On Freeze LTL with Ordered Attributes

Normann Decker and Daniel Thoma

Institute for Software Engineering and Programming Languages
Universität zu Lübeck, Germany
{decker,thoma}@isp.uni-luebeck.de

Abstract

This paper is concerned with Freeze LTL, a temporal logic on data words with registers. In a (multi-attributed) data word each position carries a letter from a finite alphabet and assigns a data value to a fixed, finite set of attributes. The satisfiability problem of Freeze LTL is undecidable if more than one register is available or tuples of data values can be stored and compared arbitrarily. Starting from the decidable one-register fragment we propose an extension that allows for specifying a dependency relation on attributes. This restricts in a flexible way how collections of attribute values can be stored and compared. This conceptual dimension is orthogonal to the number of registers or the available temporal operators. The extension is strict. Admitting arbitrary dependency relations, satisfiability becomes undecidable. Tree-like relations, however, induce a family of decidable fragments escalating the ordinal-indexed hierarchy of fast-growing complexity classes, a recently introduced framework for non-primitive recursive complexities. This results in completeness for the class $\mathbf{F}_{\varepsilon_0}$. We employ nested counter systems and show that they relate to the hierarchy in terms of the nesting depth.

1 Introduction

A central aspect in modern programming languages and software architectures is dynamic and unbounded creation of entities. In particular object oriented designs rely on instantiation of objects on demand and flexible multi-threaded execution. Finite abstractions can hardly reflect these dynamics and therefore infinite models are very valuable for specification and analysis. This motivates us to study the theoretical framework of words over infinite alphabets. It allows for abstracting, e.g., the internal structure and state of particular objects or processes while still being able to capture the architectural design in terms of interaction and relations between dynamically instantiated program parts.

These *data words*, as we consider them here, are finite, non-empty sequences $w = (a_1, \mathbf{d}_1)(a_2, \mathbf{d}_2)\dots(a_n, \mathbf{d}_n)$ where the i-th position carries a letter a_i from a finite alphabet Σ. Additionally, for a fixed, finite set of *attributes* A a *data valuation* $\mathbf{d}_i : A \to \Delta$ assigns to each attribute a *data value* from an infinite domain Δ with equality.

This is a free and extended version of an article published in the proceedings of FoSSaCS 2016. The work was partially supported by EGIDE/DAAD-Procope (FREQS).

Freeze LTL. In formal verification, temporal logics are widely used for formulating behavioural specifications and, regarding data, the concept of storing values in registers for comparison at different points in time is very natural. This paper is therefore concerned with the logic *Freeze LTL* [DLN05] that extends classical Linear-time Temporal Logic (LTL) by registers and was extensively studied during the past decade. Since the satisfiability problem of Freeze LTL is undecidable in general, we specifically consider the decidable fragment $\mathrm{LTL}_1^{\downarrow}$ [DL09] that is restricted to a single register and future-time modalities. More precisely, we propose a generalisation of this fragment and study the consequences in terms of decidability and complexity.

Considering specification and modelling, the ability of comparing *tuples* of data values arbitrarily is a valuable feature. Unfortunately, this generically renders logics on data words undecidable (cf. related work below). We therefore extend Freeze LTL by a mechanism for carefully restricting the collections of values that can be compared in terms of a *dependency relation* on attributes. In general, this does not suffice to regain decidability of the satisfiability problem. Imposing, however, a *hierarchical* dependency structure such that comparison of attribute values is carried out in an ordered fashion, we obtain a strict hierarchy of decidable fragments parameterised by the maximal depth of the attribute hierarchy.

Before we exemplify this concept, let us introduce basic notation. Let Σ be a finite alphabet and $(A, \sqsubseteq)$ a finite set of attributes together with a reflexive and transitive relation $\sqsubseteq \subseteq A \times A$, i.e., a *quasi-ordering*, simply denoted A if $\sqsubseteq$ is understood. We call our logic $\mathrm{LTL}_{qo}^{\downarrow}$ and define its *syntax* according to the grammar

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathrm{X}\,\varphi \mid \varphi\,\mathrm{U}\,\varphi \mid \downarrow^{x}\varphi \mid \uparrow^{x}$$

for letters $a \in \Sigma$ and attributes $x \in A$. We further include common abbreviations such as disjunction, implication or the temporal operators release ($\varphi\,\mathrm{R}\,\psi := \neg(\neg\varphi\,\mathrm{U}\,\neg\psi)$), weak next ($\overline{\mathrm{X}}\,\varphi := \neg\,\mathrm{X}\,\neg\varphi$) and globally ($\mathrm{G}\,\varphi := \mathrm{false}\,\mathrm{R}\,\varphi$). The restriction of $\mathrm{LTL}_{qo}^{\downarrow}$ to a particular, fixed set of attributes $(A, \sqsubseteq)$ is denoted $\mathrm{LTL}_{(A,\sqsubseteq)}^{\downarrow}$ (or simply $\mathrm{LTL}_{A}^{\downarrow}$).

In the following, we explain the idea of our extension by means of an example. The formal semantics is defined in Section 2.

Example 1. *Consider a system with arbitrarily many processes that can lock, unlock and use an arbitrary number of resources. A data word over the alphabet* $\Sigma = \{\mathsf{lock}, \mathsf{unlock}, \mathsf{use}, \mathsf{halt}\}$ *can model its behaviour in terms of an interleaving of individual actions and global signals. The corresponding data valuation can provide specific properties of an action, such as a unique identifier for the involved process and the resource. Let us use attributes* $A = \{\mathsf{pid}, \mathsf{res}\}$ *and interpret data values from* Δ *as IDs. Notice that this way, we do not assume a bound on the number of involved entities.*

Consider now the property that locked resources must not be used by foreign processes and all locks must be released on system halt. To express this, we need to store both the process and resource ID for every lock *action and verify that a* use *involving the same resource also involves the same process. As mentioned earlier, employing a too liberal mechanism to store multiple data values at once breaks the possibility of automatic analysis. In our case, however, we do not need to refer to processes independently. It suffices to consider only resources*

	lock	lock	use	unlock	unlock		lock	lock	use	unlock	halt	unlock
(res)	1	2	2	2	1		1	3	3	3	9	1
(pid)	1	1	1	1	1		1	2	1	1	8	1

Figure 1: The left word satisfies the formula from Example 1 whereas every strict prefix does not. The right word violates the property because at position 3 use holds and the value of res matches the one stored at position 2 but the whole valuation (3,1) differs from (3,2), so the check $\uparrow^{\mathsf{pid}}$ fails. Moreover, halt occurs before (1,1) was observed again in combination with unlock.

individually and formulate that the particular process that locks a resource is the only one using it before unlocking. This one-to-many *correspondence between processes and resources allows us to declare the attribute* pid *to be dependent on the attribute* res *and formulate the property by the formula*

$$\mathrm{G}(\mathsf{lock} \to \downarrow^{\mathsf{pid}}((\mathsf{use} \ \wedge \uparrow^{\mathsf{res}} \to \ \uparrow^{\mathsf{pid}}) \wedge \neg\mathsf{halt})\,\mathrm{U}(\mathsf{unlock} \ \wedge \uparrow^{\mathsf{pid}})).$$

The freeze quantifier $\downarrow^{\mathsf{pid}}$ *stores the current value assigned to* pid *and also implicitly that of all its dependencies,* res *in this case. The* check operator $\uparrow^{x}$*, for an attribute $x \in A$, then verifies at some position that the current values of x and its dependencies coincide with the information that was stored earlier. Also, properties independent of the data can be verified within the same context, e.g.,* $\neg$halt *for preventing a shut down as long as any resource is still locked. See Figure 1 for example words.*

Using this extended storing mechanism, we can select the values of the two attributes ($\downarrow^{\mathsf{pid}}$) and identify and distinguish positions in a data word where both ($\uparrow^{\mathsf{pid}}$), a particular one of them ($\uparrow^{\mathsf{res}}$) or a global signal (e.g., halt*) occurs. In contrast to other decidable fragments of Freeze LTL, we are thus able to store collections of values and can compare individual values across the hierarchy of attributes. This allows for reasoning on complex interaction of entities, also witnessed by the high, yet decidable, complexity of the logic.*

Outline and results. We define the semantics of $\mathrm{LTL}^{\downarrow}_{qo}$ in Section 2 generalising Freeze LTL based on quasi-ordered attribute sets. We show that every fragment $\mathrm{LTL}^{\downarrow}_{A}$ is undecidable unless A has a tree-like structure, formalised as what we call a *tree-quasi-ordering*.

Section 3 is devoted to *nested counter systems (NCS)* and an analysis of their coverability problem. We determine its non-primitive recursive complexity in terms of *fast-growing complexity classes* [Sch13]. These classes $\mathbf{F}_\alpha$ are indexed by ordinal numbers α and characterise complexities by fast-growing functions from the extended Grzegorczyk hierarchy (details are provided in Section 3). We show that with increasing nesting level coverability in NCS exceeds every class $\mathbf{F}_\alpha$ for ordinals $\alpha < \varepsilon_0$. By also providing a matching upper bound, we establish the following.

Theorem 2 (NCS)**.** *The coverability problem in NCS is $\mathbf{F}_{\varepsilon_0}$-complete.*

We consider the fragment $\mathrm{LTL}^{\downarrow}_{tqo}$ in Section 4. It restricts the available dependency relations to tree-quasi-orderings. By reducing the satisfiability problem to NCS coverability, we obtain a precise characterisation of the decidability frontier in $\mathrm{LTL}^{\downarrow}_{qo}$. Moreover, we transfer the lower bounds obtained for NCS to the logic setting. This leads us to a strict hierarchy of decidable fragments of $\mathrm{LTL}^{\downarrow}_{tqo}$ parameterised by the depth of the attribute orderings and a completeness result for $\mathrm{LTL}^{\downarrow}_{tqo}$.

Theorem 3 ($\mathrm{LTL}^{\downarrow}_{qo}$)**.** *The satisfiability problem of*

- $\mathit{LTL}^{\downarrow}_{A}$ *is decidable if and only if* A *is a tree-quasi-ordering.*
- $\mathit{LTL}^{\downarrow}_{tqo}$ *is* $\mathbf{F}_{\varepsilon_0}$*-complete.*

Related work. The *freeze* [Hen90] mechanism was introduced as a natural form of storing and comparing (real-time) data at different positions in time [AH94] and since studied extensively in different contexts, e.g., [Gor96, Fit02, LP05]. In particular linear temporal logic employing the freeze mechanism over domains with only equality, i.e., data words, was considered in [DLN05] and shown highly undecidable (Σ^1_1-hard). Therefore, several decidable fragments were proposed in the literature with complexities ranging from exponential [Laz06] and double-exponential space [DFP13] to non-primitive recursive complexities [DL09]. For the one-register fragment $\mathrm{LTL}^{\downarrow}_1$ that we build on here, an $\mathbf{F}_\omega$ upper bound was given in [Fig12]. Due to its decidability and expressiveness, it is called in [DL09] a "competitor" for the two-variable first-order logic over data words $\mathrm{FO}^2(\sim, <, +1)$ studied in [BDM$^+$11]. There, satisfiability was reduced to and from reachability in Petri nets in double-exponential time and polynomial time, respectively, for which recent results provide an $\mathbf{F}_{\omega^3}$ upper bound [LS15].

Our main ambition is to incorporate means of storing and comparing collections of data values. The apparent extension of storing and comparing even only pairs generically renders logics on data words, even those with essential restrictions, undecidable [BDM$^+$11, KSZ10, DHLT14]. This applies in particular to fragments of $\mathrm{LTL}^{\downarrow}_1$ [DFP13].

The logic *Nested Data LTL (ND-LTL)* studied in [DHLT14] employs a navigation concept based on an *ordered* set of attributes. It inspired our extension of Freeze LTL but in contrast, data values in ND-LTL are not handled explicitly, resulting in incomparable expressiveness and different notions of natural restrictions. While ND-LTL also features a freeze-like mechanism, it does not contain an explicit check operator ($\uparrow$). Instead, data-aware variants of temporal operators such as $\mathrm{U}^{=}$ express constraints (only) for position where the stored value is present. For example, an ND-LTL formula $\mathrm{G}(\mathsf{lock} \to \downarrow^{\mathsf{pid}}(\neg\mathsf{halt}\,\mathrm{U}^{=}\,\mathsf{unlock}))$ (in notation of this paper) requires that for every position satisfying lock there is a future position unlock with the same data value and that $\neg\mathsf{halt}$ holds (at least) on all those position in between that also carry this particular value. In contrast, $\mathrm{G}(\mathsf{lock} \to \downarrow^{\mathsf{pid}}(\neg\mathsf{halt}\,\mathrm{U}(\mathsf{unlock} \wedge \uparrow^{\mathsf{pid}})))$ asserts that at *all* position in between $\neg\mathsf{halt}$ holds. Enforcing such constraints in ND-LTL typically requires an additional level of auxiliary attributes.

The future fragment ND-LTL$^+$ was shown decidable and non-primitive recursive on finite A-attributed data words for tree-(partial-)ordered attribute sets A. However, no upper complexity bounds were provided and the developments

in this paper significantly raise the lower bounds (cf. Section 5). The influence of more general attribute orderings, in particular the precise decidability frontier in that dimension, was not investigated for ND-LTL and its fragments. Instead, the logic was shown undecidable by exploiting the combination of future- and past-time operators. Extending $\mathrm{LTL}_1^{\downarrow}$ with past-time operators is also known to lead to undecidability. ND-LTL$^+$ stays decidable even on infinite words, which is not the case for $\mathrm{LTL}_{tqo}^{\downarrow}$ since satisfiability of $\mathrm{LTL}_1^{\downarrow}$ is already Π_1^0-complete [DL09].

2 Semantics and Undecidability of $\mathrm{LTL}_{qo}^{\downarrow}$

By specifying dependencies between attributes from a set A in terms of a quasi-ordering $\sqsubseteq \subseteq A \times A$ the freeze mechanism can be used to store the values of multiple attributes at once. The essential intuition for our generalised storing mechanism can well be obtained from the special case of a linearly ordered attribute set $[k] = \{1, \ldots, k\}$ with the natural ordering $\leq$ for some natural number $k \in \mathbb{N}$. In fact, many of the technical developments in this paper concerning decidability and complexity are carried out within this setting but for concise presentation we only provide the most general formulation that captures also the undecidable case.

The valuations $\mathbf{d} \in \Delta^{[k]}$ in a $[k]$-attributed data word are essentially sequences (or vectors) $d_1 \ldots d_k$ where the x-th position carries the value $d_x = \mathbf{d}(x)$ of attribute $x \in [k]$. Note that Example 1 matches that setting when renaming res to 1 and pid to 2.

In a formula $\downarrow^x \varphi$ the subformula φ is evaluated in the context of the current value $\mathbf{d}(x)$ of the attribute $x \in A$ and the values $\mathbf{d}(y)$ of all smaller attributes $y \leq x$. Thus, the *prefix* $d_1 \ldots d_x$ of the value sequence at the current position is stored for later comparison. A check operator $\uparrow^y$ then compares the stored values $d_1 \ldots d_x$ to the values $d'_1 \ldots d'_y$ at the current position: the check is successful if the latter sequence is a prefix of the former, i.e. $y \leq x$ and $d_1 \ldots d_y = d'_1 \ldots d'_y$.

For the general setting of arbitrary (quasi-ordered) dependency relations $(A, \sqsubseteq)$, we lift the notion of the prefix of length x to the *restriction* of a valuation $\mathbf{d} : A \to \Delta$ to the *downward-closure* $\mathsf{cl}(x) = \{y \in A \mid y \sqsubseteq x\}$ of x in A. This restriction is defined as $\mathbf{d}|_x : \mathsf{cl}(x) \to \Delta$ with $\mathbf{d}|_x(y) = \mathbf{d}(y)$ for $y \sqsubseteq x$. We denote the set of all such partial data valuations by $\Delta_\perp^A = \{\mathbf{d} : \mathsf{cl}(x) \to \Delta \mid x \in A\}$.

Partial valuations $\mathbf{d}, \mathbf{d}' \in \Delta_\perp^A$ are compared in analogy to sequences: it must be possible to map one onto the other such that ordering is preserved and all values coincide. Formally, we define an equivalence relation $\simeq \subseteq \Delta_\perp^A \times \Delta_\perp^A$ by $\mathbf{d} \simeq \mathbf{d}'$ if and only if there is a bijection $h : \mathsf{dom}(\mathbf{d}) \to \mathsf{dom}(\mathbf{d}')$ such that, for all attributes $x \in \mathsf{dom}(\mathbf{d})$ we have $\mathbf{d}(x) = \mathbf{d}'(h(x))$ and, for all attributes $y \in \mathsf{dom}(\mathbf{d})$, $x \sqsubseteq y \Leftrightarrow h(x) \sqsubseteq h(y)$. Notice that this requires the domains of $\mathbf{d}$ and $\mathbf{d}'$ to be isomorphic. In the definition presented next we therefore allow for restricting the stored valuation arbitrarily before it is matched against the current one. In the linear case this simply means truncating the stored sequence before comparison and intuitively it allows for removing unnecessary information from the context.

Semantics of $\mathrm{LTL}_{qo}^{\downarrow}$. For a non-empty data word $w = (a_1, \mathbf{d}_1) \ldots (a_n, \mathbf{d}_n) \in (\Sigma \times \Delta^A)^+$, an index $1 \leq i \leq n$ in w and a partial data valuation $\mathbf{d} \in \Delta_\perp^A$ the

semantics of $\mathrm{LTL}_A^{\downarrow}$ formulae is defined inductively by

$$
\begin{array}{lll}
(w,i,\mathbf{d}) & \models a_i & \\
(w,i,\mathbf{d}) & \models \neg\varphi & :\Leftrightarrow \quad (w,i,\mathbf{d}) \not\models \varphi \\
(w,i,\mathbf{d}) & \models \varphi \wedge \psi & :\Leftrightarrow \quad (w,i,\mathbf{d}) \models \varphi \text{ and } (w,i,\mathbf{d}) \models \psi \\
(w,i,\mathbf{d}) & \models \mathrm{X}\,\varphi & :\Leftrightarrow \quad i+1 \leq n \text{ and } (w,i+1,\mathbf{d}) \models \varphi \\
(w,i,\mathbf{d}) & \models \varphi\,\mathrm{U}\,\psi & :\Leftrightarrow \quad \exists_{i\leq k\leq n} : (w,k,\mathbf{d}) \models \psi \text{ and } \forall_{i\leq j<k} : (w,j,\mathbf{d}) \models \varphi \\
(w,i,\mathbf{d}) & \models \downarrow^x \varphi & :\Leftrightarrow \quad (w,i,\mathbf{d}_i|_x) \models \varphi \\
(w,i,\mathbf{d}) & \models \uparrow^x & :\Leftrightarrow \quad \exists_{y\in A} : \mathbf{d}|_y \simeq \mathbf{d}_i|_x.
\end{array}
$$

For formulae φ where every check operator $\uparrow^x$ is within the scope of some freeze quantifier $\downarrow^y$ the stored valuation is irrelevant and we write $w \models \varphi$ if $(w,1,\mathbf{d}) \models \varphi$ for any valuation $\mathbf{d}$.

