaandrager, F.: Combining black-box and white-box techniques for learning register automata. In: Computing and Software Science – State of the Art and Perspectives, LNCS, vol. 10000, pp. 563–588. Springer (2019). https://doi.org/10.1007/978-3-319-91908-9_26
20. Kaminski, M., Francez, N.: Finite-memory automata. *Theor. Comput. Sci.* **134**(2), 329–363 (1994). [https://doi.org/10.1016/0304-3975\(94\)90242-9](https://doi.org/10.1016/0304-3975(94)90242-9)
21. Klin, B., Lelyk, M.: Scalar and vectorial mu-calculus with atoms. *Log. Methods Comput. Sci.* **15**(4) (2019), <https://lmcs.episciences.org/5877>
22. Kürtz, K., Küsters, R., Wilke, T.: Selecting theories and nonce generation for recursive protocols. In: Formal methods in security engineering, FMSE 2007. pp. 61–70. ACM (2007)
23. Manuel, A., Muscholl, A., Puppis, G.: Walking on data words. *Theory Comput. Sys.* **59**, 180–208 (2016). <https://doi.org/10.1007/s00224-014-9603-3>
24. Mottet, A., Quaas, K.: The containment problem for unambiguous register automata. In: Theoretical Aspects of Computer Science, STACS 2019. LIPIcs, vol. 126, pp. 53:1–53:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2019). <https://doi.org/10.4230/LIPIcs.STACS.2019.53>
25. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.* **5**, 403–435 (2004). <https://doi.org/10.1145/1013560.1013562>
26. Pitts, A.: Nominal Sets: Names and Symmetry in Computer Science. Cambridge University Press (2013)
27. Schröder, L., Kozen, D., Milius, S., Wißmann, T.: Nominal automata with name binding. *CoRR* **abs/1603.01455** (2016), <http://arxiv.org/abs/1603.01455>
28. Schröder, L., Kozen, D., Milius, S., Wißmann, T.: Nominal automata with name binding. In: Foundations of Software Science and Computation Structures, FOSSACS 2017. LNCS, vol. 10203, pp. 124–142 (2017). https://doi.org/10.1007/978-3-662-54458-7_8
29. Song, F., Wu, Z.: On temporal logics with data variable quantifications: Decidability and complexity. *Inf. Comput.* **251**, 104–139 (2016). <https://doi.org/10.1016/j.ic.2016.08.002>
30. Stockhusen, C., Tantau, T.: Completeness results for parameterized space classes. In: Parameterized and Exact Computation, IPEC 2013. LNCS, vol. 8246, pp. 335–347. Springer (2013). <https://doi.org/10.1007/978-3-319-03898-8>
31. Turner, D., Winskel, G.: Nominal domain theory for concurrency. In: Computer Science Logic, CSL 2009. pp. 546–560 (2009)

A Appendix

A.1 Details for Section 4

Proof of Lemma 4.2

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. \square

Proof of Lemma 4.6

Proof. In both claims, the proof is by induction along the recursive definition $S, w \models \phi$, exploiting that this definition terminates as explained just after the definition of the semantics.

(1) For \top , \perp , ϵ and $\neg\epsilon$ the statement holds trivially. For conjunction, disjunction, a -operators, and fixpoints, one simply uses the induction hypothesis. For \mathbf{la} -operators, the claim is immediate from the definition of the semantics.

(2) Again, the claim is trivial for \top , \perp , and ϵ , and $\neg\epsilon$, and the inductive steps for \wedge and \vee are straightforward. The case for fixpoints is immediate from the induction hypothesis and Lemma 4.2.

For $\phi = \Diamond_a \chi$, we have $S, w \models \Diamond_a \chi$ so that there is w' such that $w = aw'$ and $S, w' \models \chi$. As $\phi \equiv_\alpha \phi'$, we have $\phi' = \Diamond_a \chi'$ for some χ' such that $\chi \equiv_\alpha \chi'$. The induction hypothesis finishes the case.

