UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Extending Freeze-LTL on Multi-Attributed Data Words with Quantifiers

Erweiterung von Freeze-LTL auf mehrfach attributierten Datenwörtern um Quantoren

Bachelorarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Anton Pirogov

ausgegeben und betreut von
Prof. Dr. Martin Leucker

mit Unterstützung von
Normann Decker

Lübeck, den 15. Juli 2015

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne die Benutzung anderer als der angegebenen Hilfsmittel selbständig verfasst habe; die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe des Literaturzitats gekennzeichnet.

(Anton Pirogov)
Lübeck, den 15. Juli 2015

Abstract The family of temporal logics is useful to precisely specify the behaviour of systems with relation to time. For example, that some events have to follow each other, always or never happen, etc. They are often used in the context of software verification to test programs for correct behaviour, because not all errors can be caught at compile time or by unit tests. From corresponding logical formulae one can for example construct efficient monitors that evaluate the execution of programs and can detect whether there are deviations from the specification.

These logics differ in expressivity, depending on the allowed operators and quantifiers. So the ambition is to construct logics that allow for expression of interesting properties while keeping the verification or at least decidability feasible. The logic Freeze-LTL extends LTL and allows for storing a value in a register and checking for equality at some later point in time. The main contribution of this thesis is the definition of a new logic and the proof that decidability is preserved. This logic combines two different extensions of Freeze-LTL from the literature – on the one hand, the possibility to work with multiple ordered attributes, if they exhibit a kind of hierarchial structure, on the other hand, the possibility to quantify over previous values or guess some future value for which some property must hold.

Kurzfassung Die Familie der linearen Temporallogiken eignet sich hervorragend dazu, das Verhalten von Systemen in Abhängigkeit von Zeit präzise zu beschreiben bzw. spezifizieren. Zum Beispiel, dass bestimmte Ereignisse aufeinander folgen müssen, immer oder nie eintreten, etc. Sie werden oft im Kontext der Softwareverifikation verwendet, um Programme auf korrektes Verhalten zu prüfen, da sich nicht alle Fehler bereits zur Kompilierzeit oder durch statische Unit-Tests erkennen lassen. Aus entsprechenden logischen Formeln können dann zum Beispiel effiziente Monitore generiert werden, welche die beschriebenen Programme zur Laufzeit überwachen und feststellen können, ob es Abweichungen von der Spezifikation gibt.

Diese Logiken können je nach erlaubten Operatoren und Quantoren unterschiedlich ausdrucksstark sein. Es gilt also, möglichst Logiken zu konstruieren, welche einem erlauben, interessante Eigenschaften auszudrücken und dabei möglichst einfach überprüfbar oder überhaupt entscheidbar bleiben. Die auf LTL basierende Logik Freeze-LTL erlaubt das Speichern eines Wertes in einem Register und eine spätere Prüfung auf Gleichheit. In dieser Arbeit wird eine neue Logik definiert und es wird gezeigt, dass Entscheidbarkeit in der resultierenden Logik gewahrt bleibt. Diese Logik kombiniert zwei unterschiedliche Erweiterungen von Freeze-LTL – einerseits die Möglichkeit, mit mehreren geordneten Attributen zu arbeiten, solange diese eine gewisse hierarchische Struktur aufweisen, andererseits die Möglichkeit, über vergangene Werte zu quantifizieren oder einen zukünftigen Wert zu raten, für den oder die dann etwas gelten soll.

Contents

1 Introduction

1.1 Background and related work

Our modern world is increasingly dependent on the correct functioning of different technical systems – from electronic household devices to factories producing goods and space stations floting in orbit. As these systems are becoming more and more complex, there is a need for tools that allow to express how we expect these systems to behave and also supervise and evaluate their work. The field of formal verification is researching and providing the world with exactly such tools, enabling us to model different systems, specify the expected behaviour and verify that the actual behaviour does indeed correspond to our model.

As such systems are mostly implemented with computers that run according domain specific software, a big part of system verification is especially software verification, which includes two approaches complementing each other – on the one hand static methods based on analysis of the source code prior to its deployment, e.g. by usage of strong type systems, consequent unit testing, etc., on the other hand more dynamic approaches, e.g. monitoring the execution of a program to detect deviations from the expected behaviour, either to resolve problems on-the-fly, or analyse and fix them later.

Most methods of system specification employ some kind of formal language, as natural language, with its ambiguities and its verbosity, is obviously not adequate for this role. While *first-order logic*, which is used ubiquitously in mathematics, proved itself to be a good language to express different notions in a concise and exact way, it is not a good fit in this context, for the simple reason that automated verification of formulae in first-order logic is not possible in general. Basic propositional logic, on the other hand, may be too weak to express many desirable properties or just too cumbersome to use.

What all systems have in common is that they all compute or do something, which implies some form of progress or behaviour and thereby a notion of time – only in the context of

time it is possible to talk about change and hence about behaviour or progress. Therefore it is helpful, if the language used to express properties of a system includes facilities to express change over time.