Example 4. *Consider a set of attributes* $\mathsf{A} = \{\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3, \mathsf{y}_1, \mathsf{y}_2\}$ *with* $\mathsf{x}_1 \sqsubseteq \mathsf{x}_2 \sqsubseteq \mathsf{x}_3$ *and* $\mathsf{y}_1 \sqsubseteq \mathsf{y}_2$ *(this is an example for a tree-quasi-ordering, see below), the formula* $\downarrow^{\mathsf{x}_3} \mathrm{X}(\uparrow^{\mathsf{y}_2} \mathrm{U} \uparrow^{\mathsf{x}_3})$ *and a data word* $w = (a_1, \mathbf{d}_1)\ldots(a_n, \mathbf{d}_n)$*. The formula reads as: "Store the current values* d_1, d_2, d_3 *of* $\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3$*, respectively. Move on to the next position. Verify that the stored value* d_1 *appears in* y_1 *and that* d_2 *appears in* y_2 *until the values* d_1, d_2, d_3 *appear again in attributes* $\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3$*, respectively."*

At the first position, the values $d_1 = \mathbf{d}_1(\mathsf{x}_1)$*,* $d_2 = \mathbf{d}_1(\mathsf{x}_2)$ *and* $d_3 = \mathbf{d}_1(\mathsf{x}_3)$ *are stored in terms of the valuation* $\mathbf{d} = \mathbf{d}_1|_{\mathsf{x}_3} : \{\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3\} \to \Delta$ *since* $\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3$ *depend on* x_3*. Assume for the second position* $\mathbf{d}_2(\mathsf{x}_1) \neq \mathbf{d}_1(\mathsf{x}_1) = d_1$*. The formula* $\uparrow^{\mathsf{x}_3}$ *is not satisfied at the second position in the context of* $\mathbf{d}$ *since the only attribute* $p \in A$ *such that* $\mathsf{cl}(p)$ *is isomorphic to* $\{\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3\}$ *is* $p = \mathsf{x}_3$*. Then, however, any order preserving isomorphism needs to map* $\mathsf{x}_1 \in \mathsf{dom}(\mathbf{d})$ *to* $\mathsf{x}_1 \in \mathsf{dom}(\mathbf{d}_2)$ *since* x_1 *is the minimal element in both domains but* $\mathbf{d}(\mathsf{x}_1) \neq \mathbf{d}_2(\mathsf{x}_1)$*. The only way to not violate the formula is hence that* $\mathbf{d}_2(\mathsf{y}_1) = \mathbf{d}_1(\mathsf{x}_1)$ *and* $\mathbf{d}_2(\mathsf{y}_2) = \mathbf{d}_1(\mathsf{x}_2)$*. Then, we can choose* $p = \mathsf{x}_2$ *and have* $\mathbf{d}|_{\mathsf{x}_2} \simeq \mathbf{d}_2|_{\mathsf{y}_2}$ *meaning that* $\uparrow^{\mathsf{y}_2}$ *is satisfied.*

Undecidability. For $\sqsubseteq = \{(x,x) \mid x \in A\}$ (identity) we obtain the special case where only single values can be stored and compared. If $|A| = 1$ we obtain the one-register fragment $\mathrm{LTL}_1^{\downarrow}$. On the other hand, if A contains three attributes x, y, z such that x and y are incomparable and $x \sqsubseteq z \sqsupseteq y$ then storing the value of z also stores the values of x and y. This amounts to storing and comparing the *set* $\{d_x, d_y\} \subset \Delta$ of values assigned to x and y. This is not precisely the same as storing the ordered *tuple* $(d_x, d_y) \in \Delta \times \Delta$ but together with the ability of storing and comparing x and y independently it turns out to be just as contagious considering decidability.

In [BDM+11] it is shown that the satisfiability problem of two-variable first-order logic over data words with two class relations is undecidable by reduction from *Post's correspondence problem*. We can adapt this proof and formulate the necessary conditions for a data word to encode a solution using only the attributes $x \sqsubseteq z \sqsupseteq y$. With ideas from [DFP13] we can also omit using past-time operators. Moreover, this result can be generalised to arbitrary quasi-orderings that contain three attributes $x \sqsubseteq z \sqsupseteq y$.

The absence of such a constellation is formalised by the notion of a *tree-quasi-ordering* defined as a quasi-ordering where the downward-closure of every element is totally ordered. This precisely prohibits elements z that depend on two independent elements x and y. The definition describes in a general way a

hierarchical, tree-like structure. Intuitively, a tree-quasi-ordering is (the reflexive and transitive closure of) a forest of strongly connected components.

Theorem 5 (Undecidability)**.** *Let $(A, \sqsubseteq)$ be a quasi-ordered set of attributes that is* not *a tree-quasi-ordering. Then the satisfiability problem of $LTL_A^{\downarrow}$ is Σ_1^0-complete over A-attributed data words.*

See Appendix A for a complete proof. As will be discussed in Section 4, tree-quasi-orderings represent not only necessary but also sufficient conditions for the logic to be decidable.

3 Nested Counter Systems

Nested counter systems (NCS) are a generalisation of counter systems similar to higher-order multi-counter automata as used in [BB07] and nested Petri nets [LS99]. In this section we establish novel complexity results for their coverability problem. A finite number of counters can equivalently be seen as a multiset $M = \{c_1 : n_1, \ldots, c_m : n_m\}$ over a set of counter names $C = \{c_1, \ldots, c_n\}$. We therefore define NCS in the flavor of [DHLT14] as systems transforming *nested multisets*.

Let $\mathfrak{M}(A)$ denote, for any set A, the set of all *finite multisets* of elements of A. For $k \in \mathbb{N}$ we write $[k]$ to denote the set $\{1, \ldots, k\} \subset \mathbb{N}$ with the natural linear ordering $\leq$. A *k-nested counter system (k-NCS)* is a tuple $\mathcal{N} = (Q, \delta)$ comprised of a finite set Q of *states* and a set $\delta \subseteq \bigcup_{i,j \in [k+1]} (Q^i \times Q^j)$ of *transition rules*. For $0 \leq i \leq k$ the set $\mathcal{C}_i$ of *configurations of level i* is inductively defined by $\mathcal{C}_k = Q$ and $\mathcal{C}_{i-1} = Q \times \mathfrak{M}(\mathcal{C}_i)$. The set of *configurations* of $\mathcal{N}$ is then $\mathcal{C}_\mathcal{N} = \mathcal{C}_0$. Every element of $C_\mathcal{N}$ can, more conveniently, be presented as a term constructed over unary function symbols Q, constants Q and a binary operator $+$ that is associative and commutative. For example, the configuration $(q_0, \{(q_1, \emptyset) : 1, (q_1, \{(q_2, \emptyset) : 2\}) : 2, (q_1, \{(q_2, \emptyset) : 2, (q_3, \{(q_4, \emptyset) : 1\}) : 1\}) : 1\})$ can be represented by the term $q_0(q_1 + q_1(q_2 + q_2) + q_1(q_2 + q_2) + q_1(q_2 + q_2 + q_3(q_4)))$. The operational semantics of $\mathcal{N}$ is now defined in terms of the *transition relation* $\rightarrow \subseteq \mathcal{C}_\mathcal{N} \times \mathcal{C}_\mathcal{N}$ on configurations given by rewrite rules. For $((q_0, \ldots, q_i), (q'_0, \ldots, q'_j)) \in \delta$ and $i, j < k$ we let

$$q_0(X_1 + q_1(\ldots q_i(X_{i+1}) \ldots)) \rightarrow q'_0(X_1 + q'_1(\ldots q'_j(X_{j+1}) \ldots))$$

for any $X_h \in \mathfrak{M}(C_h)$ where $1 \leq h \leq k$ and $X_\ell = \emptyset$ for $i + 2 \leq \ell \leq j + 1$. For example, a rule $((q_0), (q'_0))$ changes the state q_0 in the example configuration above to q'_0. A rule $((q_0, q_1), (q_0, q_1, q'_2))$ adds a state q'_2 non-deterministically as a direct child of one of the states q_1 resulting in one of the three configurations

$$\begin{aligned}
&q_0(q_1(q'_2) + q_1(q_2 + q_2) + q_1(q_2 + q_2) + q_1(q_2 + q_2 + q_3(q_4))),\\
&q_0(q_1 + q_1(q_2 + q_2 + q'_2) + q_1(q_2 + q_2) + q_1(q_2 + q_2 + q_3(q_4))) \text{ and}\\
&q_0(q_1 + q_1(q_2 + q_2) + q_1(q_2 + q_2) + q_1(q_2 + q_2 + q_3(q_4) + q'_2)).
\end{aligned}$$

Moreover, a rule $((q_0, q_1, q_3), (q_0))$ would remove specifically and completely the sub-configuration $q_1(q_2 + q_2 + q_3(q_4))$ since it does not match any other one.

The remaining cases for transitions, where (1) $i < k = j$, (2) $i = k > j$ and (3) $i = k = j$, are defined as expected by rules

$$q_0(X_1 + q_1(\dots q_i(X_{i+1})\dots)) \to q'_0(X_1 + q'_1(\dots q'_{k-1}(X_k + q'_k)\dots)) \quad (1)$$

$$q_0(X_1 + q_1(\dots q_{k-1}(X_k + q_k)\dots)) \to q'_0(X_1 + q'_1(\dots q'_j(X_{j+1})\dots)) \quad (2)$$

$$q_0(X_1 + q_1(\dots q_{k-1}(X_k + q_k)\dots)) \to q'_0(X_1 + q'_1(\dots q'_{k-1}(X_k + q'_k)\dots)) \quad (3)$$

respectively, where for (1) we have $X_{i+2} = \ldots = X_k = \emptyset$. Note that these cases are exhaustive since the nesting depth of terms representing configurations from $\mathcal{C}_\mathcal{N}$ is at most k. As usual we denote by $\to^*$ the reflexive and transitive closure of $\to$. By $\preceq$ we denote the nested multiset ordering, i.e. $M' \preceq M$ iff M' can be obtained by removing elements (or nested multisets) from M. Given two configurations $C, C' \in \mathcal{C}_\mathcal{N}$ the *coverability problem* asks for the existence of a configuration $C'' \in \mathcal{C}_\mathcal{N}$ with $C'' \succeq C'$ and $C \to^* C''$.

To establish our complexity results on NCS we require some notions on ordinal numbers, ordinal recursive functions and respective complexity classes. We represent ordinals using the *Cantor normal form (CNF)*. An ordinal $\alpha < \varepsilon_0$ is represented in CNF as a term $\alpha = \omega^{\alpha_1} + \ldots + \omega^{\alpha_k}$ over the symbol ω and the associative binary operator $+$ where $\alpha > \alpha_1 \geq \ldots \geq \alpha_k$. Furthermore, we denote *limit ordinals* by λ. These are ordinals such that $\alpha + 1 < \lambda$ for every $\alpha < \lambda$. We associate them with a *fundamental sequence* $(\lambda_n)_n$ with supremum λ defined by

$$(\alpha + \omega^{\beta+1})_n := \alpha + \overbrace{\omega^\beta + \ldots + \omega^\beta}^{n} \qquad \text{and} \qquad (\alpha + \omega^{\lambda'})_n := \alpha + \omega^{\lambda'_n}$$

for ordinals β and limit ordinals λ'. Then, ε_0 is the smallest ordinal α such that $\alpha = \omega^\alpha$. We denote the n-th exponentiation of ω as Ω_n, i.e. $\Omega_1 := \omega$ and $\Omega_{n+1} := \omega^{\Omega_n}$. Consequently, $(\Omega_n)_m < \Omega_n$ is the m-th element of the fundamental sequence of Ω_n. Given a monotone and expansive[1] function $h : \mathbb{N} \to \mathbb{N}$, a *Hardy hierarchy* is an ordinal-indexed family of functions $h^\alpha : \mathbb{N} \to \mathbb{N}$ defined by $h^0(n) := n$, $h^{\alpha+1}(n) := h^\alpha(h(n))$ and $h^\lambda(n) := h^{\lambda_n}(n)$. Choosing h as the incrementing function $H(n) := n + 1$, the *fast growing hierarchy* is the family of functions $F_\alpha(n)$ with $F_\alpha(n) := H^{\omega^\alpha}(n)$.

The hierarchy of *fast growing complexity classes* $\mathbf{F}_\alpha$ for ordinals α is defined in terms of the fast-growing functions F_α. We refer the reader to [Sch13] for details and only remark that $\mathbf{F}_{<\omega}$ is the class of primitive recursive problems and problems in $\mathbf{F}_\omega, \mathbf{F}_{\omega^\omega}$ are solvable with resources bound by Ackermannian and Hyper-Ackermannian functions, respectively. The fact most relevant for our classification is that a basic $\mathbf{F}_\alpha$-complete problem is the termination problem of Minsky machines $\mathcal{M}$ where the sum of the counters is bounded by $F_\alpha(|\mathcal{M}|)$ [Sch13].

Upper bound. To obtain an upper bound for the coverability problem in k-NCS we reduce it to that in *priority channel systems (PCS)* [HSS14]. PCS are comprised of a finite control and a fixed number of channels, each storing a string to which a letter can be appended (write) and from which the first letter can be read and removed (read). Every letter carries a priority and can

[1] A function $f : A \to A$ over an ordering $(A, \leq)$ is monotone if $a \leq a' \Rightarrow f(a) \leq f(a')$ and expansive if $a \leq f(a)$ for all $a, a' \in A$.

be lost at any time and any position in a channel if its successor in the channel carries a higher or equal priority. PCS can easily simulate NCS by storing and manipulating an NCS configuration in a channel where a state q at level $i > 0$ in the NCS configuration is encoded by a letter $(q, k-i)$ with priority $k-i$. E.g., the 3-NCS configuration $q_0(q_1 + q_1(q_2 + q_2) + q_1(q_2 + q_2 + q_3(q_4)))$ can be encoded as a channel of the form $(q_1,2)(q_1,2)(q_2,1)(q_2,1)(q_1,2)(q_2,1)(q_2,1)(q_3,1)(q_4,0)$ while q_0 is encoded in the finite control.

Taking the highest priority for the outermost level ensures that the lossiness of PCS corresponds to descending with respect to $\preceq$ for the encoded NCS configuration. Thus the coverability problem in NCS directly translates to that in PCS. The coverability (control-state reachability) problem in PCS with one channel and k priorities lies in the class $\mathbf{F}_{\Omega_{2k}}$ [HSS14] and we thus obtain an upper bound for NCS coverability. See Appendix B.1 for further details.

Proposition 6. *Coverability in k-NCS is in $\mathbf{F}_{\Omega_{2k}}$.*

Lower bound. We can reduce, for any $k > 1$, the halting problem of $H^{(\Omega_k)_l}$-bounded Minsky machines to coverability in k-NCS with the number of states bounded by $l + c$, for some constant c. This yields the following characterisation (recall that $H^{(\Omega_{k+1})_l} = F_{(\Omega_k)_l}$).

Theorem 7. *Coverability in $(k+1)$-NCS is $\mathbf{F}_{\Omega_k}$-hard.*

The idea is to construct a k-NCS $\mathcal{N} = (Q, \delta)$ that can simulate the evaluation of the Hardy function $H^\alpha(n)$ for $\alpha \leq (\Omega_k)_l$ in forward as well as backward direction. It can then compute a *budget* that is used for simulating the Minsky machine. Lower bounds for various models were obtained using this scheme for Turing machines [CS08, HSS14] or Minsky machines [Sch10, RV14].

The following construction uses $k+1$ levels of which one can be eliminated later. We encode the ordinal parameter α of $H^\alpha(n)$ and its argument $n \in \mathbb{N}$ (unary) into a configuration

$$C_{\alpha,n} := \mathsf{main}(\mathsf{s}(M_\alpha) + \mathsf{c}(\overbrace{1 + \ldots + 1}^{n}))$$

using control-states $\mathsf{main}, \mathsf{s}, \mathsf{c}, \omega \in Q$ and configurations M_α defined by $M_0 := \emptyset$ and $M_{\omega^\alpha + \beta} := \omega(M_\alpha) + M_\beta$. For example, an ordinal $\alpha = \omega^\omega + \omega^2 + \omega^2 + 1$ is encoded by

$$M_\alpha = \{(\omega, \{(\omega, \{(\omega, \emptyset) : 1\}) : 1\}) : 1, (\omega, \{(\omega, \emptyset) : 2\}) : 2, (\omega, \emptyset) : 1\}$$

Note that we use shorthands for readability, e.g., ω^ω stands for ω^{ω^1} where 1 is again short for the ordinal ω^0. The construction has to fulfil the following two properties. As NCS do not feature a zero test exact simulation cannot be enforced but errors can be restricted to be "lossy".

Lemma 8. *For all configurations $C_{\alpha,n} \to^* C_{\alpha',n'}$ we have $H^\alpha(n) \geq H^{\alpha'}(n')$.*

The construction will, however, admit at least one run maintaining exact values.

Lemma 9. *If $H^\alpha(n) = H^{\alpha'}(n')$ then there is a run $C_{\alpha,n} \to^* C_{\alpha',n'}$.*

The main challenge is simulating a computation step from a limit ordinal to an element of its fundamental sequence, i.e., from $C_{\alpha+\lambda,n}$ to $C_{\alpha+\lambda_n,n}$ and conversely. Encoding the ordinal parameter using multisets loses the ordering of the addends of the respective CNF terms. Thus, instead of taking the last element of the CNF term we have to select the smallest element, with respect to $\preceq$, of the corresponding multiset. To achieve that, we extend NCS by two operations `cp` and `min`. Given some configuration $C = q_1(q_2(M)) \in \mathcal{C}_{\mathcal{N}}$ the operation $(q_1, q_2)\texttt{cp}(q_1', q_2')$ copies M resulting in $C' = q_1'(q_2'(M_1) + q_2'(M_2))$ with $M_1, M_2 \preceq M$. Conversely, given the configuration C' the operation $(q_1', q_2')\texttt{min}(q_1, q_2)$ results in C with $M \preceq M_1, M_2$.

Both operations can be implemented in a depth first search fashion using the standard NCS operations. Based on them the selection of a smallest element from a multiset can be simulated: all elements are copied (non-deterministically) one by one to an auxiliary set while enforcing a descending order. Applying the `min` operation in every step ensures that we either proceed indeed in descending order or make a "lossy" error. We guess, in each step, whether the smallest element is reached and in that case delete the source multiset. Thereby it is ensured, that the smallest element has been selected or, again, a "lossy" error occurs such that the selected element is now the smallest one. The additional level in the encoding of $C_{\alpha,n}$ enables us to perform this deletion step.

A similar idea to select a smallest ordinal from a multiset is used in [RV14]. However, we need to handle nested structures of variable size correctly whereas in this work the considered ordinals are below Ω_3. They are represented by a multiset of vectors of fixed length where the vectors can be compared and modified directly in order enforce the choice of a minimal one.

We now construct an NCS simulating an $H^\alpha(s)$-bounded Minsky machine $\mathcal{M}$ of size $s := |\mathcal{M}|$ analogously to the constructions in [Sch10, CS08, HSS14]. It starts in a configuration $C_{\alpha,s}$ to evaluate $H^\alpha(s)$. When it reaches $C_{0,n}$ for some $n \leq H^\alpha(s)$ it switches its control state and starts to simulate $\mathcal{M}$ using n as a budget for the sum of the two simulated counters. Zero tests can then be simulated by resets (deleting and creating multisets) causing a "lossy" error in case of an actually non-zero counter. When the simulation of $\mathcal{M}$ reaches a final state the NCS moves the current counter values back to the budged counter and performs a construction similar to the one above but now evaluating $H^\alpha(s)$ *backwards* until reaching $(C_{\alpha,s})'$, the initial configuration with a different control state. If $(C_{\alpha,s})'$ can be reached (or even covered) no "lossy" errors occurred and the Minsky machine $\mathcal{M}$ was thus simulated correctly regarding zero tests. The detailed construction is presented in Appendix B.2.