For $\phi = \Diamond_{\mathbf{la}} \chi$, we have $S, w \models \Diamond_{\mathbf{la}} \chi$ so that there are $b \in \mathbb{A}$, $\psi \in \mathbf{Bar}$, and $v \in \overline{\mathbb{A}}^*$ such that $w = \mathbf{lb}v$, $\Diamond_{\mathbf{la}} \chi \equiv_\alpha \Diamond_{\mathbf{lb}} \psi$ and $S \cup \{b\}, v \models \psi$. Since $\phi \equiv_\alpha \phi'$, there are $c \in \mathbb{A}$, $\chi' \in \mathbf{Bar}$ such that $\phi' = \Diamond_{\mathbf{lc}} \chi'$ and $\chi = (bc) \cdot \chi'$. Hence, it suffices to show that there are $\psi' \in \mathbf{Bar}$, $d \in \mathbb{A}$ and $v' \in \overline{\mathbb{A}}^*$ such that $w = \mathbf{ld}v'$, $\Diamond_{\mathbf{lc}} \chi' \equiv_\alpha \Diamond_{\mathbf{ld}} \psi'$ and $S \cup \{d\}, v' \models \psi'$. Pick $\psi' = \psi$, $v' = v$ and $d = b$ so that $w = \mathbf{lb}v = \mathbf{ld}v'$ and $S \cup \{d\}, v' \models \psi'$. We also have $\Diamond_{\mathbf{lc}} \chi' \equiv_\alpha \Diamond_{\mathbf{la}} \chi \equiv_\alpha \Diamond_{\mathbf{lb}} \psi = \Diamond_{\mathbf{ld}} \psi'$, as required. \square

Proof of Lemma 4.10

Proof. We have $S, w \models \neg\epsilon$ if and only if w is a non-empty bar string that uses only free letters from S . This is the case if and only if w is non-empty and the first letter in w is either a free letter b such that $b \in S$ or a bound letter \mathbf{lc} for some $c \in \mathbb{A}$. This in turn is the case if and only if $S, w \models \bigvee_{b \in S} \Diamond_b \top \vee \Diamond_{\mathbf{lc}} \top$.

We have $a \in S$ so that $S, w \models \Box_a \psi$ if and only if w is the empty bar string ϵ , or a non-empty bar string that starts with some free letter $b \in S$ such that $a \neq b$ or with some bound letter \mathbf{lc} , or we have $w = aw'$ and $S, w' \models \psi$. This in turn is the case if and only if $S, w \models \epsilon$ or $S, w \models \bigvee_{b \in S \setminus \{a\}} \Diamond_b \top$ or $S, w \models \Diamond_{\mathbf{lc}} \top$ or $S, w \models \Diamond_a \psi$.

Lastly, we have $S, w \models \Box_{\mathbf{la}} \psi$ if and only if $w = \epsilon$ or w starts with some free letter $b \in S$, or we have $w = \mathbf{lc}w'$ for some $c \in \mathbb{A}$ and $S \cup \{c\}, w' \models \chi$ for some $\chi \in \mathbf{Bar}$ such that $\langle a \rangle \psi = \langle b \rangle \chi$. This in turn is the case if and only if $S, w \models \epsilon$ or $S, w \models \bigvee_{b \in S} \Diamond_b \top$ or $S, w \models \Diamond_{\mathbf{lc}} \psi$. \square

A.2 Details for Section 5**Proof of Lemma 5.2**

Proof. (1) Let $Z = \{(\epsilon, q') \mid q \xrightarrow{\epsilon} q'\}$. Then we have

$$\text{supp}(q') = \text{supp}(\epsilon, q') \subseteq \text{supp}(Z) \subseteq \text{supp}(q),$$

where the first inclusion holds since Z is ufs and the second inclusion holds by equivariance of \rightarrow .

(2) Let $Z = \{(a, q') \mid q \xrightarrow{a} q'\}$. Then we have

$$\text{supp}(q') \cup \{a\} = \text{supp}(a, q') \subseteq \text{supp}(Z) \subseteq \text{supp}(q),$$

where the first inclusion holds since Z is ufs and the second inclusion holds by equivariance of \rightarrow .