Linear Temporal Logic (LTL) is a formal logic that allows the expression of propositions with respect to a notion of time. Pnueli (1977) was the first one proposing to use it for formal verification of computer programs.

LTL is syntactially an extension of classical propositional logic (i.e. formulae with just $\vee, \wedge, \neg$ etc.) that includes time-related operators. LTL formulae are usually checked on sequences of sets of propositions that encode different events. The time operators can refer to different positions in a sequence, for example the operator $\mathbf{X}$ ("*next*") expresses that something must be true in the next position of a given sequence and $\varphi \mathbf{U} \psi$ ("*until*") expresses that ψ must be true at some point in the future (i.e., later position in the sequence) and until then φ must be true. Other useful time-related operators can be defined using these two, commonly used are e.g. $\mathbf{F}$ ("*finally*"), expressing that something must be true at some point in the future and $\mathbf{G}$ ("*globally*"), expressing that something must be true forever.

Today many LTL variants and extensions exist, each with a different set of permitted operators and different semantics. For example, LTL can contain different operators to refer to events in the future and in the past, so the operators may have past-time counterparts like $\mathbf{X}^{-1}$ to refer to the *previous* position, etc. Also, different LTL variants are designed for finite and (theoretically) infinite sequences, because different semantics are needed as a consequence of dealing with infinity. Decidability results for variants of LTL logics then depend on the different characteristics of the logic, e.g. among other things, the presence or absence of past-time operators.

While sequences give events or points in time an ordering, they normally do not allow to quantify the time that passed between them. Therefore Henzinger (1990) introduced the *freeze quantifier* as a means to store a kind of timestamp assigned to each position in some sequence and later compare it to a different timestamp. Based on this idea the *Timed Propositional Temporal Logic (TPTL)* has been introduced over *timed state sequences*, sequences associated with a monotonically growing value representing time. The idea of such a freeze quantifier was then investigated in different other contexts, where the values to be stored and compared not necessarily represent points in time and where models obey completely different constraints than the ones assumed in TPTL.

One such development is *Freeze LTL*, which is an extension of LTL that considers sequences called *data words* which additionally contain a single arbitrary data value from an infinite data domain in each position. It adds the *freeze* and *check* operators to LTL, allowing to store a value in a *register* (an abstract memory cell) and compare it with some value contained in a different position. In Demri et al. (2005) it has been shown that this extension is in general not decidable. Especially the addition of more than one register to store values or the addition of past-time operators causes undecidability. In Demri and Lazic (2009) it has been shown that $\mathsf{LTL}^{\downarrow}_{1}$ is decidable. This logic is a fragment of the general Freeze LTL defined over data words with only future-time operators, a freeze quantifier with one register and comparison only for equality. The proof of decidability goes via translation of the logic to *Alternating Register Automata (ARA)*, which then are translated to *counter automata* (Demri et al. (2008)) that are decidable.

In Figueira (2012) ARA are further investigated and a different, more simple and direct proof of decidability is given that uses a technique based on *well-structured transition systems (WSTS)* (Finkel and Schnoebelen (2001)) and requires no translation to a different automata model. Further, using this technique it is shown that a strict extension of $\mathsf{LTL}^{\downarrow}_{1}$ with two new quantifiers $\exists_{\geq}$ and $\forall_{\leq}$ is also decidable, based on a translation of formulae to corresponding ARA. The $\exists_{\geq}$ quantifier allows the expression of statements that refer to some data value that may come in the future, basically guessing some value non-deterministically and then verifying the formula for the chosen value. The $\forall_{\leq}$ quantifier allows for looking at past values and verify a formula for all of them. It is also shown that the dual operators $\exists_{\leq}$ and $\forall_{\geq}$ are not decidable.

A different extension based on $\mathsf{LTL}^{\downarrow}_{1}$ is $\mathsf{LTL}^{\downarrow}_{A}$ introduced by Decker and Thoma (2015). It still has just one register, but increases the expressivity by a semantics that gives more structure to the value associated with a letter. This is done by assuming a set of attributes that has an appropriate ordering of the elements and encoding the values currently assigned to these attributes in each position of a word. The possibility to compare subsets of these attributes is limited, though. Also a restriction to *tree-quasi-ordered* attribute sets (which basically means that the attributes have to depend on each other in a tree-like parent-children-relationship) is necessary, as in in Decker and Thoma (2015) it is shown that $\mathsf{LTL}^{\downarrow}_{A}$ is decidable if and only if A is a tree-quasi-ordering. This restricted, decidable logic is called $\mathsf{LTL}^{\downarrow}_{tqo}$.