4 From $\text{LTL}^{\downarrow}_{tqo}$ to NCS and Back

Theorem 5 established a necessary condition for $LTL^{\downarrow}_{A}$ to have a decidable satisfiability problem, namely that A is a tree-quasi-ordering. In the following we show that this is also sufficient. Let $\text{LTL}^{\downarrow}_{tqo}$ denote the fragment of $\text{LTL}^{\downarrow}_{qo}$ restricted to tree-quasi-ordered sets of attributes. The decidability and complexity results for NCS can be transferred to $\text{LTL}^{\downarrow}_{tqo}$ to obtain upper and lower bounds for the satisfiability problem of the logic.

We show a correspondence between the nesting depth in NCS and the depths of the tree-quasi-ordered attribute sets that thus constitutes a semantic hierarchy

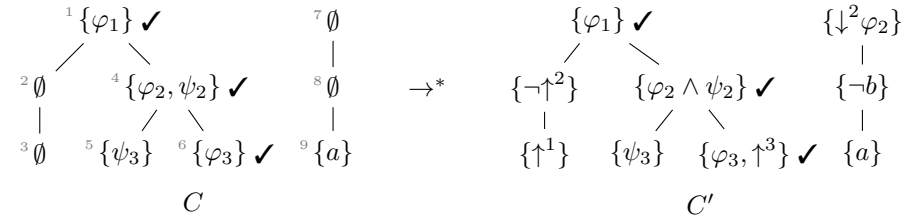

Figure 2: Example of a guarantee forest of depth 3 maintained and modified by the NCS constructed for some $\mathrm{LTL}^{\downarrow}_{[3]}$ formula. Node enumeration (grey) is only for reference.

of logical fragments. We provide the essential ideas in the following and refer the reader to Appendix C and D for the detailed constructions.

The *depth* of a finite tree-quasi-ordering A is the maximal length k of strictly increasing sequences $x_1 \sqsubset x_2 \sqsubset \ldots \sqsubset x_k$ of attributes in A. The first observation is that we can reduce satisfiability of any $\mathrm{LTL}^{\downarrow}_{tqo}$ formula over attributes A to satisfiability of an $\mathrm{LTL}^{\downarrow}_{[k]}$ formula where $[k] = \{1, \ldots, k\}$ is an initial segment of the natural numbers with natural *linear ordering* and k is the depth of A.

Proposition 10 ($\mathrm{LTL}^{\downarrow}_{tqo}$ to $\mathrm{LTL}^{\downarrow}_{[k]}$)**.** *For a tree-quasi-ordered attribute set A of depth k every $\mathrm{LTL}^{\downarrow}_{A}$ formula can be translated to an equisatisfiable $\mathrm{LTL}^{\downarrow}_{[k]}$ formula of exponential size.*

To reduce an arbitrary tree-quasi-ordering A of depth k we first remove maximal strongly connected components (SCC) in the graph of A and replace each of them by a single attribute. This does only affect the semantics of formulae φ if attributes are compared that did not have an isomorphic downward-closure in A. These cases can, however, be handled by additional constraints added to φ. Data words over a thus obtained *partially* ordered attribute set of depth k can now be encoded into words over the *linear* ordering $[k]$ of equal depth k. The idea is to encode a single position into a frame of positions in the fashion of [KSZ10, DHLT14]. That way a single attribute on every level suffices. Any formula can be transformed to operate on these frames instead of single positions at the cost of an at most exponential blow-up.

From $\mathrm{LTL}^{\downarrow}_{[k]}$ to NCS. Given an $\mathrm{LTL}^{\downarrow}_{[k]}$ formula Φ we can now construct a $(k+1)$-NCS $\mathcal{N}$ and two configurations $C_{init}, C_{final} \in \mathcal{C}_{\mathcal{N}}$ such that Φ is satisfiable if and only if C_{final} can be covered from C_{init}.

The idea is to encode sets of *guarantees* into NCS configurations. These guarantees are subformulae of Φ and are guaranteed to be satisfiable. The constructed NCS can instantiate new guarantees and combine existing ones while maintaining the invariant that there is always a data word $w \in (\Sigma \times \Delta^{[k]})^+$ that satisfies all of them. To ensure the invariant, the guarantees are organised in a forest of depth k as depicted in Figure 2.

All formulae φ contained in the same node v of this forest are moreover not only satisfied by the same word w but also with respect to a common valuation $\mathbf{d}_v \in \Delta^{[k]}_{\perp}$, i.e., $(w, 1, \mathbf{d}_v) \models \varphi$. Recall that valuations over linearly ordered

attributes can be seen as sequences. The forest structure now represents the common-prefix relation between these valuations $\mathbf{d}_v$. For two nodes v, v' having a common ancestor at level $i \in [k]$ in the forest, the corresponding valuations $\mathbf{d}_v$, $\mathbf{d}_{v'}$ can be chosen such that they agree on attributes 1 to i. A uniquely marked branch in the forest further represents the valuation $\mathbf{d}_1$ at the first position in w. If a formula φ is contained in the marked node at level i in the forest then $(w, 1, \mathbf{d}_1|_i) \models \varphi$. In that case $(w, 1, \mathbf{d}) \models \downarrow^i \varphi$ holds for any $\mathbf{d} \in \Delta_\perp^{[k]}$ and the formula $\downarrow^i \varphi$ could be added to any of the nodes in the forest without violating the invariant. Similarly, for a marked node v at level i the formulae $\uparrow^i$ can be added to any node in the subtree with root v. Moreover, other atomic formulae, Boolean combinations, and temporal operators can also be added consistently. The NCS $\mathcal{N}$ can perform such modifications on the forest, represented by its configuration, by corresponding transitions.

Example 11. *Consider the two guarantee forests depicted in Figure 2 that are encoded in configurations C and C' of an NCS constructed for some $LTL^{\downarrow}_{[3]}$ formula. The invariant is the existence of a word $w = (a, \mathbf{d})\ldots$ and valuations $\mathbf{d}_v \in \Delta^{[i]}$ such that $(w, 1, \mathbf{d}_v)$ satisfies the formulae in a node v at level i. The forest structure relates these valuations to $\mathbf{d}$ (nodes marked by ✓) and each other. E.g., $(w, 1, \mathbf{d}|_1) \models \varphi_1$, $(w, 1, \mathbf{d}|_2) \models \varphi_2$ and there is $\mathbf{e}$ with $\mathbf{e}|_2 = \mathbf{d}|_2$ and $\mathbf{e}(3) \neq \mathbf{d}(3)$ s.t. $(w, 1, \mathbf{e}) \models \psi_3$. Let $v_1, \ldots, v_9$ be the nodes of the forest (as enumerated in the figure). Several possible operations are exemplified by the transition between C and C'. The formula $\uparrow^3$ can be added to the node v_6 containing the formula φ_3 since that node is checked on level 3. Similarly, there is $\mathbf{d}_3$ for node v_3 such that $\mathbf{d}_3(1) = \mathbf{d}(1)$ and hence $(w, \mathbf{d}_3) \models \uparrow^1$. The formula $\uparrow^2$ cannot be added to the node v_2 since it is not below the checked node on level two. Consequently, the node can contain $\neg\uparrow^2$. Node v_4 on level 2 does already contain φ_2 and ψ_2, meaning they are both satisfied by w and a valuation $\mathbf{d}_4 \in \Delta^{[2]}$. Hence the same holds for their conjunction. Moreover, v_4 is checked and therefore $\mathbf{d}_4 = \mathbf{d}|_2$. This implies that $(w, \mathbf{d}') \models \downarrow^2 \varphi_2$ for any $\mathbf{d}'$ and that the formula can be added to any node in the tree, e.g. v_7.*

Recall that we only need to consider subformulae of Φ and thus remain finite-state for representing nodes. More precisely, the number of states in $\mathcal{N}$ is exponential in the size of Φ since they encode sets of formulae.

A crucial aspect is how the NCS can consistently add formulae of the form $\mathrm{X}\,\varphi$. This needs to be done for all stored guarantees at once but NCS do not have an atomic operation for modifying all states in a configuration. Therefore, the forest is copied recursively, processing each copied node. The NCS $\mathcal{N}$ can choose at any time to stop and remove the remaining nodes. That way it might loose guarantees but maintains the invariant since only processed nodes remain in the configuration. The forest of depth k itself could be maintained by a k-NCS but to implement the copy operation an additional level is needed.

The initial configuration C_{init} consists of a forest without any guarantees. In a setup phase, the NCS can add branches and formulae of the form $\overline{\mathrm{X}}\,\varphi$ since they are all satisfied by any word of length 1. Once the formula Φ is encountered in the current forest the NCS can enter a specific target state q_{final}. A path starting in C_{init} and covering the configuration $C_{final} = q_{final}$ then constitutes a model of Φ and vice versa.

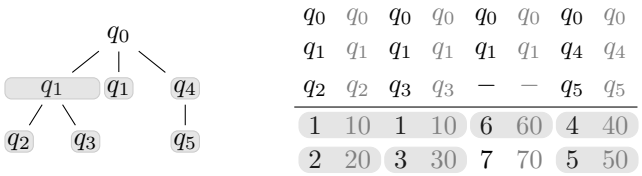

Figure 3: Encoding of a 2-NCS configuration (l.) as [2]-attributed data word (r.). Instead of letters from Σ the encoded tuples of states from Q are displayed at every position.

Theorem 12. *For tree-quasi-ordered attribute sets A with depth k satisfiability of $\mathrm{LTL}^{\downarrow}_{A}$ can be reduced in exponential space to coverability in $(k+1)$-NCS.*

From k-NCS to $\mathrm{LTL}^{\downarrow}_{[k]}$. Let $\mathcal{N} = (Q, \delta)$ be a k-NCS. We are interested in describing witnesses for coverability. It suffices to construct a formula $\Phi_{\mathcal{N}}$ that characterises precisely those words that encode a *lossy* run from a configuration C_{start} to a configuration C_{end}. We call a sequence $C_0 C_1 \dots C_n$ of configurations $C_j \in \mathcal{C}_{\mathcal{N}}$ a lossy run from C_0 to C_n if there is a sequence of intermediate configurations $C'_0 \dots C'_{n-1}$ such that $C_i \succeq C'_i \to C_{i+1}$ for $0 \leq i < n$. Then C_{end} is coverable from C_{start} if and only if there is a lossy run from C_{start} to some $C_n \succeq C_{end}$.

A configuration of a k-NCS is essentially a tree of depth $k+1$ and can be encoded into a $[k]$-attributed data word as a frame of positions, similar as done to prove Proposition 10. We use an alphabet Σ where every letter $a \in \Sigma$ encodes, among other information, a $(k+1)$-tuple of states from Q, i.e., a possible branch in the tree. Then a sequence of such letters represents a set of branches that form a tree. The data valuations represent the information which of the branches share a common prefix. Further, this representation is interlaced: it only uses odd positions. The even position in between are used to represent an exact copy of the structure but with distinct data values. We use appropriate $\mathrm{LTL}^{\downarrow}_{[k]}$ formulae to express this shape. Figure 3 shows an example.

To be able to formulate the effect of transition rules without using past-time operators we encode lossy runs *reversed*. Given that a data word encodes a sequence $C_0 C_1 \dots C_n$ of configurations as above we model the (reversed) control flow of the NCS $\mathcal{N} = (Q, \delta)$ by requiring that every configuration but for the last be annotated by some transition rule $t_j \in \delta$ for $0 \leq j < n$. The labelling is encoded into the letters from Σ and we impose that this transition sequence actually represents the reversal of a lossy run. That is, for every configuration C_j in the sequence (for $0 \leq j < n$) with annotated transition rule t_j there is a configuration C'_{j+1} (not necessarily in the sequence) such that $C_j \xleftarrow{t_j} C'_{j+1} \preceq C_{j+1}$.

For the transition t_j to be executed correctly (up to lossiness) we impose that every branch in C_j must have a corresponding branch in C_{j+1}. Yet, there may be branches in C_{j+1} that have no counterpart in C_j and were thus lost upon executing t_j. Shared data values are now used to establish a link between corresponding branches: for every even position in the frame that encodes C_j

there must be an odd position in the consecutive frame (thus encoding C_{j+1}) with the same data valuation. To ensure that links are unambiguous we require that every data valuation occurs at most twice in the whole word. Depending on the effect of the current transition the letters of linked positions are related accordingly. E.g., for branches not affected at all by t_j the letters are enforced to be equal. This creates a chain of branches along the run that are identified: an odd position links forward to an even one, the consecutive odd position mimics it and links again forward.

Based on these ideas we can construct a formula satisfied precisely by words encoding a lossy run between particular configurations. The size of the formula is polynomial in the size of the NCS $\mathcal{N}$ and can be built by instantiating a set of patterns while iterating over the transitions and states of $\mathcal{N}$, requiring logarithmic space to control the iterations.

Theorem 13. *The coverability problem of k-NCS can be reduced in logarithmic space to* $\mathrm{LTL}^{\downarrow}_{[k]}$ *satisfiability.*

5 Conclusion

By Theorem 12 together with Proposition 6 and Theorem 13 with Theorem 7 we can now characterise the complexity of $\mathrm{LTL}^{\downarrow}_{tqo}$ fragments as follows.

Proposition 14. *Satisfiability of* $\mathrm{LTL}^{\downarrow}_{A}$ *over a tree-quasi-ordered attribute set of depth k is in* $\mathbf{F}_{\Omega_{2(k+1)}}$ *and* $\mathbf{F}_{\Omega_k}$*-hard.*

Together with Theorem 5 this completes the proof of Theorem 3 stating that $\mathrm{LTL}^{\downarrow}_{tqo}$ is the maximal decidable fragment of $\mathrm{LTL}^{\downarrow}_{qo}$ and $\mathbf{F}_{\varepsilon_0}$-complete. The result also shows that the complexity of the logic continues to increase strictly with the depth of the attribute ordering.

The logics ND-$\mathrm{LTL}^{\pm}$ were shown to be decidable by reduction to NCS [DHLT14]. Our results thus provide a first upper bound for their satisfiability problem. Moreover, we derive significantly improved lower bounds by applying the construction to prove Theorem 13 analogously to ND-LTL^{+} and, with reversed encoding, to the past fragment ND-LTL^{-}. A subtle difference is that an additional attribute level is needed in order to express the global data-aware navigation needed to enforce the links between encoded configurations.

Corollary 15. *Satisfiability of ND-*$\mathrm{LTL}^{\pm}$ *with* $k+1$ *levels is in* $\mathbf{F}_{\Omega_{2(k+1)}}$ *and* $\mathbf{F}_{\Omega_k}$*-hard.*

PCS were proposed as a "master problem" for $\mathbf{F}_{\varepsilon_0}$ [HSS14] and indeed our upper complexity bounds for NCS rely on them. However, they are not well suited to prove our hardness results. This is due to PCS being based on sequences and the embedding ordering while NCS are only based on multisets and the subset ordering. In a sense, PCS generalise the concept of channels to multiple levels of nesting, whereas NCS generalise the concept of counters. Hence, we believe NCS are a valuable addition to the list of $\mathbf{F}_{\varepsilon_0}$-complete models. They may serve well to prove lower bounds for formalisms that are like Freeze LTL more closely related to the concept of counting.

References

[AH94] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.

[BB07] Henrik Björklund and Mikołaj Bojańczyk. Shuffle expressions and words with nested data. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 750–761. Springer, 2007.

[BDM+11] Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.

[CS08] Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 205–216. IEEE Computer Society, 2008.

[DFP13] Stéphane Demri, Diego Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 33–42. IEEE Computer Society, 2013.

[DHLT14] Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. Ordered navigation on multi-attributed data words. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 497–511. Springer, 2014.

[DL09] Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

[DLN05] Stéphane Demri, Ranko Lazic, and David Nowak. On the freeze quantifier in constraint LTL: decidability and complexity. In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*, pages 113–121. IEEE Computer Society, 2005.

[Fig12] Diego Figueira. Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1), 2012.

[Fit02] Melvin Fitting. Modal logics between propositional and first-order. *J. Log. Comput.*, 12(6):1017–1026, 2002.

[Gor96] Valentin Goranko. Hierarchies of modal and temporal logics with reference pointers. *Journal of Logic, Language and Information*, 5(1):1–24, 1996.

[Hen90] Thomas A. Henzinger. Half-order modal logic: How to prove real-time properties. In Cynthia Dwork, editor, *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, August 22-24, 1990*, pages 281–296. ACM, 1990.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - (2. ed.)*. Addison-Wesley series in computer science. Addison-Wesley-Longman, 2001.

[HSS14] Christoph Haase, Sylvain Schmitz, and Philippe Schnoebelen. The power of priority channel systems. *Logical Methods in Computer Science*, 10(4), 2014.

[KSZ10] Ahmet Kara, Thomas Schwentick, and Thomas Zeume. Temporal logics on words with multiple data values. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 481–492. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[Laz06] Ranko Lazic. Safely freezing LTL. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 2006.

[LP05] Alexei Lisitsa and Igor Potapov. Temporal logic with predicate lambda-abstraction. In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*, pages 147–155. IEEE Computer Society, 2005.

[LS99] Irina A. Lomazova and Philippe Schnoebelen. Some decidability results for nested petri nets. In Dines Bjørner, Manfred Broy, and Alexandre V. Zamulin, editors, *Perspectives of System Informatics, Third International Andrei Ershov Memorial Conference, PSI'99, Akademgorodok, Novosibirsk, Russia, July 6-9, 1999, Proceedings*, volume 1755 of *Lecture Notes in Computer Science*, pages 208–220. Springer, 1999.

[LS15] Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 56–67. IEEE, 2015.

[RV14] Fernando Rosa-Velardo. Ordinal recursive complexity of unordered data nets. Technical report TR-4-14, Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, 2014.

[Sch10] Philippe Schnoebelen. Revisiting ackermann-hardness for lossy counter machines and reset petri nets. In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 616–628. Springer, 2010.

[Sch13] Sylvain Schmitz. Complexity hierarchies beyond elementary. *CoRR*, abs/1312.5686, 2013.

A Undecidability of $\mathrm{LTL}^{\downarrow}_{qo}$

In this section we provide the technical details for establishing undecidability of $\mathrm{LTL}^{\downarrow}_{qo}$. Recall Theorem 5.

Theorem 5 (Undecidability)**.** *Let $(A, \sqsubseteq)$ be a quasi-ordered set of attributes that is* not *a tree-quasi-ordering. Then the satisfiability problem of $LTL^{\downarrow}_{A}$ is Σ^0_1-complete over A-attributed data words.*

Semi-decidability is obvious when realising that the particular data values in a data word are irrelevant. It suffices to enumerate representatives of the equivalence classes modulo permutations on Δ.

We proceed by first establishing undecidability for a base case with three attributes and generalise it then to an arbitrary number of attributes.

A.1 Base Case

Lemma 16. *For the quasi-ordered set $(A, \sqsubseteq)$ of attributes with $A = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$ where $\mathsf{x} \sqsubseteq \mathsf{z} \sqsupseteq \mathsf{y}$ and x, y are incomparable the satisfiability problem of $LTL^{\downarrow}_{A}$ is undecidable.*

We can reduce the undecidable *Post's correspondence problem (PCP)* (see, e.g., [HMU01]) to satisfiability of $\mathrm{LTL}^{\downarrow}_{A}$. We consider an encoding of the problem used in [BDM+11]. An instance of PCP is given by a finite set $T \subseteq \Sigma^* \times \Sigma^*$ of tiles of the form $t = (u, v)$, $u, v \in \Sigma^*$, over some finite alphabet Σ. The problem is to decide whether there exists a finite sequence $t_1 t_2 \dots t_n = (u_1, v_1)(u_2, v_2) \dots (u_n, v_n)$ such that the "u-part" and the "v-part" coincide, i.e. $u_1 u_2 \dots u_n = v_1 v_2 \dots v_n$.