(3) Let $Z = \{\langle a \rangle q' \mid q \xrightarrow{!a} q'\}$. Then we have

$$\text{supp}(q') \subseteq \text{supp}(\langle a \rangle q') \cup \{a\} \subseteq \text{supp}(Z) \cup \{a\} \subseteq \text{supp}(q) \cup \{a\},$$

where the first inclusion holds since $\text{supp}(\langle a \rangle q') = \text{supp}(q') \setminus \{a\}$, the second inclusion holds since Z is ufs and the third inclusion holds by equivariance of the relation \rightarrow . \square

Proof of Lemma 5.6

Proof. The proof works just as the proof of the Lemmas 5.7 and Lemma 5.8. in [28], with the only difference being that, in contrast to RNNA, ERNNA may contain states q such that $f(q) = \top$. By definition of ERNNA, such nodes have empty support and no outgoing transitions so that the name-dropping construction does not change anything for these nodes. \square

Proof of Lemma 5.8

Proof. Since cl is a nominal set, the set Q is a nominal set as well and we have at most $2^{|\text{cl}(\phi)|}$ orbits since every $\Gamma \in \mathcal{P}(\text{cl})$ gives rise to at most one orbit in Q . The sets **states** and **prestates** are equivariant, and since renaming does not affect formulae \top and ϵ , f is equivariant as well. The transition relation \rightarrow is equivariant, α -invariant and finitely branching up to α -equivalence by construction. \square

Proof of Lemma 5.9

Proof. Without loss of generality, we assume that w is clean, that is, that every name is bound at most once in w (this indeed is without loss of generality since both the bar language of automata and the languages of formulae are closed under α -equivalence).

Let $S, w \models \phi$. It suffices to construct a run τ of $A(\phi)$ on w in the context S . We show the more general property that there is, for all nodes Γ in $A(\phi)$, all postfixes v of w , and all contexts $S \subseteq \mathbf{FN}(\phi)$ such that $S, v \models \psi$ for all $\psi \in \Gamma$, an accepting run τ of $A(\phi)$ on v in context S . We construct a suitable run by induction, using the measure $(|v|, \mathbf{u}(\Gamma), |\Gamma|)$, in lexicographic ordering, where $\mathbf{u}(\Gamma)$ denotes maximum of the numbers of unguarded fixpoint operators of formulae in Γ , and $|\Gamma|$ denotes the sum of the sizes of formulae in Γ . If $\Gamma \in \mathbf{states}$, then we distinguish the following cases (if $|v| = 0$, then we have $v = \epsilon$ and every formula in Γ is satisfied by ϵ , that is, we are either in case (1) or case (2) below, showing that the inductive construction eventually terminates and yields an accepting run).

- (1) If $\Gamma = \{\top\}$, then we have $v \in \mathbf{free}(S)$ since $S, v \models \top$. We are done since $\tau = (S, \Gamma)$ is an accepting run as $f(\Gamma) = \top$ by construction.
- (2) If $\Gamma = \{\epsilon, \top\}$ or $\Gamma = \{\epsilon\}$, then we have $v = \epsilon$ since $S, v \models \epsilon$. We are done since $\tau = (S, \Gamma)$ is an accepting run as $f(\Gamma) = \epsilon$ by construction and the run word of τ is $\epsilon = v$.
- (3) If $\Gamma \setminus \{\top\}$ contains some formula of the shape $\Diamond_a \chi$, then we have $v = av'$ and all formulae in Γ are \Diamond_a -formulae since we have $S, v \models \psi$ for each $\psi \in \Gamma$ (that is, Γ does not contain ϵ , \perp and also no formula of the shape $\Diamond_{\mathbf{b}} \chi$). Let $\Gamma = \{\Diamond_a \chi_1, \dots, \Diamond_a \chi_o\}$. By the semantics of \Diamond_a -operators, we have $S, v' \models \chi_i$ for each $\Diamond_a \chi_i \in \Gamma$. As $|v'| < |v|$, we have – by the inductive hypothesis – an accepting run τ' on v' in context S that starts at the node $\{\chi_1, \dots, \chi_o\}$. Since $\Gamma \xrightarrow{a} \{\chi_1, \dots, \chi_o\}$ by definition of $A(\phi)$, $\tau = (S, \Gamma), \tau$ is an accepting run on v' in context S that starts at Γ .
- (4) If $\Gamma \setminus \{\top\}$ contains some formula of the shape $\Diamond_{\mathbf{b}} \chi$, then we have $v = |cv'$ for some $c \in \mathbb{A}$ by the semantics of $\Diamond_{\mathbf{b}}$ -operators. Then all formulae in Γ are of this shape as $S, v \models \psi$ for each $\psi \in \Gamma$. Let $\Gamma = \{\Diamond_{\mathbf{b}_1} \chi_1, \dots, \Diamond_{\mathbf{b}_o} \chi_o\}$. We also have, for each $\Diamond_{\mathbf{b}_i} \chi_i \in \Gamma$, some formula θ_i such that $\langle \mathbf{b}_i \rangle \chi_i = \langle c \rangle \theta_i$ and $S \cup \{c\}, v' \models \theta_i$. By definition of $A(\phi)$, there is a transition $\Gamma \xrightarrow{\mathbf{c}} \{\theta_1, \dots, \theta_o\}$. As $|v'| < |v|$, we have – by the inductive hypothesis – an accepting run τ' on v' in context $S \cup \{c\}$ that starts at the node $\{\theta_1, \dots, \theta_o\}$. Thus $\tau = (S, \Gamma), \tau$ is an accepting run on v' in context S that starts at Γ .