The idea in [BDM+11] is to encode a sequence of tiles in a word over the alphabet $\Sigma \dot\cup \overline{\Sigma}$, where a distinct copy $\overline{\Sigma} := \{\overline{a} \mid a \in \Sigma\}$ is used to encode the v-part and letters from Σ encode the u-part. For $v = a_1 a_2 \dots \in \Sigma^*$, $a_i \in \Sigma$, we let $\overline{v} = \overline{a}_1 \overline{a}_2 \dots$. A sequence of tiles $(u_1, v_1)(u_2, v_2) \dots$ is then encoded as $\overline{v_1} u_1 \overline{v_2} u_2 \dots$. The switched order of encoding a tile (u_i, v_i) avoids some edge-cases later.

In our setting we require letters to encode additional information and therefor use an alphabet $\Gamma = (\Sigma \dot\cup \overline{\Sigma}) \times 2^{AP}$ where AP is a finite set of atomic propositions.

To show undecidability of the logic it now suffices to construct from an arbitrary instance of PCP T a formula that expresses sufficient and necessary conditions for a data word $(a_1, \mathbf{d}_1)(a_2, \mathbf{d}_2) \dots \in (\Gamma \times \Delta^A)^+$ to encode a solution to the PCP instance in terms of the projection $a'_1 a'_2 \dots \in (\Sigma \dot\cup \overline{\Sigma})^+$ of w where $a_i = (a'_i, m_i)$ for some $m_i \in 2^{AP}$.

In order to translate the conditions given in [BDM+11] in terms of first-order logic formulae, past-time operators would be necessary. To avoid these we use the following two ideas from [DFP13]. Let, for a sequence of tiles $(u_1, v_1)(u_2, v_2) \dots (u_n, v_n) \in T^*$ denote $u := u_1 u_2 \dots u_n$ and $v := v_1 v_2 \dots v_n$. First, we assume AP to contain two propositions e and o that shall mark even and odd positions, respectively, in u and v. Second, we use a variant of PCP that imposes additional restrictions on a valid solution $t_1 t_2 \dots t_n \in T^*$:

- The initial tile is fixed to $t_1 = \hat{t} = (\hat{u}, \hat{v}) \in T$ with $|\hat{u}| > 1$ and $|\hat{v}| > 2$,

- for every strict prefix $t_1 t_2 \dots t_i$, $i < n$, the u-part must be strictly *shorter* than the v part and
- $|u|$ (i.e., the length of the solution) is odd.

The first condition turns the problem into what is called a *modified* PCP in [HMU01] and shown undecidable there by a reduction from the halting problem of Turing machines. It was observed in [DFP13] that this encoding of Turing machines actually guarantees that the length of the u-part is always shorter. As pointed out in [BDM$^+$11], the last condition is not an actual restriction because adding a tile $(\$x, \$y)$ for every tile $(x, y) \in T$ yields a PCP instance that has an odd solution if and only if the original instance had any solution.

We can now adjust the set of conditions from [BDM$^+$11] such that they can be formulated in $\mathrm{LTL}_A^{\downarrow}$ and impose the additional restrictions on a solution.

For easier reading, we use letters $a \in (\Sigma \dot\cup \overline{\Sigma})$ in formulae to denote $\bigvee_{p \in AP}(a, p)$ and propositions $p \in AP$ to denote $\bigvee_{(a,m)\in\Gamma | p \in m}(a, m)$. Also, we use Σ and $\overline{\Sigma}$ to denote the formulae $\bigvee_{a\in\Sigma} a$ and $\bigvee_{\overline{a}\in\overline{\Sigma}} \overline{a}$, respectively. We write $\mathrm{F}\,\varphi$ for $\mathrm{true}\,\mathrm{U}\,\varphi$.

Global structure. Let AP contain a proposition end that we use to mark the end of a sequence of letters that encode a tile. For a tile $t = (b_1 \dots b_m, a_1 \dots a_n) \in T$ where $b_i, a_i \in \Sigma$ let

$$\varphi_t := \left(\bigwedge_{i=1}^{n} \mathrm{X}^{i-1}\,\overline{a_i}\right) \wedge \left(\bigwedge_{i=1}^{m} \mathrm{X}^{n+i-1}\,b_i\right) \wedge \left(\bigwedge_{i=1}^{n+m-1} \mathrm{X}^{i-1}\,\neg\mathsf{end}\right) \wedge \mathrm{X}^{n+m-1}\,\mathsf{end}$$

be the formula expressing that the following positions encode the tile t.

The global structure of a word that correctly encodes a sequence of tiles as described above can now be expressed in terms of the following conditions.

- The word encodes a sequence of tiles from T, starting with $\hat{t}$:

$$\varphi_{\hat{t}} \wedge \mathrm{G}(\mathsf{end} \to \overline{\mathrm{X}} \bigvee_{t\in T} \varphi_t) \tag{4}$$

- The even and odd positions on the substrings u and $\overline{v}$ of a solution are marked correctly with e and o, respectively. For $\hat{t} = (u_1, v_1)$ the formula

$$\mathrm{G}(e \leftrightarrow \neg o) \wedge o \wedge \mathrm{X}^{|v_1|}\,o, \tag{5}$$

  expresses the exclusiveness of markings e and o and specifies the first v-position and the first u-position in the encoding to be odd. Then, the alternation of these markings in the subsequence encoding u is expressed by

$$\begin{aligned} &\mathrm{G}((\Sigma \wedge o) \to \mathrm{X}(\neg\Sigma\,\mathrm{U}(\Sigma \wedge e) \vee \mathrm{G}(\neg\Sigma))) \\ \wedge\,&\mathrm{G}((\Sigma \wedge e) \to \mathrm{X}(\neg\Sigma\,\mathrm{U}(\Sigma \wedge o) \vee \mathrm{G}(\neg\Sigma))) \end{aligned} \tag{6}$$

  and the alternation of markings in the subsequence encoding v by

$$\begin{aligned} &\mathrm{G}((\overline{\Sigma} \wedge o) \to \mathrm{X}(\neg\overline{\Sigma}\,\mathrm{U}(\overline{\Sigma} \wedge e) \vee \mathrm{G}(\neg\overline{\Sigma}))) \\ \wedge\,&\mathrm{G}((\overline{\Sigma} \wedge e) \to \mathrm{X}(\neg\overline{\Sigma}\,\mathrm{U}(\overline{\Sigma} \wedge o) \vee \mathrm{G}(\neg\overline{\Sigma}))) \end{aligned} \tag{7}$$

Let Φ_T be the conjunction of Formulae 4,5, 6 and 7.

	v_1	v_2	v_3	u_1	u_2	v_4	v_5	v_6	u_3	u_4	u_5
AP	o	e	o	o	e, end	e	o	e	o	e, end	o, end
$\Sigma/\overline{\Sigma}$	$\overline{a}$	$\overline{b}$	$\overline{c}$	a	b	$\overline{c}$	$\overline{a}$	$\overline{b}$	c	c	a
x	2	2	4	2	2	4	6	6	4	4	6
y	1	3	3	1	3	5	5	7	3	5	5
z	10	20	30	10	20	40	50	70	30	40	50

Figure 4: Structure of a data word encoding a sequence of tiles $t_1t_2t_3$ with $t_1 = (ab, abc)$, $t_2 = (cc, cab)$, $t_3 = (a, \varepsilon)$.

Chaining u and v. In order to connect the positions belonging to the subword u and the subword v we link consecutive position by a shared data value as depicted in Figure 4.

For the subword encoding u the structure is imposed by the following constraints.

- Each data value occurs at most twice and not in both attributes x and y:

$$\begin{aligned} &\mathrm{G}(\Sigma \to \downarrow^{\mathsf{x}}((\neg \mathrm{F} \uparrow^{\mathsf{y}}) \wedge \neg \mathrm{X}\,\mathrm{F}(\Sigma \wedge \uparrow^{\mathsf{x}} \wedge \mathrm{X}\,\mathrm{F}(\Sigma \wedge \uparrow^{\mathsf{x}}))) \\ \wedge\, &\mathrm{G}(\Sigma \to \downarrow^{\mathsf{y}}((\neg \mathrm{F} \uparrow^{\mathsf{x}}) \wedge \neg \mathrm{X}\,\mathrm{F}(\Sigma \wedge \uparrow^{\mathsf{y}} \wedge \mathrm{X}\,\mathrm{F}(\Sigma \wedge \uparrow^{\mathsf{y}}))) \end{aligned} \tag{8}$$

- At any odd position (except for the last) the data value for attribute x occurs again in x at an even future position and the value of attribute y does never occur again. At any even position, the same holds vice versa for y and x.

$$\mathrm{G}\left(\begin{array}{cccl} & (\Sigma \wedge o \wedge \mathrm{X}\,\mathrm{F}\,\Sigma) & \to (\downarrow^{\mathsf{x}} \mathrm{F}(\uparrow^{\mathsf{x}} \wedge \Sigma \wedge e)) \wedge \neg\downarrow^{\mathsf{y}} \mathrm{X}\,\mathrm{F}(\uparrow^{\mathsf{y}} \wedge\ \Sigma) \\ \wedge & (\Sigma \wedge e) & \to (\downarrow^{\mathsf{y}} \mathrm{F}(\uparrow^{\mathsf{y}} \wedge \Sigma \wedge o)) \wedge \neg\downarrow^{\mathsf{x}} \mathrm{X}\,\mathrm{F}(\uparrow^{\mathsf{x}} \wedge\ \Sigma) \end{array}\right) \tag{9}$$

The same restrictions can be formulated analogously for the subword that encodes v.

$$\begin{aligned} &\mathrm{G}(\overline{\Sigma} \to \downarrow^{\mathsf{x}}((\neg \mathrm{F} \uparrow^{\mathsf{y}}) \wedge \neg \mathrm{X}\,\mathrm{F}(\overline{\Sigma} \wedge \uparrow^{\mathsf{x}} \wedge \mathrm{X}\,\mathrm{F}(\overline{\Sigma} \wedge \uparrow^{\mathsf{x}}))) \\ \wedge\, &\mathrm{G}(\overline{\Sigma} \to \downarrow^{\mathsf{y}}((\neg \mathrm{F} \uparrow^{\mathsf{x}}) \wedge \neg \mathrm{X}\,\mathrm{F}(\overline{\Sigma} \wedge \uparrow^{\mathsf{y}} \wedge \mathrm{X}\,\mathrm{F}(\overline{\Sigma} \wedge \uparrow^{\mathsf{y}}))) \\ \wedge\, &\mathrm{G}\left(\begin{array}{cccl} & (\overline{\Sigma} \wedge o \wedge \mathrm{X}\,\mathrm{F}\,\overline{\Sigma}) & \to (\downarrow^{\mathsf{x}} \mathrm{F}(\uparrow^{\mathsf{x}} \wedge \overline{\Sigma} \wedge e)) \wedge \neg\downarrow^{\mathsf{y}} \mathrm{X}\,\mathrm{F}(\uparrow^{\mathsf{y}} \wedge\ \overline{\Sigma}) \\ \wedge & (\overline{\Sigma} \wedge e) & \to (\downarrow^{\mathsf{y}} \mathrm{F}(\uparrow^{\mathsf{y}} \wedge \overline{\Sigma} \wedge o)) \wedge \neg\downarrow^{\mathsf{x}} \mathrm{X}\,\mathrm{F}(\uparrow^{\mathsf{x}} \wedge\ \overline{\Sigma}) \end{array}\right) \end{aligned} \tag{10}$$

Let Φ_{chain} denote the conjunction of the formulae from Equations 8, 9 and 10.

To formalise the guarantee on the structure that we obtain from these constraints let

$$w = v_1 \dots v_{m_1} u_1 \dots u_{n_1} v_{m_1+1} \dots v_{m_1+m_2} u_{n_1+1} \dots u_{n_1+n_2} \dots$$

for $v_i, u_i \in \Gamma \times \Delta^A$ be the encoding of a sequence of tiles $t_1t_2\dots \in T^+$ with $w \models \Phi_T \wedge \Phi_{\mathsf{chain}}$. Further let $u = u_1u_2\dots = (a_1, \mathbf{d}_1)(a_2, \mathbf{d}_2)\dots$ and $v = v_1v_2\dots = (b_1, \mathbf{e}_1)(b_2, \mathbf{e}_2)\dots$ be the subwords of w encoding the u-part and the v-part of the tile sequence, respectively.

Lemma 17. *Let $i < k$ be a position in the subword u of w with length $|u| = k$.*

1. *If i is* odd *then* $\mathbf{d}_i(\mathsf{x}) = \mathbf{d}_{i+1}(\mathsf{x})$.
2. *If i is* even *then* $\mathbf{d}_i(\mathsf{y}) = \mathbf{d}_{i+1}(\mathsf{y})$.
3. *For all positions i, j on u we have* $(\mathbf{d}_i(\mathsf{x}) = \mathbf{d}_j(\mathsf{x}) \wedge \mathbf{d}_i(\mathsf{y}) = \mathbf{d}_j(\mathsf{y})) \Rightarrow i = j$.

The same holds analogously for v.

Proof. Given the formula Φ_T it is easy to see that the even and odd positions in the subwords u and v are correctly marked by the respective propositions. We present the proof only for u since it is identical for v.

1.+2.) Assume u has length k. We proceed by induction on the positions i, backward from $k-1$ down to 1.

Base case $i = k-1$. The length k of u needs to be odd, otherwise there is no even future position and Formula 9 is violated. Hence $i = k-1$ is even and Formula 9 ensures that the value $\mathbf{d}_{k-1}(\mathsf{y})$ is repeated in attribute y, leaving $\mathbf{d}_k$ as only choice.

Induction. Assume for $i+1 < k-1$ the statement holds. Assume i is odd. Since the position $i+1$ exists, by Formula 9 there is a position $j > i$ such that $\mathbf{d}_i(\mathsf{x}) = \mathbf{d}_j(\mathsf{x})$. Now for every even $j > i+2$ the induction hypothesis holds for $j-1$, being odd. I.e. $\mathbf{d}_{j-1}(\mathsf{x}) = \mathbf{d}_j(\mathsf{x})$. Since the value $\mathbf{d}_j(\mathsf{x})$ can only occur at most twice for x in u (Formula 8), we have $\mathbf{d}_i(\mathsf{x}) \neq \mathbf{d}_j(\mathsf{x})$, leaving $j = i+1$ as only choice.

Assume i is even. Formula 9 requires that $\mathbf{d}_i(\mathsf{y}) = \mathbf{d}_j(\mathsf{y})$ for some odd $j > i$. Again, for any odd $j > i+1$ the induction hypothesis holds for $j-1 \geq i+1$, being even. We have $\mathbf{d}_{j-1}(\mathsf{y}) = \mathbf{d}_j(\mathsf{y})$ and thus $\mathbf{d}_i(\mathsf{y}) \neq \mathbf{d}_j(\mathsf{y})$ for any odd $j > i+1$. Therefore only position $j = i+1$ remains to carry the same value for y.

3.) Let $\mathbf{d}_i(\mathsf{x}) = \mathbf{d}_j(\mathsf{x})$ and $\mathbf{d}_i(\mathsf{y}) = \mathbf{d}_j(\mathsf{y})$. Assume i is odd. Then, $\mathbf{d}_i(\mathsf{x}) = \mathbf{d}_{i+1}(\mathsf{x})$ (see above) and, by Formula 9, $\forall_{i'>i+1} : \mathbf{d}_{i'}(\mathsf{x}) \neq \mathbf{d}_i(\mathsf{x})$. Hence $i \leq j \leq i+1$. Moreover, $\forall_{i'>i} : \mathbf{d}_{i'}(\mathsf{y}) \neq \mathbf{d}_i(\mathsf{y})$ and thus $j = i$. For even i the argument holds analogously. □

Synchronising u and v. Now that the encoding of u and v is set up, we enforce that

1. every position in v matches a unique position in u,
2. the first v position matches the first u position and
3. for any two consecutive positions in v the corresponding matching positions in u are also consecutive. Finally,
4. the last position in v matches the last position in u.

This is accomplished by the formula Φ_{sync} being the conjunction of the three formulae

$$(\downarrow^{\mathsf{z}} \mathrm{X}^{|\hat{v}|} \uparrow^{\mathsf{z}}), \tag{11}$$

$$\bigwedge_{\overline{a} \in \overline{\Sigma}} \mathrm{G}(\overline{a} \rightarrow \downarrow^{\mathsf{z}}(\mathrm{X}\,\mathrm{F}(a \ \wedge \uparrow^{\mathsf{z}}))), \tag{12}$$

$$\mathrm{G}\left((\overline{\Sigma} \wedge \neg\,\mathrm{X}\,\mathrm{F}\,\overline{\Sigma}) \rightarrow \downarrow^{z}(\mathrm{X}\,\mathrm{F}(\uparrow^{z} \wedge \neg\,\mathrm{X}\,\mathrm{true}))\right), \tag{13}$$

where $\hat{v}$ is part of the fixed initial tile $\hat{t} = (\hat{u}, \hat{v})$. The formula specifies that

- each set $\{\mathbf{e}_i(\mathsf{x}), \mathbf{e}_i(\mathsf{y})\}$ of values occurring at some position i in v occurs again at a position in u with the same (encoded),
- the data values at the first positions in v and u coincide and
- the data values at the last positions in v and u coincide.

For easier reading let $\overline{(a, \mathbf{d})} := (\overline{a}, \mathbf{d}) \in \overline{\Sigma} \times \Delta$ for $(a, \mathbf{d}) \in \Sigma \times \Delta$. The essential observation is now the following.

Lemma 18. *Let $w \models \Phi_T \wedge \Phi_{\mathsf{chain}} \wedge \Phi_{\mathsf{sync}}$ be a data word and w' its projection to the alphabet $(\Sigma \cup \overline{\Sigma}) \times \Delta$. Let $u = u_1 \dots u_{|u|}$ and $v = v_1 \dots v_{|v|}$ be the maximal subwords of w' over $\Sigma \times \Delta$ and $\overline{\Sigma} \times \Delta$, respectively. Then, for all $0 < i \leq |v|$ we have $i \leq |u|$ and $v_i = \overline{u_i}$.*

That is, the i-th position in the v-part of w corresponds to the i-th position in the u-part and thus v encodes a prefix of the u-part. Since the last position in v must corresponds to the last position in u (Equation 13), v and u must encode the *same* sequence of letters and w therefore a solution to the PCP T. On the other hand, given a solution for T, we can easily encode it according to the scheme depicted in Figure 4 where corresponding positions in u and v can be linked appropriately. This encoding satisfies (by construction) all the constraints imposed by the formulae above. Hence, by proving Lemma 18 we complete the proof of Lemma 16.

Lemma 18. Let $u = (a_1, \mathbf{d}_1) \dots (a_\ell, \mathbf{d}_\ell)$ and $v = (b_1, \mathbf{e}_1) \dots (b_k, \mathbf{e}_k)$. We proceed by induction on i.

Base case ($i = 1$). For the initial tile $\hat{t} = (\hat{u}, \hat{v})$ we assumed that $|\hat{v}| \geq 1$ and $|\hat{u}| \geq 1$ so 1 is a position in u as well as in v. Equation 11 requires that $v_1 = \overline{u_1}$.

Induction ($i > 1$). Assume $i \leq |v|$ is a position in v. Thus, all $0 < j < i$ are positions in v and by the induction hypothesis (IH) also positions in u with $v_j = u_j$.