If $\Gamma \in \mathbf{prestates}$, then we pick some propositional formula $\psi \in \Gamma$.

- (1) If $\psi = \chi_1 \vee \chi_2$, then we have $S, v \models \psi$ if and only if $S, v \models \chi_1$ or $S, v \models \chi_2$. Pick $i \in \{1, 2\}$ such that $S, v \models \chi_i$. We have $\mathbf{u}(\Gamma) = \mathbf{u}((\Gamma \setminus \{\psi\}) \cup \{\chi_i\})$ and $|\chi_i| < |\psi|$ so that the inductive hypothesis yields an accepting run τ' on v in context S that starts at $(\Gamma \setminus \{\psi\}) \cup \{\chi_i\}$. By construction of $A(\phi)$, there is a transition $\Gamma \xrightarrow{\epsilon} (\Gamma \setminus \{\psi\}) \cup \{\chi_i\}$, showing that $\tau = (S, \Gamma), \tau'$ is an accepting run on v in context S that starts at Γ .
- (2) If $\psi = \chi_1 \wedge \chi_2$, then we have $S, v \models \psi$ if and only if $S, v \models \chi_1$ and $S, v \models \chi_2$. We have $\mathbf{u}(\Gamma) = \mathbf{u}((\Gamma \setminus \{\psi\}) \cup \{\chi_1, \chi_2\})$, $|\chi_1| < |\psi|$ and $|\chi_2| < |\psi|$ so that the inductive hypothesis yields an accepting run τ' on v in context S that starts at $(\Gamma \setminus \{\psi\}) \cup \{\chi_1, \chi_2\}$. By construction of $A(\phi)$, there is a transition $\Gamma \xrightarrow{\epsilon}$

$(\Gamma \setminus \{\psi\}) \cup \{\chi_1, \chi_2\}$, showing that $\tau = (S, \Gamma), \tau'$ is an accepting run on v in context S that starts at Γ .

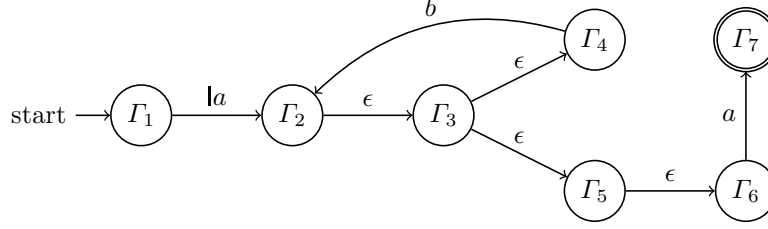
(3) If $\psi = \mu X. \chi_1$, then we have $S, v \models \psi$ if and only if $S, v \models \chi_1[X \mapsto \mu X. \chi_1]$. Fixpoint variables are guarded by modal operators so that $\mu X. \chi_1$ is guarded in $\chi_1[X \mapsto \mu X. \chi_1]$. Hence we have $u(\Gamma) > u((\Gamma \setminus \{\psi\}) \cup \{\chi_1[X \mapsto \mu X. \chi_1]\})$ and the inductive hypothesis yields an accepting run τ' on v in context S that starts at $(\Gamma \setminus \{\psi\}) \cup \{\chi_1[X \mapsto \mu X. \chi_1]\}$. By construction of $A(\phi)$, there is a transition $\Gamma \xrightarrow{\epsilon} (\Gamma \setminus \{\psi\}) \cup \{\chi_1[X \mapsto \mu X. \chi_1]\}$, showing that $\tau = (S, \Gamma), \tau'$ is an accepting run on v in context S that starts at Γ .

For the converse direction, let $\tau = (S_0, \Gamma_0), (S_1, \Gamma_1), \dots, (S_n, \Gamma_n)$ be an accepting run of $A(\phi)$ on w in context $S_0 = S$ and starting at $\Gamma_0 = \{\phi\}$. We have to show that $S, w \models \phi$. It suffices to show the more general property that for all contexts S' , all nodes $\Gamma \in Q$, all postfixes v of w and all formulae $\psi \in \{\pi \cdot \chi \mid \chi \in \text{cl}(\phi)\}$, if $\psi \in \Gamma$ and there is an accepting run τ of $A(\phi)$ on v in context S' that starts at Γ , then we have $S', v \models \psi$. The proof of this proceeds by induction on the length $|\tau|$ of τ . If $|\tau| = 1$, then Γ is an accepting state. By definition of f , we have either $f(\Gamma) = \top$ or $f(\Gamma) = \epsilon$. In the former case, we have $\Gamma = \{\top\}$ by definition of f and $\psi = \top$ since $\psi \in \Gamma$. Since τ is an accepting run in context S' , we have $\text{free}(v) \subseteq S'$ so that $S', v \models \top$, as required. In the latter case, we have $v = \epsilon$ since τ is an accepting run; furthermore we have either $\Gamma = \{\epsilon\}$ or $\Gamma = \{\top, \epsilon\}$ by definition of f so that ψ is either ϵ or \top . If $\psi = \epsilon$, then we have $S', v \models \epsilon$. If $\psi = \top$, then we have $S', v \models \top$ since $\epsilon \in \text{free}(S')$. If $|\tau| > 1$, then there is at least one transition tuple $(S', \Gamma), (S'', \Gamma')$ in τ . If $\Gamma \in \text{prestates}$ and $\psi \in \Gamma'$, then the inductive hypothesis finishes the case. If $\Gamma \in \text{prestates}$ and $\psi \notin \Gamma'$, then the ϵ -transition of the automaton manipulates ψ which then is of the shape $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$ or $\mu X. \psi_1$. Also we have $S' = S''$.

- If $\psi = \psi_1 \vee \psi_2$, then we have $\psi_1 \in \Gamma'$ or $\psi_2 \in \Gamma'$ by definition of \rightarrow . Since $S', v \models \psi_1 \vee \psi_2$ if and only if $S', v \models \psi_1$ or $S', v \models \psi_2$, we are done by the inductive hypothesis.
- If $\psi = \psi_1 \wedge \psi_2$, then we have $\psi_1 \in \Gamma'$ and $\psi_2 \in \Gamma'$ by definition of \rightarrow . Since $S', v \models \psi_1 \wedge \psi_2$ if and only if $S', v \models \psi_1$ and $S', v \models \psi_2$, we are done by the inductive hypothesis.
- If $\psi = \mu X. \psi_1$, then we have $\psi_1[X \mapsto \mu X. \psi_1] \in \Gamma'$ by definition of \rightarrow . Since $S', v \models \mu X. \psi_1$ if and only if $S', v \models \psi_1[X \mapsto \mu X. \psi_1]$, we are done by the inductive hypothesis.