Assume i is odd. We have

$$\mathbf{d}_i(\mathsf{y}) \overset{\text{Lem. 17}}{=} \mathbf{d}_{i-1}(\mathsf{y}) \overset{\text{IH}}{=} \mathbf{e}_{i-1}(\mathsf{y}) \overset{\text{Lem. 17}}{=} \mathbf{e}_i(\mathsf{y})$$

Let $v_i = (b_i, \mathbf{e}_i) = (\overline{a}, \mathbf{e}_i)$ for some $a \in \Sigma$. By Equation 12 thre must be a position $u_j = (a_j, \mathbf{d}_j)$ in u where $a_j = a$ and $\mathbf{d}_j = \mathbf{e}_i$.

By Φ_{chain}, there can be at most two positions j with $\mathbf{d}_j(\mathsf{y}) = \mathbf{e}_i(\mathsf{y})$ and we have already $\mathbf{d}_{i-1}(\mathsf{y}) = \mathbf{d}_i(\mathsf{y}) = \mathbf{e}_i(\mathsf{y})$. Hence, $j \in \{i, i-1\}$. That is, either $\overline{u_i} = v_i$ or $\overline{u_{i-1}} = v_i$. The latter can be excluded since

$$\overline{u_{i-1}} \overset{\text{IH}}{=} v_{i-1} \overset{\text{Lem. 17}}{=} v_i.$$

If i is assumed to be even the same arguments apply when exchanging attribute y by attribute x. □

Remark 19. *Notice that we do not rely on using an until operator. Instead, we can replace it by a bounded version $\mathrm{U}^{\leq k}$ that in turn can be replaced by a finite unfolding only using nested* X *operators. The relevant range can be bound by the length of the tiles in T as*

$$k \geq 2 \cdot \max\{|r| \mid \exists_s (r, s) \in T \vee (s, r) \in T\}.$$

Thus, we take k to be at least as large as the longest consecutive pair $\overline{v_i}u_i$ or $u_i\overline{v_{i+1}}$ in the encoding of a solution could possibly be. This is a bound on the distance between two position in the encoding that are consecutive in u or v. Hence, only the operators X *and* F *are essential.*

A.2 General Case

We can now complete the proof of Theorem 5.

Theorem 5. Lemma 16 established undecidability for the essential case of a non-tree-quasi-ordering. It remains to conclude that this results generalises to arbitrary non-tree-quasi-orderings.

Let $(A, \sqsubseteq)$ be the quasi-ordering defined in Lemma 16. First of all, $(A, \sqsubseteq)$ is not a tree-quasi-ordering since the downward-closure $\mathsf{cl}(\mathsf{z})$ of z is not quasi-linear (total). Moreover, every non-tree-quasi-ordering $(A', \sqsubseteq')$ has a subset that is isomorphic to A: By definition A' must contain an element z' of which the downward-closure is not quasi-linear and must hence contain two incomparable elements $x' \sqsubseteq z'$ and $y' \sqsubseteq z'$. Hence from now on we assume w.l.o.g. that $A \subseteq A'$ by identifying $\mathsf{x}, \mathsf{y}, \mathsf{z}$ with x', y', z', respectively.

We now show that the formula Φ constructed to prove Lemma 16 is satisfiable over A-attributed data words if and only if it is satisfiable when being interpreted over A'-attributed data words.

($\Rightarrow$) Consider an A-attributed data word w satisfying Φ. Choose a data value $e \in \Delta$ that does not occur in w and extend w to an A'-attributed data word w' by assigning e to every attribute $p \in A' \setminus A$ at every position in w'. This does not change the satisfaction relation because Φ still only uses attributes from A and the evaluation of formulae $\uparrow^r$ for $r \in A$ is not affected: For $w = (a_1, \mathbf{d}_1) \dots (a_n, \mathbf{d}_n)$, $w' = (a_1, \mathbf{d}'_1) \dots (a_n, \mathbf{d}'_n)$, $0 < i \leq j \leq n$, $r' \in A$ we have

$$(w, j, \mathbf{d}_i|_r) \models \uparrow^{r'} \quad \Leftrightarrow \quad (w', j, \mathbf{d}'_i|_r) \models \uparrow^{r'}.$$

i) Let $r = r' \in \{\mathsf{x}, \mathsf{y}\}$. Notice that $(w, j, \mathbf{d}_i|_r) \models \uparrow^r$ iff $\mathbf{d}_i(r) = \mathbf{d}_j(r)$. Then $\mathbf{d}_i(r) = \mathbf{d}_j(r)$ implies that $\exists_{p \in A'} : \mathbf{d}'_i|_r|_p \simeq \mathbf{d}'_j|_r$ since for $p = r$ the restrictions are isomorphic as all other attributes in $\mathsf{cl}(r)$ are always mapped to e. Conversely, if $\exists_{p \in A'} : \mathbf{d}'_i|_r|_p \simeq \mathbf{d}'_j|_r$ then it can only be the case for some p such that $\mathsf{cl}(p) = \mathsf{cl}(r)$. Since $\mathbf{d}_j(q) = \mathbf{d}_i(q) = e \neq \mathbf{d}_i(r)$ for all $q \in \mathsf{cl}(r) \setminus \{r\}$ the valuations can only be isomorphic if $\mathbf{d}_j(r) = \mathbf{d}_i(r)$.

ii) Let $r = r' = \mathsf{z}$. We have that $(w, j, \mathbf{d}_i|_\mathsf{z}) \models \uparrow^\mathsf{z}$ iff $\mathbf{d}_i(\mathsf{z}) = \mathbf{d}_j(\mathsf{z})$ and $\{\mathbf{d}_i(\mathsf{x}), \mathbf{d}_i(\mathsf{y})\} = \{\mathbf{d}_j(\mathsf{x}), \mathbf{d}_j(\mathsf{y})\}$. In our case, the models of Φ only admit disjoint values for attributes x and y (cf. Formulae 8 and 10 in the proof of Lemma 16). Thus, $\mathbf{d}_i(\mathsf{x}) = \mathbf{d}_j(\mathsf{x})$ and $\mathbf{d}_i(\mathsf{y}) = \mathbf{d}_j(\mathsf{y})$. This further implies $\exists_{p \in A'} : \mathbf{d}'_i|_\mathsf{z}|_p \simeq \mathbf{d}'_j|_\mathsf{z}$ witnessed by choosing $p = \mathsf{z}$ since all other attributes are evaluated to e by $\mathbf{d}'_i$ and $\mathbf{d}'_j$. Moreover, the opposite direction holds for the same reason.

iii) Let $r' = \mathsf{x}$ and $r = \mathsf{y}$ or vice versa. Again $(w, j, \mathbf{d}_i|_r) \models \uparrow^{r'}$ iff $\mathbf{d}_i(r) = \mathbf{d}_j(r')$, which however, cannot be true due to x and y being assigned disjoint sets of values in every model. On the other hand, assume there is $p \in A'$ s.t. $\mathbf{d}'_i|_r|_p \simeq \mathbf{d}'_j|_{r'}$. Clearly the witnessing isomorphism must map r to r' since

they are not assigned the value e. Then, however, $\mathbf{d}_i(r) = \mathbf{d}_j(r')$ which violates Φ.

The remaining cases do not occur in Φ (and would evaluate to false anyway). We conclude that if w is a model for Φ then w' is as well.

($\Leftarrow$) Consider an A'-attributed data word w' satisfying φ and let $\overline{\Delta_{w'}} \subseteq \Delta$ be an enumerable set of data values not occurring in w'. Let $f : \Delta_{\perp}^{A'}/{\simeq} \hookrightarrow \overline{\Delta_{w'}}$ be an injection from the $\simeq$-equivalence classes of data valuations to data values uniquely representing them. We can then construct an A-attributed model w for φ from w' by erasing all attributes except for $\mathsf{x}, \mathsf{y}, \mathsf{z}$ and let $\mathbf{d}_i(p) := f([\mathbf{d}'_i|_p]_\simeq)$ for $p \in A$ where $[\mathbf{d}]_\simeq$ denotes the $\simeq$-equivalence class of a data valuation $\mathbf{d}$. Intuitively, at any position in w', we just collapse the structure of data values to a single one representing its equivalence class. By similar arguments as above, we can again show that

$$(w, j, \mathbf{d}_i|_r) \models \uparrow^{r'} \quad \Leftrightarrow \quad (w', j, \mathbf{d}'_i|_r) \models \uparrow^{r'}.$$

i) For $r = r'$ we have that $\exists_{p \in A'} : \mathbf{d}'_i|_r|_p \simeq \mathbf{d}'_j|_r$ iff $\mathbf{d}'_i|_r \simeq \mathbf{d}'_j|'_r$ iff $[\mathbf{d}'_i|_r]_\simeq = [\mathbf{d}'_j|_r]_\simeq$ iff $\mathbf{d}_i(r) = \mathbf{d}_j(r)$.

ii) For $r = \mathsf{x}$ and $r' = \mathsf{y}$ or vice versa $\exists_{p \in A'} : \mathbf{d}'_i|_r|_p \simeq \mathbf{d}'_j|_r$ cannot be true in a model for φ and this being false implies equally $\mathbf{d}_i(r) \neq \mathbf{d}_j(r')$.

Again, other cases do not apply. □

B Nested Counter Systems

B.1 Upper Bound for NCS Coverability

Recall Proposition 6.

Proposition 6. *Coverability in k-NCS is in* $\mathbf{F}_{\Omega_{2k}}$.

The statement can be proven by a direct reduction to coverability (equivalently, control-state reachability) in priority channel systems (PCS) that we briefly recall from [HSS14] in the following.

Priority Channel Systems. PCS can be defined over so called *generalised priority alphabets*. Given a *priority level* $d \in \mathbb{N}$ and a well-quasi-ordering $(\Gamma, \leq_\Gamma)$ a generalised priority alphabet is a set $\Sigma_{d,\Gamma} := \{(a, w) \mid 0 \leq w \leq d, w \in \Gamma\}$. Then, a PCS is a tuple $S = (\Sigma_{d,\Gamma}, \mathsf{Ch}, Q, \Delta)$, where Ch is a finite set of *channel names*, Q is a finite set of *control states* and $\Delta \subseteq Q \times \mathsf{Ch} \times \{!, ?\} \times \Sigma_{d,\Gamma} \times Q$ is a set of *transition rules*. The *semantics* of PCS is defined as a transition system over configurations $\mathrm{Conf}_S := Q \times (\Sigma^*_{d,\Gamma})^{\mathsf{Ch}}$ consisting of a control state and a function assigning to every channel a sequence of messages (letters from the generalised priority alphabet) it contains. A PCS can either execute one of its transition rules or an internal "lossy" operation called a *superseding step*. A (writing) transition rule of the form $(q, c, !, (a, w), q')$ is performed by changing the current control state q to q' and appending the letter (a, w) to the content of channel c. A (reading) transition rule of the form $(q, c, ?, (a, w), q')$ is performed by changing the current control state q to q' and removing the letter (a, w) from the first position of channel c. An internal superseding step is performed by overriding a letter by a subsequent letter with higher or equal priority, i.e. the channel content $(a_1, w_1) \dots (a_i, w_i)(a_{i+1}, w_{i+1}) \dots (a_k, w_k)$ with $w_i \leq w_{i+1}$ can be replaced by $(a_1, w_1) \dots (a_{i-1}, w_{i-1})(a_{i+1}, w_{i+1}) \dots (a_k, w_k)$.

Encoding. The semantics of NCS is defined in terms of rewriting rules on configurations represented as terms. A PCS can simulate this semantics by keeping the top-level state in its finite control and storing the term representation of the nested multiset (with an additional marker at the end) in a single channel. The rewriting rules of the semantics can be applied by alternately reading from and writing to that channel. To simulate one step of the NCS the PCS guesses the rule that should be applied and then starts to loop one time through the channel. Going through the channel, the PCS guesses the positions where the rule matches and at the same time writes the corresponding messages back to the channel. During one iteration, the PCS has to keep track of hte guessed transition and up to which nesting level it already has been applied.

As PCS are lossy the only objective is to ensure that lossiness with respect to the PCS semantics corresponds to descending with respect to $\preceq$ for the encoded NCS configurations. This can be easily ensured by encoding the nesting structure of an NCS configuration using priorities where the highest priority corresponds to the outermost nesting level. E.g., the 3-NCS configuration

$$q_0(q_1 + q_1(q_2 + q_2) + q_1(q_2 + q_2 + q_3(q_4)))$$

can be encoded as

$$(q_1, 2)(q_1, 2)(q_2, 1)(q_2, 1)(q_1, 2)(q_2, 1)(q_2, 1)(q_3, 1)(q_4, 0)(\$, 2)$$

while q_0 is encoded using the control state and \$ marks the end of the encoding. A superseding step then always corresponds to removing an element from an innermost multiset.

Complexity. The above encoding gives us a PCS with k priorities (maximal priority $d = k - 1$), one channel, a number of control states s polynomial in k and the number of states and transitions of the NCS and a size of the alphabet Γ linear in the number of the states of the NCS.

The upper bound on the complexity of PCS control-state reachability is proved in [HSS14] by providing a bound on the so-called length function $L_{\Sigma^*_{p,\Gamma}}$. It measures the length controlled bad sequences in the well-quasi-ordered set of channel configurations $\Sigma^*_{p,\Gamma}$. Specifically, the proof of [HSS14, Corollary 4.2] provides a bound on the length function for the well-quasi-ordering $\Sigma^*_{k-1,\Gamma}$ on PCS configurations with a single channel: $L_{\Sigma^*_{k-1,\Gamma}}(n) \leq H^{(\Omega_{2k+1})_{|\Gamma|}}(n)$. Taking into account the control states, following [HSS14, Section 4.3], this yields a bound $H^{(\Omega_{2k+1})_{|\Gamma|}\cdot s}(n)$ on the length function for configurations of the PCS that we construct from an NCS as outlined above. The coverability problem of k-NCS is hence contained in $\mathbf{F}_{(\Omega_{2k})_{|\Gamma|}+1}$. For simplicity, we use the larger class $\mathbf{F}_{\Omega_{2k}}$ in the statement of Proposition 6.

B.2 Lower Bound for NCS Coverability

In this section we give the detailed constructions proving Theorem 7. As we have discussed above, we need a construction fulfilling Lemma 8 and Lemma 9.

Auxilary operations. To this end, we extend the NCS with two auxiliary operations $\mathtt{cp}$ and $\mathtt{min}$. The semantics of the operation $(q_1, \ldots, q_l)\mathtt{cp}(q'_1, \ldots, q'_l)$ can be given by the rewriting rule

$$\begin{aligned} & q_1(X_1 + q_2(X_2 + \ldots q_l(X_l)) + q'_2(X'_2 + \ldots q'_{l-1}(X'_{l-1}))) \\ \rightarrow & q'_1(X_1 + q_2(X_2 + \ldots q_{l-1}(X_{l-1} + q_l(X''))) \\ & + q'_2(X'_2 + \ldots q'_{l-1}(X'_{l-1} + q'_l(X')))) \end{aligned}$$

where $X' \preceq X_l$ and $X'' \preceq X_l$. The operation copies the multiset marked by $q_2, \ldots, q_l$ "lossily" to a multiset marked by $q'_2, \ldots, q'_l$. Consider the configuration $q_1(q_2(q_3 + q_3) + q_4(q_5))$ and the copy rule $(q_1, q_2)\mathtt{cp}(q_6, q_7)$. The rule would select the part $q_2(q_3 + q_3)$ and copy it, changing its label to q_7 and the control state to q_6 resulting in $q_6(q_2(q_3 + q_3) + q_4(q_5) + q_7(q_3 + q_3))$.

The operation $(q_1, \ldots, q_l)\mathtt{min}(q'_1, \ldots, q'_l)$ can be seen as the inverse operation. Its semantics can be given by

$$\begin{aligned} & q_1(X_1 + q_2(X_2 + \ldots q_l(X_l)) + q'_2(X'_2 + \ldots q'_l(X'_l))) \\ \rightarrow & q'_1(X_1 + q_2(X_2 + \ldots q_{l-1}(X_{l-1})) + q'_2(X'_2 + \ldots q'_{l-1}(X'_{l-1} + q'_l(X')))) \end{aligned}$$

where $X' \preceq X_l$ and $X' \preceq X'_l$. It deletes the multiset marked by $q_2, \ldots, q_l$ and replaces the multiset marked by $q'_2, \ldots, q'_l$ with the minimum of both (or a smaller multiset). Consider the configuration $q_1(q_2(q_3 + q_4) + q_5(q_3 + q_6))$ and the rule $(q_1, q_2)\mathtt{min}(q_7, q_5)$. The rule would remove the part $q_2(q_3 + q_4)$, replace $q_5(q_3 + q_6)$ by the minimum and change the control state to q_7 resulting in $q_7(q_5(q_3))$. Both operations can be implemented using standard NCS transition rules and do thus not extend the computational power of NCS.

Implementing $\mathtt{cp}$. A copy rule $t = (q_1, \ldots, q_l)\mathtt{cp}(q'_1, \ldots, q'_l)$ can be implemented as follows: (The variables r_i and index q are universially quantified over all states)

$$\begin{gathered} (q_1, \ldots, q_l)\delta(\mathsf{cpi}_{t,q_l}, q_2, \ldots, q_{l-1}, \mathsf{i}) \\ (\mathsf{cpi}_{t,q}, q_2, \ldots, q_{l-1})\delta(\mathsf{cpi}'_{t,q}, q_2, \ldots, q_{l-1}, \mathsf{o}_1) \\ (\mathsf{cpi}'_{t,q}, q'_2, \ldots, q'_{l-1})\delta(\mathsf{cp}_{t,q}, q'_2, \ldots, q'_{l-1}, \mathsf{o}_2) \\ (\mathsf{cp}_{t,q}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, \mathsf{o}_1)\delta(\mathsf{cpd}_{t,q}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, q, \mathsf{o}_1) \\ (\mathsf{cpd}_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, \mathsf{o}_2)\delta(\mathsf{cpd}'_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, q, \mathsf{o}_2) \\ (\mathsf{cpd}'_{t,q}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, \mathsf{i}, r_{m+1})\delta(\mathsf{cp}_{t,r_{m+1}}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, q, \mathsf{i}) \\ (\mathsf{cp}_{t,q}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, r_{m+1}, \mathsf{o}_1)\delta(\mathsf{cpu}_{t,q}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, \mathsf{o}_1, q) \\ (\mathsf{cpu}_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, r_{m+1}, \mathsf{o}_2)\delta(\mathsf{cpu}'_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, \mathsf{o}_2, q) \\ (\mathsf{cpu}'_{t,q}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, r_{m+1}, \mathsf{i})\delta(\mathsf{cp}_{t,r_{m+1}}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, \mathsf{i}) \\ (\mathsf{cp}_{t,q}, q_2, \ldots, q_{l-1}, \mathsf{i})\delta(\mathsf{cpf}_t, q_2, \ldots, q_{l-1}) \\ (\mathsf{cpf}_t, q_2, \ldots, q_{l-1}, \mathsf{o}_1)\delta(\mathsf{cpf}'_t, q_2, \ldots, q_l) \\ (\mathsf{cpf}'_t, q'_2, \ldots, q'_{l-1}, \mathsf{o}_2)\delta(q'_1, \ldots, q'_l) \end{gathered}$$

The construction works in a depth-first-search fashion using a symbol i to mark the set, that is currently copied (and subsequently deleted), and two symbols o_1 and o_2 to mark the two copies, that are currently created. First (the control states named cpi) the markings are placed. Then either a new element of the multiset marked by i is selected, corresponding, new multisets are created under o_1 and o_2 and all markings are moved inwards (cpd-states) or copying of multiset marked by i has been completed, the multiset is deleted, and the markings are moved back outwards (cpu-states). When the markings are back on the outermost level, the copy process has been completed and the markings can be replaced (cpf-states).