If $\Gamma \in \text{states}$, then there is some $\alpha \in \overline{\mathbb{A}}$ such that Γ has an outgoing α -transition since τ does not end at Γ . As $\psi \in \Gamma$, we have $\psi = \Diamond_a \psi_1$ or $\psi = \Diamond_{\mathbf{1}a} \psi_1$ for some $a \in \mathbb{A}$, $\psi_1 \in \text{Bar}$ by definition of \rightarrow . If $\psi = \Diamond_a \psi_1$, then we have $v = av'$, $\psi_1 \in \Gamma'$ and $S' = S''$ so that $S', v' \models \psi_1$ by the inductive hypothesis which finishes the case. If $\psi = \Diamond_{\mathbf{1}a} \psi_1$, then we have $v = bv'$ for some $b \in \mathbb{A}$. By definition of \rightarrow , there is some $\theta \in \text{Bar}$ such that $\langle a \rangle \psi_1 = \langle b \rangle \theta$ and $\theta \in \Gamma'$. Since τ is a run in context S' , we have $S'' = S' \cup \{b\}$. By the inductive hypothesis, we have $S \cup \{b\}, v' \models \theta$ and are done. \square

Example A.1. Let $\chi = \mu X.(\Diamond_b X \vee (\Diamond_a \epsilon \wedge \Diamond_a \top))$. The formula $\psi = \Diamond_{\text{la}} \chi$ is translated to the ERNNA $A(\psi)$ denoted by



where

$$\begin{aligned}
 \Gamma_1 &= \{\psi\} & \Gamma_2 &= \{\chi\} & \Gamma_3 &= \{\Diamond_b \chi \vee (\Diamond_a \epsilon \wedge \Diamond_a \top)\} \\
 \Gamma_4 &= \{\Diamond_b \chi\} & \Gamma_5 &= \{\Diamond_a \epsilon \wedge \Diamond_a \top\} & \Gamma_6 &= \{\Diamond_a \epsilon, \Diamond_a \top\} & \Gamma_7 &= \{\epsilon, \top\}.
 \end{aligned}$$

Here, all states are to be read as representing orbits that arise by the action of G on formulae. We have $f(\Gamma_7) = \epsilon$ since Γ_7 contains only ϵ and \top , and $f(\Gamma_i) = \perp$ for all $i \neq 7$. For all contexts S that contain b , the automaton accepts all finite bar strings in which the first letter is some bar letter la , followed by arbitrary many letters b and in which the last letter is the initially bound letter a . This also is the language of ψ .

A.3 Details for Section 6

We first recall the definition of nondeterministic finite bar automata from [28].

Definition A.2. A *nondeterministic finite bar automaton* (bar NFA) over \mathbb{A} is an NFA A with alphabet $\overline{\mathbb{A}}$. The bar NFA A *literally accepts* the language $L_0(A)$ that A accepts as an NFA with alphabet $\overline{\mathbb{A}}$. The *bar language* accepted by A is defined by

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

The degree $\deg(A)$ of A is the number of names $a \in \mathbb{A}$ that occur in transitions $q \xrightarrow{a} q'$ or $q \xrightarrow{\text{la}} q'$ in A .

Next, we restate Theorem 5.13 from [28].

Theorem A.3 ([28]). *Given an RNNA A_1 , there is a bar NFA A_2 such that $L_\alpha(A_1) = L_\alpha(A_2)$ and the number of states in A_2 is linear in the number of orbits of A_1 and exponential in $\deg(A)$. Furthermore, $\deg(A_2) \leq \deg(A_1) + 1$.*

Proof of Theorem 6.2

Proof. Let A be an RNNA with initial state s_1 and A' an ϵ -free ERNNA with initial state s_2 and let $S = \text{supp}(s_1) \cup \text{supp}(s_2)$. By Theorem A.3, we can transform A to a bar NFA A_1 , such that $L_\alpha(A) = L_\alpha(A_1)$ and the number of states

in A_1 is linear in the number of orbits of A and exponential in $\deg(A)$, and $\deg(A_1) \leq \deg(A) + 1$. By Lemma 5.6, we have $L_\alpha(A', S) = L_\alpha(\text{nd}(A', S))$ and $\text{nd}(A')$ is of degree $\deg(A')$ and has at most $n2^{\deg(A')}$ orbits, where n is the number of orbits in A' ; furthermore, the literally accepted language of $\text{nd}(A')$ is closed under α -equivalence.