Implementing `min`. A minimum rule $t = (q_1, \ldots, q_l)\mathtt{min}(q'_1, \ldots, q'_l)$ can be implemented in a similar fashion:

$$
\begin{gathered}
(q_1, \ldots, q_l)\delta(\mathsf{mini}_{t,q_l}, q_2, \ldots, q_{l-1}, \mathsf{i}_1) \\
(\mathsf{mini}_{t,q}, q'_2, \ldots, q'_l)\delta(\mathsf{mini}'_{t,q}, q'_2, \ldots, q'_{l-1}, \mathsf{i}_2) \\
(\mathsf{mini}'_{t,q}, q'_2, \ldots, q'_{l-1})\delta(\mathsf{min}_{t,q}, q'_2, \ldots, q'_{l-1}, \mathsf{o}) \\
(\mathsf{min}_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, \mathsf{o})\delta(\mathsf{mind}_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, q, \mathsf{o}) \\
(\mathsf{mind}_{t,q}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, \mathsf{i}_1, r_{m+1})\delta(\mathsf{mind}'_{t,r_{m+1}}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, q, \mathsf{i}_1) \\
(\mathsf{mind}'_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, \mathsf{i}_2, q)\delta(\mathsf{min}_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, q, \mathsf{i}_2) \\
(\mathsf{min}_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, r_{m+1}, \mathsf{o})\delta(\mathsf{minu}_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, \mathsf{o}, q) \\
(\mathsf{minu}_{t,q}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, r_{m+1}, \mathsf{i}_1)\delta(\mathsf{minu}'_{t,r_{m+1}}, q_2, \ldots, q_{l-1}, r_1, \ldots, r_m, \mathsf{i}_1) \\
(\mathsf{minu}'_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, q, \mathsf{i}_2)\delta(\mathsf{min}_{t,q}, q'_2, \ldots, q'_{l-1}, r_1, \ldots, r_m, \mathsf{i}_2) \\
(\mathsf{min}_{t,q}, q_2, \ldots, q_{l-1}, \mathsf{i}_1)\delta(\mathsf{minf}_t, q_2, \ldots, q_{l-1}) \\
(\mathsf{minf}_t, q'_2, \ldots, q'_{l-1}, \mathsf{i}_2)\delta(\mathsf{minf}'_t, q'_2, \ldots, q'_{l-1}) \\
(\mathsf{minf}'_t, q'_2, \ldots, q'_{l-1}, \mathsf{o})\delta(q'_1, \ldots, q'_l)
\end{gathered}
$$

It follows exactly the same idea, but deletes elements from two marked multisets (i_1 and i_2) and only creates elements in one marked multiset (o).

Hardy computations. Having these auxiliary operations at our disposal, we can now give the exact transition rules to implement Hardy computations. The encoding of the ordinal parameter α and the natural attribute n of a Hardy function $H^\alpha(n)$ is encoded into transitions as defined above. We have to come up with transition rules that allow four kinds of runs

1. $C_{\alpha+1,n} \to^* C_{\alpha,n+1}$,
2. $C_{\alpha,n+1} \to^* C_{\alpha+1,n}$,
3. $C_{\alpha+\lambda,n} \to^* C_{\alpha+\lambda_n,n}$ and
4. $C_{\alpha+\lambda_n,n} \to^* C_{\alpha+\lambda,n}$

in order to satisfy Lemma 8 without violating Lemma 9.

Case (1) is straightforward, we only have to remove some element from the multiset encoding the ordinal and move it to the multiset encoding the argument:

$$
\begin{gathered}
(\mathsf{main}, s, \omega)\delta(\mathsf{R1}, s) \\
(\mathsf{R1}, c)\delta(\mathsf{main}, c, 1)
\end{gathered}
$$

Case (2) works just the other way around:

$$(\mathsf{main}, c, 1)\delta(\mathsf{R2}, c)$$
$$(\mathsf{R2}, s)\delta(\mathsf{main}, s, \omega)$$

Case (3) requires to replace the smallest addend ω^β of a limit ordinal $\alpha + \omega^\beta$ with the nth element of its fundamental sequence $(\omega^\beta)_n$. If β is a limit ordinal, it has to be replaced by β_n, i.e. the same process has to be applied recursively. Otherwise, the immediate predecessor of $\beta' + 1 = \beta$ has to be copied n times. The states in the following constructions are parametrised by the recursion depth l.

$$(\mathsf{main}, s, \omega)\delta(\mathsf{R3}_0, s, a_1)$$
$$(\mathsf{R3}_l, \overbrace{s', \ldots, s'}^{l})\delta(\mathsf{R3s}_l, \overbrace{s', \ldots, s'}^{l}, s')$$
$$(\mathsf{R3s}_l, s, \overbrace{s, \ldots, s}^{l}, \omega)\delta(\mathsf{R3s}'_l, s, \overbrace{s, \ldots, s}^{l}, a_2)$$
$$(\mathsf{R3s}'_l, s, \overbrace{s, \ldots, s}^{l}, a_1)\mathsf{cp}(\mathsf{R3s}''_l, s', \overbrace{s', \ldots, s'}^{l}, \omega)$$
$$(\mathsf{R3s}''_l, s, \overbrace{s, \ldots, s}^{l}, a_2)\mathtt{min}(\mathsf{R3s}_l, s, \overbrace{s, \ldots, s}^{l}, a_1)$$
$$(\mathsf{R3s}_l, s, a_1, \omega)\delta(\mathsf{R3}_{l+1}, s, s, a_1)$$
$$(\mathsf{R3s}_l, s, \overbrace{s, \ldots, s}^{l}, a_1, \omega)\delta(\mathsf{R3c}_l, s, \overbrace{s, \ldots, s}^{l}, a_1)$$
$$(\mathsf{R3c}_l, s, \overbrace{s, \ldots, s}^{l}, a_1)\mathsf{cp}(\mathsf{R3c}'_l, s', \overbrace{s', \ldots, s'}^{l}, \omega)$$
$$(\mathsf{R3c}'_l, c, 1)\delta(\mathsf{R3c}''_l, c)$$
$$(\mathsf{R3c}''_l, c')\delta(\mathsf{R3c}_l, c, 1)$$
$$(\mathsf{R3c}_l, c)\delta(\mathsf{R3q}_l)$$
$$(\mathsf{R3q}_l, c')\delta(\mathsf{R3q}'_l, c)$$
$$(\mathsf{R3q}'_l, s)\delta(\mathsf{R3q}''_l)$$
$$(\mathsf{R3q}''_l, s', \overbrace{s', \ldots, s'}^{l})\delta(\mathsf{main}, s, \overbrace{\omega, \ldots, \omega}^{l})$$

The construction starts selecting the smallest addend, by copying the multiset marked by s to the multiset marked by s' in descending order. The descending order is ensured using the `min` operation introduced above. a_1 and a_2 are used to mark the currently largest and second largest addend. Once the copying process is stopped, a_1 marks the supposedly smallest addend, the construction moves down one level, and repeats this process. This part is implemented using the R3s-states. Once, a level is reached where the exponent is no longer a limit ordinal, one element is removed from the respective multiset (transition from R3s to R3c). Then, the copy operation is used to copy that exponent n times. This part is implemented by the R3c-states. Finally the multiset from the old ordinal is deleted and replaced by the newly computed ordinal (R3q-states). This construction might make several lossy errors in the sense that they result in a smaller ordinal to be computed. E.g. it might not select the smallest addend

at the cost of losing all smaller addends or it might stop at a level, where the exponent is still a limit ordinal. In this case instead of decreasing it by only one, a larger addend will be removed.

Case (4) can be handled similarly to (3). The construction recursively guesses the smallest addend ($\mathsf{R4s}$-states) as before. Then n copies of an addend ω^β have to be replaced by $\omega^{\beta+1}$ ($\mathsf{R4m}$-states). This is realised by deleting at most n elements in descending order and maintaining their minimum using the minimum operation. The construction counts the number of elements actually deleted and uses it as the new value for n, ensuring that a lossy error occurs in case less than n elements are removed. The exponent is then increased by one and the addend is moved to the newly created ordinal.

$$(\mathsf{main}, s, \omega)\delta(\mathsf{R4}_0, s, a_1)$$
$$(\mathsf{R4}_l, \overbrace{s', \dots, s'}^{l})\delta(\mathsf{R4s}_0, \overbrace{s', \dots, s'}^{l}, s')$$
$$(\mathsf{R4s}_l, s, \overbrace{s, \dots, s}^{l}, \omega)\delta(\mathsf{R4s}'_l, s, \overbrace{s, \dots, s}^{l}, a_2)$$
$$(\mathsf{R4s}'_l, s, \overbrace{s, \dots, s}^{l}, a_1)\mathtt{cp}(\mathsf{R4s}''_l, s', \overbrace{s', \dots, s'}^{l}, \omega)$$
$$(\mathsf{R4s}''_l, s, \overbrace{s, \dots, s}^{l}, a_2)\mathtt{min}(\mathsf{R4s}_l, s, \overbrace{s, \dots, s}^{l}, a_1)$$
$$(\mathsf{R4s}_l, s, a_1, \omega)\delta(\mathsf{R4}_{l+1}, s, s, a_1)$$
$$(\mathsf{R4s}_l, s, \overbrace{s, \dots, s}^{l}, \omega)\delta(\mathsf{R4m}_l, s, \overbrace{s, \dots, s}^{l}, a_2)$$
$$(\mathsf{R4m}_l, s, \overbrace{s, \dots, s}^{l}, a_2)\mathtt{min}(\mathsf{R4m}'_l, s, \overbrace{s, \dots, s}^{l}, a_1)$$
$$(\mathsf{R4m}'_l, c, 1)\delta(\mathsf{R4m}''_l, c)$$
$$(\mathsf{R4m}''_l, c')\delta(\mathsf{R4m}_l, c, 1)$$
$$(\mathsf{R4m}_l, s, \overbrace{s, \dots, s}^{l}, a_1)\delta(\mathsf{R4q}_l, s, \overbrace{s, \dots, s}^{l}, a_1, \omega)$$
$$(\mathsf{R4q}_l, s, \overbrace{s, \dots, s}^{l}, a_1)\mathtt{cp}(\mathsf{R4q}'_l, s', \overbrace{s', \dots, s'}^{l}, \omega)$$
$$(\mathsf{R4q}'_l, c)\delta(\mathsf{R4q}''_l)$$
$$(\mathsf{R4q}''_l, c')\delta(\mathsf{R4q}'''_l, c)$$
$$(\mathsf{R4q}'''_l, s)\delta(\mathsf{R4q}''''_l)$$
$$(\mathsf{R4q}''''_l, s', \overbrace{s', \dots, s'}^{l})\delta(\mathsf{main}, s, \overbrace{\omega, \dots, \omega}^{l})$$

Finally, observe that for $\alpha \leq (\Omega_k)_l$ the innermost (level-k) exponents of the CNF terms that can arise during the computation of $H^\alpha(n)$ are bounded by l because they are only decreased. Hence, using additional states $\omega^0, \dots, \omega^l \in Q$ on level $k-1$ to represent configurations $(\omega, \{(\omega, \emptyset) : i\})$ by $(\omega^i, \emptyset)$ avoids one level of nesting in $\mathcal{N}$.

C From $\mathrm{LTL}^{\downarrow}_{tqo}$ to Nested Counter Systems

In this section we provide the technical details of the reduction from the satisfiability problem for $\mathrm{LTL}^{\downarrow}_{A}$ formulae over tree-quasi-ordered attributes A to the coverability problem in NCS.

C.1 Reduction to Linear Orderings

We recall and prove Proposition 10.

Proposition 10 ($\mathrm{LTL}^{\downarrow}_{tqo}$ to $\mathrm{LTL}^{\downarrow}_{[k]}$)**.** *For a tree-quasi-ordered attribute set A of depth k every $\mathrm{LTL}^{\downarrow}_{A}$ formula can be translated to an equisatisfiable $\mathrm{LTL}^{\downarrow}_{[k]}$ formula of exponential size.*

Let Φ be an $\mathrm{LTL}^{\downarrow}_{A}$ formula. To translate Φ into an equisatisfiable $\mathrm{LTL}^{\downarrow}_{[k]}$ formula for some $k \in \mathbb{N}$ we first turn A into a *tree-partial-order* A' by collapsing maximal strongly connected components (SCC) and adjust Φ to obtain an equisatisfiable formula Φ' over $\mathrm{LTL}^{\downarrow}_{A'}$. Second, we show how to encode A'-attributed data words into $[k]$-attributed data words and translate Φ' to operate on this encoding.

Collapsing SCC. Let $C_{2,1}, \dots, C_{2,n_2}, C_{3,1}, \dots, C_{3,n_3}, \dots, C_{m,n_m} \subseteq A$ be all maximal strongly connected components in the graph of the tree-quasi-ordering $(A, \sqsubseteq)$ of size larger than 1 such that $|C_{i,j}| = i$. I.e., $C_{i,j}$ is the j-th distinct such component of size i. Notice that all $C_{i,j}$ are disjoint since they are maximal. Choose some arbitrary $x_{i,j} \in C_{i,j}$ from each component and remove all components from A but for those elements $x_{i,j}$. Thus we collapse all SCC in A and obtain a tree-partial-ordering A'. In the formula Φ we syntactically replace every attribute $x \in C_{i,j}$ by the corresponding representative $x_{i,j}$ and obtain an $\mathrm{LTL}^{\downarrow}_{A'}$ formula.

Due to the semantics of the logic being defined in terms of downward closures the only significant change upon collapsing SCC is their size. While the downward-closures of two SCCs that have different sizes cannot be isomorphic replacing them with a single attribute can make valuations for them equal wrt. $\simeq$. We therefore add the following constraint to Φ disallowing a collapsed model to assign the same data value to representatives of SCCs that had different size.

$$\bigwedge_{x_{i,j}, x_{i',j'} \in A' | i \neq i'} \mathrm{G}(\downarrow^{x_{i,j}} \neg \mathrm{F} \uparrow^{x_{i',j'}})$$

Compared to the original models of Φ this is not a restriction and thus every model of Φ still induces a model of Φ' and vice versa.

Frame encoding. In the following we assume that A is a tree(-partial)-ordering, i.e., it does not contain non-trivial SCCs. Let k be the depth of A, i.e., the length of the longest simple path starting at some root (minimal element). We can pad A, by additional attributes s.t. *every* maximal path in A has length k. The additional attributes added to A this way are not smaller than the original ones and hence do not affect the semantics of formulae over A except that the new attributes need to be assigned an arbitrary value. Thus,

regarding A as a forest, we can assume that every leaf is at level k (roots are at level 1).

Let $\ell_1, \ldots, \ell_n \in A$ be the leafs in A (enumerated in an *in-order* fashion). We use the ideas from [KSZ10, DHLT14] to encode an A-attributed data word $w = w_1 w_2 \ldots$ into a $[k]$-attributed data word $u = u_1 u_2 \ldots$ where a single position in w is represented by a *frame* of n positions in u. Then, each position $w_i = (a_i, \mathbf{d}_i)$ in w corresponds to the frame $u_{(i-1)n+1} \ldots u_{(in)}$ in u. In the i-th such frame, each position $u_{(i-1)n+j} = (a_i, \mathbf{g}_{i,j})$ carries the same letter a_i as w_i. The data valuation $\mathbf{g}_{(i,j)} \in \Delta^k$ at the j-th position in the frame shall represents the j-th "branch" $\mathbf{d}_i|_{\ell_j}$ of the valuation $\mathbf{d}_i$. Thus, let for a leaf ℓ_j in A be $x_{j,1} \sqsubseteq x_{j,2} \sqsubseteq \ldots \sqsubseteq x_{j,k} = \ell_j$ the attributes in $\mathsf{cl}(\ell_j)$, representing the branch in A from a root to ℓ_j. Now for $r \in [k]$ we let $\mathbf{g}_{(i,j)}(r) = \mathbf{d}_i(x_{j,r})$

Translation. Based on this encoding we can translate any $\mathrm{LTL}^{\downarrow}_{A}$ formula Φ to an $\mathrm{LTL}^{\downarrow}_{[k]}$ formula $\hat{\Phi}$ that specifies precisely the encodings of models of Φ. In particular, $\hat{\Phi}$ is satisfiable iff Φ is.

Given the (in-order) enumeration $\ell_1, \ldots, \ell_n \in A$ of leafs in A and an attribute $x \in A$ we let $\mathsf{sb}(x) = \mathtt{min}\{r \in [n] \mid x \sqsubseteq \ell_r\}$ denote the smallest index r of a branch containing x and $\mathsf{lb}(x) = \max\{r \in [n] \mid x \sqsubseteq \ell_r\}$ the largest such branch index. Further, we denote by $\mathsf{lvl}(x) = |\{x' \sqsubseteq x' \mid x' \in A\}|$ its level in A.

We can assume Φ to be in a normal form where every freeze quantifier $\downarrow^x$ is followed immediately by either an X, $\overline{\mathrm{X}}$ or $\uparrow^y$ operator for attributes $x, y \in A$. This is due to the following equivalences for arbitrary formulae ψ, ξ, letters $a \in \Sigma$ and attributes $x, y \in A$.

$$
\begin{array}{rcl@{\qquad}rcl}
\downarrow^x a & \equiv & a & \downarrow^x \mathrm{F}\,\psi & \equiv & (\downarrow^x \psi) \vee \downarrow^x \mathrm{X}\,\mathrm{F}\,\psi \\
\downarrow^x \downarrow^y \psi & \equiv & \downarrow^y \psi & \downarrow^x \mathrm{G}\,\psi & \equiv & (\downarrow^x \psi) \wedge \downarrow^x \overline{\mathrm{X}}\,\mathrm{G}\,\psi \\
\downarrow^x \neg\psi & \equiv & \neg\downarrow^x \psi & \downarrow^x (\psi\,\mathrm{U}\,\xi) & \equiv & (\downarrow^x \xi) \vee ((\downarrow^x \psi) \wedge \downarrow^x \mathrm{X}(\psi\,\mathrm{U}\,\xi)) \\
\downarrow^x (\psi \wedge \xi) & \equiv & (\downarrow^x \psi) \wedge (\downarrow^x \xi) & \downarrow^x (\psi\,\mathrm{R}\,\xi) & \equiv & (\downarrow^x \xi) \wedge ((\downarrow^x \psi) \vee \downarrow^x \overline{\mathrm{X}}(\psi\,\mathrm{R}\,\xi)) \\
\downarrow^x (\psi \vee \xi) & \equiv & (\downarrow^x \psi) \vee (\downarrow^x \xi)
\end{array}
$$

We can further assume that for every formula $\downarrow^x \uparrow^y$ we have $\mathsf{sb}(x) \leq \mathsf{sb}(y)$: if $x \sqsubset y$ or $x \sqsupseteq y$ we can completely remove the formula, replacing it with a contradiction or a tautology, respectively. Otherwise x and y are incomparable. Then, if $\mathsf{lvl}(x) < \mathsf{lvl}(y)$ the formula is again false and we can remove it. For $\mathsf{lvl}(x) = \mathsf{lvl}(y)$ we have $\downarrow^x \uparrow^y \equiv \downarrow^y \uparrow^x$ and can swap them if necessary. Finally, if $\mathsf{lvl}(x) > \mathsf{lvl}(y)$ there is a unique attribute $p \sqsubset x$ with $\mathsf{lvl}(p) = \mathsf{lvl}(y)$ and by the definition of the semantics we have $\downarrow^x \uparrow^y \equiv \downarrow^p \uparrow^y$. We can thus replace x by p and swap the attributes if necessary.