We exhibit a NEXPSpace procedure to check that $L_\alpha(A_1)$ is *not* a subset of $L_\alpha(\text{nd}(A'), S)$. The claimed bound then follows by Savitch's theorem. Our algorithm guesses a bar string and an accepting run of A_1 on w for which $\text{nd}(A')$ has no accepting run in context S . To this end, our construction maintains a tuples (q, S', Γ) , where q is a state of A_1 , $S \subseteq S' \subseteq \mathbb{A}$ is a context and Γ is a set of states of $\text{nd}(A')$, with the intuition that all states from $\text{nd}(A')$ are reachable by the bar string that has been read so far. The algorithm initializes q to the initial state of A_1 , S' to S and Γ to $\{s'\}$ where s' is the initial state of $\text{nd}(A')$, and then iterates the following:

- (1) Guess a transition $q \xrightarrow{\alpha} q'$ in A_1 and update q to q' .
- (2) Compute the set Γ' of all states of $\text{nd}(A)$ that are reachable from some state in Γ by an α -transition. If $\alpha = |a$ for some $a \in \mathbb{A}$, then update S' to $S' \cup \{a\}$. If $\alpha = |a$ or $\alpha \in S'$, then put $\Gamma'' = \{q' \in \Gamma \mid f(q') = \top\}$, otherwise put $\Gamma'' = \emptyset$. Update Γ to $\Gamma' \cup \Gamma''$.

The algorithm terminates successfully and reports $L_\alpha(A) \not\subseteq L_\alpha(A')$ if it reaches (q, S', Γ) such that q is a final state of A_1 and there is no state $q' \in \Gamma$ such that $f(q') = \epsilon$ or $f(q') = \top$.

Correctness follows from the proof of Theorem 7.1 in [28] together with the observation that whenever the algorithm reaches (q, S', Γ) such that there is $q' \in \Gamma$ such that $f(q') = \top$, the state q' accepts all bar strings from $\text{free}(S')$, in particular the empty bar string. Since the algorithm has to ensure that there is no accepting run of $\text{nd}(A')$ for the guessed word, any further α -transitions do not remove q' from Γ unless $\alpha \in \mathbb{A} \setminus S'$. This prevents the algorithm from terminating successfully until some free occurrence of a name from $\alpha \in \mathbb{A} \setminus S'$ is read without having previously been bound.

For space usage, we always have $|S'| \leq \deg(A_1) \cup |S|$. Regarding the size of Γ , we first note that for each state in $q' \in \Gamma$, we have $\text{supp}(q') \subseteq \text{supp}(s') \cup \text{names}(A_1)$, where s' is the initial state in $\text{nd}(A')$ and $\text{names}(A_1)$ denotes the set of all names a that occur as label of a transition in A_1 , either in the form $|a$ or a . This is the case since we start with the initial state s' of $\text{nd}(A')$ which has support $\text{supp}(s')$ and since every α -transition in A_1 that the algorithm tracks in $\text{nd}(A')$ adds either nothing (if $\alpha \in \mathbb{A}$), or at most $|a$ (if $\alpha = |a$) to the support of states. Hence we always have $|\text{supp}(q')| \leq \deg(A_1) + \deg(A')$ for all $q' \in \Gamma$. Accounting for the order in which the names from $\text{supp}(q')$ may have been encountered, we hence have that Γ contains at most $(\deg(A_1) + \deg(A'))!m$ states, where m is the number of orbits of $\text{nd}(A')$. Recall that $\text{nd}(A')$ has $n2^{\deg(A')}$ orbits, so we can make do with space

$$\mathcal{O}((\deg(A) + \deg(A') + 1)! \cdot n \cdot 2^{\deg(A')}) \in \mathcal{O}(n \cdot 2^{\deg(A) + 2\deg(A') + 1}),$$

where n is the number of orbits of A' . □

Proof of Corollary 6.3

Proof. Immediate from Theorem 6.2; the exponential dependency on $\deg(\phi)$ inherited from Theorem 6.2 is dominated by the dependency on $|\text{cl}(\phi)|$. \square