Next we extend the alphabet to $\Sigma' = \Sigma \times [n]$. The attached number is supposed indicate the relative position in every frame. This is enforced by a formula

$$
\beta_1 := \Sigma_i \wedge \mathrm{G}\left(\bigwedge_{i \in [n]} \Sigma_i \rightarrow \left((\overline{\mathrm{X}}\,\Sigma_{(i \mod n)+1}) \wedge \bigwedge_{j \in [n] \setminus \{i\}} \neg\Sigma_j\right)\right)
$$

where Σ_i for $i \in [n]$ stands for the formula $\bigvee_{a \in \Sigma}(a, i)$. Further, we impose that models actually have the correct structure and thereby encode an A-attributed

data word. The formula

$$\beta_2 := \bigwedge_{(a,i)\in\Sigma\times[n-1]} \mathrm{G}((a,i) \to \mathrm{X}(a,i+1))$$

expresses that the letter from Σ is constant throughout a frame and

$$\beta_3 := \bigwedge_{x\in A} \mathrm{G}\left(\Sigma_1 \to \mathrm{X}^{\mathsf{sb}(x)-1} \downarrow^{\mathsf{lvl}(x)} \left(\uparrow^{\mathsf{lvl}(x)} \mathrm{U}(\Sigma_{\mathsf{lb}(x)} \wedge \uparrow^{\mathsf{lvl}(x)})\right)\right)$$

ensures that the frame consistently encodes a valuation from Δ^A. Finally, we define the translation $t(\Phi)$ inductively for subformulae ψ, ξ of Φ, $x \in A$ and $a \in \Sigma$ as follows.

$$\begin{array}{rcl@{\qquad}rcl} t(\downarrow^x \psi) & := & \mathrm{X}^{\mathsf{sb}(x)-1} \downarrow^{\mathsf{lvl}(x)} t(\psi) & t(a) & := & a \\ t(\mathrm{X}\,\psi) & := & \bigwedge_{j=1}^{n} \Sigma_j \to \mathrm{X}^{n-j+1}\, t(\psi) & t(\neg\psi) & := & \neg t(\psi) \\ t(\psi\,\mathrm{U}\,\xi) & := & ((\Sigma_1 \to t(\psi))\,\mathrm{U}(\Sigma_1 \wedge t(\xi))) & t(\psi \wedge \xi) & := & t(\psi) \wedge t(\xi) \\ t(\uparrow^x) & := & \bigwedge_{j=1}^{n} \Sigma_j \to \mathrm{X}^{\mathsf{sb}(x)-j} \uparrow^{\mathsf{lvl}(x)} & & & \end{array}$$

We omit the remaining operators since they can be expressed in terms of the ones considered above.

To see that $\hat{\Phi} := t(\Phi) \wedge \beta_1 \wedge \beta_2 \wedge \beta_3$ exactly characterises the encodings of models of Φ consider the underlying invariant that all subformulae of Φ are always evaluated on the first position of a frame except those preceded by a freeze quantifier. Those that directly follow a freeze quantifier have the form $\mathrm{X}\,\psi$ or $\uparrow^x$ and are relocated to the first position of the successive frame or to the position encoding the branch of data values that needs to be checked, respectively.

C.2 From $\mathrm{LTL}^{\downarrow}_{[k]}$ to NCS

Recall Theorem 12.

Theorem 12. *For tree-quasi-ordered attribute sets A with depth k satisfiability of $\mathrm{LTL}^{\downarrow}_{A}$ can be reduced in exponential space to coverability in $(k+1)$-NCS.*

By Proposition 10 it suffices to show that given an $\mathrm{LTL}^{\downarrow}_{[k]}$ formula Φ we can construct a $(k+1)$-NCS $\mathcal{N}$ and two configurations $C_{init}, C_{final} \in \mathcal{C}_{\mathcal{N}}$ s.t. Φ is satisfiable if and only if C_{final} can be covered from C_{init}.

The idea is to construct from the $\mathrm{LTL}^{\downarrow}_{[k]}$ formula Φ a $(k+1)$-NCS $\mathcal{N}$ that guesses an (abstraction of a) data word $w \in (\Sigma \times \Delta^k)^+$ position-wise starting with the last position and prepending new ones. Simultaneously, $\mathcal{N}$ maintains a set of *guarantees* for the so far constructed suffix of w. These guarantees are subformulae φ of Φ together with an (abstraction of a) data valuation representing the register value under which φ is satisfied by the current suffix of w. Guarantees can be assembled to larger formulae in a way that maintains satisfaction by the current suffix of w. Then Φ is satisfiable if and only if there is a reachable configuration of $\mathcal{N}$ that contains Φ as one of possibly many guarantees.

Normal form. We fix for the rest of this section $k \in \mathbb{N}$ and an $\mathrm{LTL}^{\downarrow}_{[k]}[\mathrm{X},\overline{\mathrm{X}},\mathrm{U},\mathrm{R}]$ formula Φ over the finite alphabet Σ and the data domain Δ. W.l.o.g. we restrict to the reduced set of temporal operators and expect Φ to be in *negation normal form*, i.e., negation appears only in front of letters $a \in \Sigma$ and check operators $\uparrow^i$ for $i \in [k]$. Further, we assume that every check operator $\uparrow^i$ occurs within the scope of the freeze quantifier $\downarrow^j$ of level $j \geq i$ since otherwise the check necessarily fails and the formula can easily be simplified syntactically. Let $\mathsf{sub}(\Phi)$ denote the set of syntactical subformulae as well as the unfoldings of U and R formulae.

State space. For the $\mathrm{LTL}^{\downarrow}_{[k]}[\mathrm{X}, \overline{\mathrm{X}}, \mathrm{U}, \mathrm{R}]$ formula Φ we construct a $(k+1)$-NCS $\mathcal{N}_\Phi = (Q, \delta)$ as follows. The state space is defined as $Q = Q_{\mathsf{ctrl}} \cup Q_{\mathsf{cell}}$ where

$$
\begin{aligned}
Q_{\mathsf{ctrl}} &= Q_{\mathsf{add}} \cup Q_{\mathsf{next}} \cup Q_{\mathsf{setup}} \cup Q_{\mathsf{stor}},\\
Q_{\mathsf{add}} &= \{\mathsf{add}\} \times (\Sigma \cup (\Sigma \times \mathsf{sub}(\Phi)),\\
Q_{\mathsf{next}} &= \{\mathsf{next}_1, \mathsf{next}_2, \mathsf{copy}, \mathsf{copy}_{bt}\} \cup (\{\mathsf{copy}\} \times 2^{\mathsf{sub}(\Phi)}),\\
Q_{\mathsf{setup}} &= \{\mathsf{setup}\},\\
Q_{\mathsf{stor}} &= \{\mathsf{stor}, \mathsf{stor}^{\checkmark}, \mathsf{aux}, \mathsf{aux}^{\checkmark}\} \text{ and}\\
Q_{\mathsf{cell}} &= \{\checkmark, \text{✗}\} \times 2^{\mathsf{sub}(\Phi)}.
\end{aligned}
$$

The two outer-most levels (level 0 and 1) of configurations will only use states from Q_{ctrl} and control the management of the configurations of level 2 to k below. These configurations only use states from Q_{cell} and implement a *storage* for a tree structure (more precisely, a forest) of depth k represented by a multiset of configurations of level 2. Every node in that forest, a *cell*, stores a set of formulae and is checked (✓) or unchecked (✗).

Next we define the transition rules $\delta \subseteq \bigcup_{i,j \in [k+1]} (Q^i \times Q^j)$.

Setup phase. The storage of the initial configuration

$$C_{init} = \mathsf{setup}(\mathsf{stor}(q_1(\dots q_{k-1}(q_k)\dots)))$$

with $q_1 = \dots = q_k = (\checkmark, \emptyset)$ is empty except for a single checked branch of length k. We allow the NCS to arbitrarily add new (unchecked) branches and then populate the branches with guarantees of the form $\overline{\mathrm{X}}\,\varphi \in \mathsf{sub}(\Phi)$. Thus, let

$$(\mathsf{setup}, \mathsf{stor}, q_1, \dots, q_i)\delta(\mathsf{setup}, \mathsf{stor}, q_1, \dots, q_i, q'_{i+1}, \dots, q'_k)$$
$$(\mathsf{setup}, \mathsf{stor}, q_1, \dots, q_i, (m, F))\delta(\mathsf{setup}, \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\overline{\mathrm{X}}\,\varphi\}))$$

for all $0 \leq i < k$, $q_1, \dots, q_i \in Q_{\mathsf{cell}}$, $q'_{i+1} = \dots = q'_k = (\text{✗}, \emptyset)$, $m \in \{\checkmark, \text{✗}\}$, $F \subseteq \mathsf{sub}(\Phi)$ and $\overline{\mathrm{X}}\,\varphi \in \mathsf{sub}(\Phi)$.

Construction phase. After the initial setup the NCS guesses a letter $a \in \Sigma$ by applying

$$(\mathsf{setup})\delta((\mathsf{add}, a)).$$

New atomic formulae can be added by the rules

$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F))\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{a\}))$$
$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F))\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\neg b\}))$$
$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q^{\checkmark}_{i+1}, \dots, (m_j, F_j))\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q^{\checkmark}_{i+1}, \dots, (m_j, F_j \cup \{\uparrow^{\ell}\}))$$
$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (\text{✗}, F))\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (\text{✗}, F \cup \{\neg\uparrow^{\ell'}\}))$$

and existing formulae can be combined by rules

$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\varphi\}))\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\varphi, \varphi \vee \psi\}))$$
$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\varphi\}))\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\varphi, \psi \vee \varphi\}))$$
$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\varphi, \psi\}))\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\varphi, \psi, \varphi \wedge \psi\}))$$
$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\varphi, \psi\}))\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\varphi, \psi, \psi \wedge \varphi\}))$$
$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (\checkmark, F \cup \{\varphi\}))\delta((\mathsf{add}, a, \downarrow^{i+1}\varphi), \mathsf{stor}, q_1, \dots, q_i, (\checkmark, F \cup \{\varphi\}))$$
$$((\mathsf{add}, a, \downarrow^{i+1}\varphi), \mathsf{stor}, q_1, \dots, q_j, (m, F))\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_j, (m, F \cup \{\downarrow^{i+1}\varphi\}))$$

$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\psi \vee (\varphi \wedge \mathrm{X}(\varphi\,\mathrm{U}\,\psi))\}))$$
$$\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\varphi\,\mathrm{U}\,\psi\}))$$
$$((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\psi \wedge (\varphi \vee \overline{\mathrm{X}}(\varphi\,\mathrm{R}\,\psi))\}))$$
$$\delta((\mathsf{add}, a), \mathsf{stor}, q_1, \dots, q_i, (m, F \cup \{\varphi\,\mathrm{R}\,\psi\}))$$

for, respectively, $F, F_j \subseteq \mathsf{sub}(\Phi)$, $m, m_j \in \{\checkmark, \text{✗}\}$, $0 \leq i, j < k$, $\ell \in [i+1]$, $i < \ell' \leq k$, $q_1, \dots, q_k \in Q_{\mathsf{cell}}$, $q^{\checkmark}_{i+1} = (\checkmark, F)$, $b \in \Sigma \setminus \{a\}$ and $\varphi, \psi, \varphi \vee \psi, \psi \vee \varphi, \varphi \wedge \psi, \psi \wedge \varphi, \downarrow^{i+1}\varphi, \uparrow^{\ell}, a, \neg b, \varphi\,\mathrm{U}\,\psi, \varphi\,\mathrm{R}\,\psi \in \mathsf{sub}(\Phi)$.

Advancing phase. To ensure consistency, prepending of X and $\overline{\mathrm{X}}$ operators can only be done for all stored formulae at once. This corresponds to guessing a new position in a data word, prepending it to the current one and computing a set of guarantees for that preceeding position from the guarantees of the current position.

The NCS can enter the advancing phase by the rules

$$((\mathsf{add}, a))\delta(\mathsf{next}_1, \mathsf{aux}^{\checkmark})$$

for $a \in \Sigma$. This also creates an auxiliary storage. Next, the original storage is copied cell by cell to the auxiliary storage. Upon copying a cell the formulae stored within are preceeded by next-time operators. To this end, for $F \subseteq \mathsf{sub}(\Phi)$ we denote by $F_{\mathrm{X}} = \{\mathrm{X}\,\varphi, \overline{\mathrm{X}}\,\varphi \in \mathsf{sub}(\Phi) \mid \varphi \in F\}$.

The markings are now utilised as pointers to the cell currently being copied. The rules

$$(\mathsf{next}_1, \mathsf{stor}, q_1^{\checkmark}, \ldots, q_k^{\checkmark})\delta(\mathsf{copy}, \mathsf{stor}^{\checkmark}, q_1^{\text{✗}}, \ldots, q_k^{\text{✗}})$$

for $q_1^{\checkmark} = (\checkmark, F_1), \ldots, q_k^{\checkmark} = (\checkmark, F_k)$, $q_1^{\text{✗}} = (\text{✗}, F_1), \ldots, q_k^{\text{✗}} = (\text{✗}, F_k)$, where $F_1, \ldots, F_k \subseteq \mathsf{sub}(\Phi)$, set these pointers to the root of the storage

To allow the NCS to copy the cells over in a depth-first, *lossy* manner let

$$(\mathsf{copy}, \mathsf{stor}^{\checkmark}, (\text{✗}, F))\delta((\mathsf{copy}, F), \mathsf{stor}, (\checkmark, F))$$
$$((\mathsf{copy}, F), \mathsf{aux}^{\checkmark})\delta(\mathsf{copy}, \mathsf{aux}, (\checkmark, F_{\mathrm{X}}))$$
$$(\mathsf{copy}, \mathsf{stor}, q_1, \ldots, q_i, (\checkmark, F'), (\text{✗}, F))\delta((\mathsf{copy}, F), \mathsf{stor}, q_1, \ldots, q_i, (\text{✗}, F'), (\checkmark, F))$$
$$((\mathsf{copy}, F), \mathsf{aux}, q_1, \ldots, q_i, (\checkmark, F'))\delta(\mathsf{copy}, \mathsf{aux}, q_1, \ldots, q_i, (\text{✗}, F'), (\checkmark, F_{\mathrm{X}}))$$

for, respectively, $F, F' \subseteq \mathsf{sub}(\Phi)$, $0 \leq i < k$ and $q_1 \ldots, q_i \in Q_{\mathsf{cell}}$. To allow for backtracking we let

$$(\mathsf{copy}, \mathsf{stor}, q_1, \ldots, q_i, (\text{✗}, F), (\checkmark, F'))\delta(\mathsf{copy}_{bt}, \mathsf{stor}, q_1, \ldots, q_i, (\checkmark, F)$$
$$(\mathsf{copy}_{bt}, \mathsf{aux}, q_1, \ldots, q_i, (\text{✗}, F), (\checkmark, F'))\delta(\mathsf{copy}, \mathsf{aux}, q_1, \ldots, q_i, (\checkmark, F), (\text{✗}, F'))$$
$$(\mathsf{copy}, \mathsf{stor}, (\checkmark, F))\delta(\mathsf{copy}_{bt}, \mathsf{stor}^{\checkmark})$$
$$(\mathsf{copy}_{bt}, \mathsf{aux}, (\checkmark, F))\delta(\mathsf{copy}, \mathsf{aux}^{\checkmark}, (\text{✗}, F))$$

for $0 \leq i < k$, $F, F' \subseteq \mathsf{sub}(\Phi)$ and $q_1, \ldots, q_i \in Q_{\mathsf{cell}}$.

Finally the original storage can be replaced by the auxiliary one by

$$(\mathsf{copy}, \mathsf{stor}^{\checkmark})\delta(\mathsf{copy}_{bt})$$
$$(\mathsf{copy}_{bt}, \mathsf{aux}^{\checkmark})\delta(\mathsf{next}_2, \mathsf{stor})$$

The storage is now (partially) copied to the auxiliary storage. To enter the construction phase and thereby complete the transition from the old position in the imaginary data word to the preceeding position a new checked branch and a new letter from Σ is guessed by

$$(\mathsf{next}_2, \mathsf{stor}, (\text{✗}, F_1), \ldots, (\text{✗}, F_i))\delta((\mathsf{add}, a), \mathsf{stor}, (\checkmark, F_1) \ldots, (\checkmark, F_k))$$

for any $a \in \Sigma$, $0 \leq i \leq k$, $F_1, \ldots, F_i \subseteq \mathsf{sub}(\Phi)$ and $F_{i+1} = \ldots = F_k = \emptyset$.

C.3 Correctness

The NCS $\mathcal{N} = (Q, \delta)$ that construct above maintains a forest of depth k where every node is labelled by a set of subformulae of Φ. Configurations reachable from the initial configuration

$$C_{init} = \mathsf{setup}(\mathsf{stor}(q_1(\dots q_{k-1}(q_k)\dots)))$$

with $q_1 = \dots = q_k = (\checkmark, \emptyset)$ always have the form $q_{ctrl}(q_{\mathsf{stor}}(X) + X')$ or $\mathsf{copy}_{bt}(\mathsf{aux}^{\checkmark}(X))$ where X is a multiset of configurations of level 2 that represents a forest T_C of depth k.Let V_C be the set of nodes and $F : V_C \to 2^{\mathsf{sub}(\Phi)}$ their labelling by sets of formulae. For a node $v \in V_C$ at level i (roots have level 1) in T_C we denote by $\rho(v) = v_1 \dots v_i$ the unique path from a root to $v_i = v$.

The structure of the forest represents the context in which the individual formulae are assumed to be evaluated. To formalise this let $con : V_C \to \Delta$ be a labelling of T_C by data values called *concretisation*. Such a labelling induces a set $G_{con}(T_C) \subseteq \mathsf{sub}(\Phi) \times \Delta^k$ with $(\varphi, \mathbf{d}) \in G_{con}(T_C)$ iff

- $\varphi \in F(v)$ for some node $v \in V_C$ with $\rho(v) = v_1 \dots v_j$ and
- $\mathbf{d} \in \Delta^j$ with $\mathbf{d}(i) = con(v_i)$ for $i \in [j]$.

Now, let $w = (a, \mathbf{d})u \in (\Sigma \times \Delta^k)^+$ be a data word. We say that w and a configuration $C = (q_0, M)$ are *compatible* if and only if there is a concretisation $con : V \to \Delta$ such that

- for all $(\varphi, \mathbf{d}') \in G_{con}(T_C)$ we have $(w, 1, \mathbf{d}') \models \varphi$ (guarantees are satisfied),
- $q_0 \notin \{(\mathsf{add}, b), (\mathsf{add}, b, \varphi) \mid b \in \Sigma \setminus \{a\}, \varphi \in \mathsf{sub}(\Phi)\}$ (letter is compatible) and
- if $q_0 \notin Q_{\mathsf{next}}$ and $v_1 \dots v_k$ is the unique path in T_C corresponding to the checked cells in C then $(con(v_1), \dots, con(v_k)) = (\mathbf{d}(1), \dots, \mathbf{d}(k))$ (valuation is compatible).

Lemma 20 (Invariant)**.** *Let $C \to C'$ be two configurations reachable from C_{init} and $w \in (\Sigma \times \Delta^k)^+$ a $[k]$-attributed data word such that T_C and w are compatible. Then there is a $[k]$-attributed data word $w' \in (\Sigma \times \Delta^k)^+$ such that $T_{C'}$ and w' are compatible.*

Proof. The initial configuration C_{init} does not contain any guarantees and hence every data word is compatible with $T_{C_{init}}$. The only formulae added during the setup phase are of the form $\overline{\mathrm{X}}\,\varphi$. Thus, every configuration reachable during this phase is compatible at least with every data word of length 1.

Consider a configuration C in the construction phase being compatible with a data word $w \in (\Sigma \times \Delta^k)^+$ due to a concretisation *con*. It is easy to see that the atomic formulae that can be added are satisfied on w under the same concretisation *con*. Also, the Boolean combinations of satisfied formulae that can be added remain satisfied and the folding of temporal operators respects the corresponding equivalences.

A rule adding a formula $\downarrow^i \varphi$ can obviously only be executed if φ is present in the marked cell at level i. Since in particular the valuation $\mathbf{d}$ of the first position of w is compatible with the marking there is $(\varphi, \mathbf{d}|_i) \in G_{con}(T_C)$ and

$(w, 1, \mathbf{d}|_i) \models \varphi$. Hence $(w, 1, \mathbf{d}') \models \downarrow^i \varphi$ for any valuation $\mathbf{d}'$ and $\downarrow^i \varphi$ can be put into any cell in T_C without breaking compatibility with w under *con*.

Consider a configuration C in the advancing phase. The transition rules staying in the phase do not add any new formula to any cell in the storage of the configuration and hence any word compatible with C remains compatible.

Finally, assume that w is compatible with a configuration C due to a concretisation $con : V_C \to \Delta$ and that a transition rule of the form

$$(\mathsf{next}_2, \mathsf{stor}, (\text{✗}, F_1), \ldots, (\text{✗}, F_i))\delta((\mathsf{add}, a), \mathsf{stor}, (\text{✓}, F_1) \ldots, (\text{✓}, F_k)),$$

for $a \in \Sigma$, $0 \leq i \leq k$, $F_1, \ldots, F_i \subseteq \mathsf{sub}(\Phi)$ and $F_{i+1} = \ldots = F_k = \emptyset$, is applied to obtain a configuration C'.

Let $(a', \mathbf{d}')$ be the first position of w and $d_{i+1}, \ldots, d_k \in \Delta \setminus \mathsf{img}(con)$ data values that are not assigned to any node in T_C by *con*. We define a new valuation $\mathbf{d} \in \Delta^k$ such that $\mathbf{d}|_i = \mathbf{d}'|_i$ and $\mathbf{d}(j) = d_j$ for $i < j \leq k$. Then the word $(a, \mathbf{d})w$ is compatible with C' witnessed by the concretisation $con' : V_{C'} \to \Delta$ with $con'(v) = con(v)$ for nodes $v \in V_C$ that were already present in T_C and $con'(v'_j) = d_j$ for the new nodes v_j $(i < j \leq k)$ created by the rule. □

As a consequence of the previous lemma we conclude that if a configuration C containing Φ as guarantee is reachable from the initial configuration C_{init} then it is satisfiable. We allow the NCS to enter a specific target state q_{final} once the formula Φ is encountered somewhere in the current tree. Thus, a path covering $C_{final} = q_{final}$ proves Φ satisfiable. Conversely, if Φ has some model w than the NCS $\mathcal{N}$ as constructed above can guess according to the letters and valuations along the word and assemble Φ from its subformulae.

D From NCS to $\mathrm{LTL}^{\downarrow}_{tqo}$

We provide the detailed construction to prove Theorem 13.

Theorem 13. *The coverability problem of k-NCS can be reduced in logarithmic space to $\mathrm{LTL}^{\downarrow}_{[k]}$ satisfiability.*

Let $\mathcal{N} = (Q, \delta)$ be a k-NCS. We are interested in describing witnesses for coverability. It suffices to construct a formula $\Phi_{\mathcal{N}}$ that characterises precisely those words that encode a *lossy* run from some configuration C_{start} to some configuration C_{end}. We call a sequence $C_0 C_1 \dots C_n$ of configurations $C_j \in \mathcal{C}_{\mathcal{N}}$ a lossy run from C_0 to C_n if there is a sequence of intermediate configurations $C'_0 \dots C'_{n-1}$ such that $C_i \succeq C'_i \to C_{i+1}$ for $0 \leq i < n$, i.e.,

$$C_0 \succeq C'_0 \to C_1 \succeq C'_1 \to \dots \to C_{n-1} \succeq C'_{n-1} \to C_n.$$

Then C_{end} is coverable from C_{start} if and only if there is a lossy run from C_{start} to some $C_n \succeq C_{end}$. For $\mathcal{N}$ we construct a formula

$$\Phi_{\mathcal{N}} = \Phi_{\text{conf}} \wedge \Phi_{\text{flow}} \wedge \Phi_{\text{rn}} \wedge \Phi_{\text{inc}} \wedge \Phi_{\text{dec}} \wedge \Phi_{\text{start}} \wedge \Phi_{\text{end}}.$$

where

- Φ_{conf} describes the shape of a word to encode a sequence of configurations,
- Φ_{flow} enforces that in addition to the plain sequence of encoded configurations there are annotations that indicate which transition rule is applied and which part of configuration is affected by the rule,
- Φ_{rn}, Φ_{inc}, Φ_{dec}, encode the correct effect of transition of the respective type (see below),
- Φ_{start} encodes that the encoded run starts with C_{start} and
- Φ_{end} encodes that the encoded run ends with some configuration $C \succeq C_{end}$.

We omit to construct Φ_{start} and Φ_{end} since it is straightforward given the considerations below. For easier reading we use an alphabet of the form $\Sigma = 2^{AP}$ where AP is a set of *atomic propositions*. Formally, every proposition $p \in AP$ used in a formula below could be replaced by

$$\bigvee_{a \in \Sigma \mid p \in a} a$$

to adhere to the syntax defined in Section 1.

D.1 Configurations

A configuration $C = (q, M) \in \mathcal{C}_{\mathcal{N}}$ of some k-NCS $\mathcal{N} = (Q, \delta)$ can be interpreted as a tree T of depth at most $k+1$ where the root carries q as label. The children of the root are the subtrees $T_{(1,1)}, \dots, T_{(1,n)}$ represented by the configurations of level 1 contained in the multiset M.

Similar to the approach of Proposition 10 we encode such a tree as $[k]$-attributed data word. We use an alphabet Σ where every letter $a \in \Sigma$ encodes a

$(k+1)$-tuple of states from Q, i.e., a possible branch in the tree. Then a sequence of such letters represents a set of branches that form a tree. The data valuations represent the structure of the tree, i.e., which of the branches share a common prefix. Two branches represented by positions $(a, \mathbf{d}), (a', \mathbf{d}') \in \Sigma \times \Delta^k$ are considered to be identical up to level $0 \leq i \leq k$ if and only if $(\mathbf{d}(1), \ldots, \mathbf{d}(i)) = (\mathbf{d}'(1), \ldots, \mathbf{d}'(i))$. Notice that the tuples of states represented by a and a' must also coincide on their i-th prefix if (but not only if) $\mathbf{d}$ and $\mathbf{d}'$ do.

For technical reasons we require that positions are arranged such that in between two positions representing branches with a common prefix of length i there is no position representing a branch that has a different prefix of length i. Further, this representation is interlaced: it refers only to odd positions. The even position in between are used to represent an exact copy of the structure but uses different data values. An example is shown in Figure 3.

We specify the shape of data words that encode (sequences of) configurations by the following formulae. For convenience we assume w.l.o.g. that the NCS uses a distinct set of states $Q_i \subseteq Q$ for each level $0 \leq i \leq k$ that includes an additional state $-_i \in Q_i$ not occurring in any transition rule from δ.

- Positions are marked by even/odd.

$$(\mathsf{odd} \to \overline{\mathrm{X}}(\neg\mathsf{odd} \wedge \overline{\mathrm{X}}\,\mathsf{odd})) \wedge (\mathsf{even} \leftrightarrow \neg\mathsf{odd})$$

- Data values are arranged in blocks. Once a block ends, the respective value (valuation prefix) will never occur again (on an odd position).

$$\bigwedge_{i\in[k]} \downarrow^k((\mathrm{X}\,\mathrm{X}\neg\uparrow^i) \wedge \mathsf{odd}) \to \neg\,\mathrm{X}\,\mathrm{F}(\mathsf{odd} \ \wedge \uparrow^i) \tag{14}$$

- Positions in the same block on level i carry the same states up to level i.

$$\bigwedge_{i\in[k]} \bigwedge_{q\in Q_i} (q \wedge \downarrow^k \mathrm{X}\,\mathrm{X}\uparrow^i) \to \mathrm{X}\,\mathrm{X}\,q$$

- Even positions mimic precisely the odd positions but use different data values.

$$\mathsf{odd} \to ((\bigwedge_{q\in Q} q \leftrightarrow \mathrm{X}\,q) \wedge (\downarrow^1 \mathrm{X}\neg\uparrow^1) \wedge \bigwedge_{i\in[k]} \downarrow^k \mathrm{X}\,\mathrm{X}\uparrow^i \leftrightarrow \mathrm{X}(\downarrow^k \mathrm{X}\,\mathrm{X}\uparrow^i)) \tag{15}$$

- State propositions are obligatory and mutually exclusive on every level.

$$\begin{aligned} &\left(\bigwedge_{0\leq i\leq k} \bigvee_{q\in Q_i} q\right) \\ \wedge &\bigwedge_{0\leq i\leq k} \bigwedge_{q,q'\in Q_i | q\neq q'} q \to \neg q' \end{aligned}$$

- Branches shorter than $k+1$ are padded by states $-_i \in Q$.

$$\begin{aligned} &\Big(\bigwedge_{i\in[k-1]} -_i \to -_{i+1}\Big) \\ \wedge &\bigwedge_{i\in[k-1]} (\downarrow^k \mathrm{X}\,\mathrm{X}\uparrow^i) \to \neg -_{i+1} \wedge \mathrm{X}\,\mathrm{X}\neg -_{i+1} \end{aligned} \tag{16}$$

- The proposition \$ is used to mark the first position of every configuration and can thus only occur on odd positions and the beginning of a new block.

$$(\$ \to \mathsf{odd}) \wedge ((\downarrow^1 \mathrm{X}\,\mathrm{X} \uparrow^1) \to \mathrm{X}\,\mathrm{X}\,\neg\$)$$

- Freshness propositions mark only positions carrying a valuations of which the prefix of length i has not occurred before.

$$\bigwedge_{i\in[k]} \downarrow^i \neg\,\mathrm{X}\,\mathrm{F}(\uparrow^i \wedge\ \mathsf{fresh}_i)$$

Let φ be the conjunction of these constraints and $\Phi_{\mathrm{conf}} := \$ \wedge \mathrm{G}\,\varphi$.

D.2 Control Flow

To be able to formulate the effect of transition rules without using past-time operators we encode the runs *reversed*. Given that a data word encodes a sequence $C_0C_1\dots C_n$ of configurations as above we model the (reversed) control flow of the NCS $\mathcal{N} = (Q, \delta)$ by requiring that every configuration but for the last be annotated by some transition rule $t_j \in \delta$ for $0 \le j < n$.

$$\mathrm{G}((\$ \wedge \mathrm{X}\,\mathrm{F}\,\$) \leftrightarrow \bigvee_{t\in\delta} t) \tag{17}$$

Now, the following constraints impose that this labelling by transitions actually represents the reversal of a lossy run. That is, for every configuration C_j in the sequence (for $0 \le j < n$) with annotated transition rule t_j there is a configuration C' (not necessarily in the sequence) such that $C' \xrightarrow{t} C_j$ and $C' \preceq C_{j+1}$.

Marking rule matches. Consider a position C_j in the encoded sequence that is annotated by a transition $t_j = ((q_0, \dots, q_i), (q'_0, \dots, q'_j)) \in \delta$. In order for C_j, t_j and C_{j+1} to encode a correct (lossy) transition first of all there must be a branch in C_j that matches $(q'_0, \dots, q'_j)$. We require that one such branch is marked by propositions $\checkmark_1 \dots \checkmark_j$:

$$\bigwedge_{t=((q_0,\dots,q_i),(q'_0,\dots,q'_j))\in\delta} \mathrm{G}\,\Big(t \to\ ((\mathrm{X}\,\neg\$)\,\mathrm{U}(\mathsf{even} \wedge \bigwedge_{\ell\in[j]} \checkmark_\ell \wedge q'_\ell)) \\ \wedge (q'_0 \wedge \neg\checkmark_{j+1} \wedge \dots \wedge \neg\checkmark_k)\,\mathrm{U}\,\$\Big)$$

This selects a branch of length j in every configuration. An operation that affects this branch may also affect other branches sharing a prefix. Thus, they are supposed to be marked accordingly. Since a node at some level $i \in [k]$ in the configuration tree is encoded by a block of equal data valuations on level i, blocks are marked entirely or not at all.

$$\bigwedge_{i\in[k]} \mathrm{G}(\downarrow^k \mathrm{X}\,\mathrm{X} \uparrow^i \to (\checkmark_i \leftrightarrow \mathrm{X}\,\mathrm{X}\checkmark_i))$$

Moreover, at most one block is marked in every configuration frame.

$$\bigwedge_{i\in[k]} \mathrm{G}\left((\checkmark_i \wedge \downarrow^k \mathrm{X}\,\mathrm{X}\,\neg\uparrow^i) \rightarrow \mathrm{X}((\neg\checkmark_i)\,\mathrm{U}\,\$)\right)$$

Now the markers indicate which positions in the word are affected by their respective transition rule. Notice, that the even positions are supposed to carry the marking. Let Φ_{flow} be the conjunction of the three formulae above and that in Equation 17.

D.3 Transition Effects

It remains to assert the correct effect of each transition rule to the marked branches. We distinguish three rule types. Let

$$\delta_{rn} = \{((q_0,\ldots,q_i),(q'_0,\ldots,q'_i)) \in \delta \mid 0 \leq i \leq k\}$$

be the set of *renaming* transition rules,

$$\delta_{dec} = \{((q_0,\ldots,q_i,q_{i+1},\ldots,q_j),(q'_0,\ldots,q'_i)) \in \delta \mid 0 \leq i < j \leq k\}$$

be the set of *decrementing* transition rules and

$$\delta_{inc} = \{((q_0,\ldots,q_i),(q'_0,\ldots,q'_i,q'_{i+1},\ldots,q'_j)) \in \delta \mid 0 \leq i < j \leq k\}$$

be the set of *incrementing* transition rules. Then $\delta = \delta_{rn} \cup \delta_{dec} \cup \delta_{inc}$.

Renaming rules. Let

$$\Phi_{\text{rn}} = \bigwedge_{t=((q_0,\ldots,q_i),(q'_0,\ldots,q'_i))\in\delta_{rn}} \mathrm{G}\,(t \rightarrow \mathsf{copy}(q_0,\ldots,q_i))\,. \tag{18}$$

The formula specifies that whenever a configuration C_n is supposed to be obtained from a configuration C_{n+1} using a renaming transition rule $t \in \delta_{rn}$ then every branch in C_n is also present in C_{n+1}. Moreover the states $q'_0,\ldots,q'_i$ in the marked branch should be replaced by $q_0,\ldots,q_i$.

The idea to realise this is to use the interlaced encoding of configurations to link (identify) branches from a configuration C_n with the configuration C_{n+1} in the sequence represented by a potential model. We consider a branch in C_n linked (on level $i \in [k]$) to a branch in C_{n+1} if the corresponding even position in C_n and the corresponding odd position in C_{n+1} carry the same valuation (up to level i). Since valuations uniquely identify a particular block (Equation 14) the following formula enforces for a position that there is a corresponding position in a unique block at level i of the next configuration where additionally φ is satisfied.

$$\mathsf{link}_i(\varphi) = \downarrow^k((\neg\$)\,\mathrm{U}\left(\$ \wedge \left((\mathrm{X}\,\neg\$)\,\mathrm{U}(\uparrow^i \wedge\ \mathsf{odd} \wedge\ \varphi)\right)\right)).$$

For $i = k$ a block represents an individual branch in a configuration and the formula $\mathsf{link}_k(\varphi)$ enforces that there is a unique corresponding branch in the consecutive configuration.

The even positions in turn mimic the odd ones using different data values (Equation 15). Thereby we can create a chain of branches that are linked and thus identified. We use this to enforce that for renaming transition rules, each branch present in a configuration C_n will again occur in C_{n+1} ensuring that the sequence is "gainy" wrt. the branches—and hence lossy when being reversed.

Using this we define the copy formula in Equation 18 as

$$\mathsf{copy}(q_0,\ldots,q_i) \;=\; \left(\mathsf{even} \rightarrow \left(\begin{array}{c} \mathsf{link}_k(q_0) \;\wedge \big(\bigwedge_{\ell\in[i]} (\checkmark_\ell \rightarrow \mathsf{link}_k(q_\ell))\big) \\ \wedge \bigwedge_{\ell\in[k],q\in Q_\ell} (\neg\checkmark_\ell \wedge q) \rightarrow \mathsf{link}_k(q) \end{array}\right)\right) \mathrm{U}\,\$$

Decrementing rules. We address this case by the formula

$$\Phi_{\mathrm{dec}} = \bigwedge_{t=((q_0,\ldots,q_j),(q'_0,\ldots,q'_i))\in\delta_{dec}} \mathrm{G}\left(t \rightarrow \left(\begin{array}{c} \mathsf{copy}(q_0,\ldots,q_i) \\ \wedge\ \mathsf{new}_i(q_0,\ldots,q_j) \end{array}\right)\right) \tag{19}$$

where $i < j$ and

$$\mathsf{new}_i(q_0,\ldots,q_j) = (\neg\,\mathrm{X}\,\$)\,\mathrm{U}(\checkmark_i \wedge \mathsf{link}_i(\mathsf{fresh}_{i+1} \wedge q_0 \wedge \ldots \wedge q_j))$$

ensures that a configuration C_{n+1} actually contains a branch that can be removed by a decrementing rule $t \in \delta_{dec}$ rule to obtain C_n.

Incrementing rules. For the remaining case let

$$\Phi_{\mathrm{inc}} = \bigwedge_{t=((q_0,\ldots,q_i),(q'_0,\ldots,q'_j))\in\delta_{inc}} \mathrm{G}\left(t \rightarrow \mathsf{copyBut}_j(q_0,\ldots,q_i) \wedge \mathsf{zero}_i\right) \tag{20}$$

where $i < j$,

$$\mathsf{zero}_j = (\mathrm{X}\,\neg\$)\,\mathrm{U}(\checkmark_j \wedge \bigwedge_{j<\ell\leq k} -_\ell)$$

asserts that the new branch that is created by an incrementing rule $t \in \delta_{inc}$ does not contain more states than explicitly specified in t. Recall that Equation 16 ensures that the propositions $-_\ell$ for $\ell \in [k]$ can only appear if there are no actual states below level $\ell - 1$ in the tree structure of the corresponding configuration.

Finally, the formula

$$\mathsf{copyBut}_j(q_0,\ldots,q_i) = \left((\mathsf{even} \wedge \neg\checkmark_j) \rightarrow \left(\begin{array}{c} \mathsf{link}_k(q_0) \;\wedge \big(\bigwedge_{\ell\in[i]} (\checkmark_\ell \rightarrow \mathsf{link}_k(q_\ell))\big) \\ \wedge \bigwedge_{\ell\in[k],q\in Q_\ell} (\neg\checkmark_\ell \wedge q) \rightarrow \mathsf{link}_k(q) \end{array}\right)\right) \mathrm{U}\,\$.$$

is similar to the copy formula above but omits to copy the particular branch that was created by the incrementing rule.