A possible example application presented in Decker and Thoma (2015) is the verification of

resource acquiration by processes - considering a model with the events lock, unlock, use and halt that may refer to some *process id* (pid) and *resource id* (res), we can write a formula like

$$\mathbf{G}(\mathsf{lock} \Rightarrow \downarrow^{\mathsf{pid}} ((\mathsf{use} \wedge \uparrow^{\mathsf{res}} \Rightarrow \uparrow^{\mathsf{pid}}) \wedge \neg \mathsf{halt}) \mathbf{U} (\mathsf{unlock} \wedge \uparrow^{\mathsf{pid}}))$$

encoding the statement that whenever a resource is locked, at some time it is released again and until that happens the program is not terminated and the resource is not used by any other process. As a resource can only be locked by one process at the same time, we can assume that the events concerning a resource are always dependent on the current process (if any) holding that resource.

The logic defined in this thesis combines those two different decidable extensions to give the possibility to express formulae that can store and check multiple attributes and also to quantify over the values in different positions. The definition of the syntax also will make sure that the $\exists_{\geq}$ and $\forall_{\leq}$ operators can not be negated, as the negation would result in the dual operators that are known to be undecidable. While the proof in Decker and Thoma (2015) uses the connection found between logics on data words and *counter systems* (Demri et al. (2013)) to obtain decidability and complexity results, here an adaptation of alternative proofs based on a generalisation of ARA as given in Figueira (2012) is used to show decidability of the logic.

1.2 Contribution of this thesis

Based on the work in Decker and Thoma (2015) and Figueira (2012), in this thesis the logic $\mathsf{LTL}_A^{\downarrow}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall_{\leq}^{x}]$ is defined, which extends $\mathsf{LTL}_A^{\downarrow}[\mathbf{X}, \mathbf{U}]$ with the quantifiers $\exists_{\geq}$ and $\forall_{\leq}^{x}$. Further, a corresponding automata model based on ARA and alternative decidability proofs for $\mathsf{LTL}_A^{\downarrow}[\mathbf{X}, \mathbf{U}]$ are presented and extended to show decidability of the new logic over tree-quasi-ordered attribute sets.

2 Definition of the logic

In this chapter I will first define the necessary structures and vocabulary, providing examples for clarity, and then define the logic $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$, based on the logic $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}]$ from Decker and Thoma (2015) and incorporating the operators $\exists_{\geq}$ and $\forall^{x}_{\leq}$ suggested in Figueira (2012).

2.1 Preliminaries

Definition 2.1
Let $\mathbb{N} := \{1, 2, 3, \ldots\}$ *denote the infinite set of positive integer numbers,* $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$ *the infinite set of non-negative integer numbers and* $[n] := \{1, \ldots, n\}, n \in \mathbb{N}$ *the set of numbers from 1 to* n*.*

Definition 2.2
Let $\mathcal{P}(M)$ *be the set of all subsets of a set M and* $\mathcal{P}_{<\infty}(M)$ *the set of finite subsets of M.*

Definition 2.3
Let $f : A \to B$ *be a function. Then* $\mathsf{dom}(f) := A$ *is the* domain, $\mathsf{cod}(f) := B$ *the* codomain *and* $\mathsf{img}(f) := \{f(a) \mid a \in A\}$ *the* image *of the function* f*. Further* B^A *denotes the set of all functions from* A *to* B*.*

Definition 2.4
Let $f : A \to B$ *be a function and* $A' \subseteq A$ *a subset of the domain. Then* $f|_{A'}$ *is the* restriction $f' : A' \to B$ *of* f *with* $f'(a) = f(a)$ *for all* $a \in A'$*. The set of restrictions of functions from* A *to* B *is denoted by* $B^A_{\perp} := \bigcup_{A' \subseteq A} B^{A'}$*.*

Definition 2.5 (Quasi-Orderings)
Let M *be a set and* $\preccurlyeq$ *a reflexive and transitive relation. Then* $(M, \preccurlyeq)$ *is called* quasi-ordering.

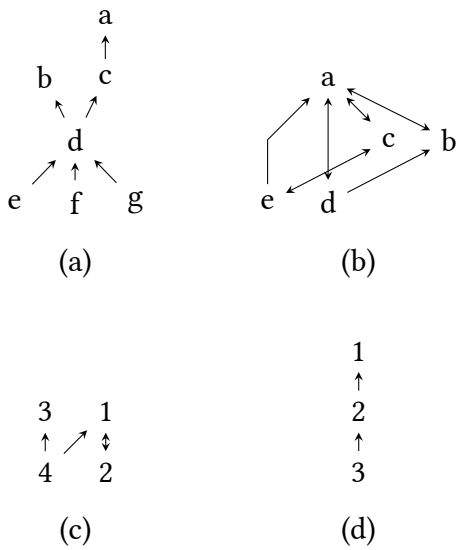


Figure 2.1: Examples of reflexive and transitive orderings that are:
(a) partial (b) total (c) non-partial and non-total (d) linear

- $\mathsf{cl}(m) := \{m' \in M \mid m' \preccurlyeq m\}$ *is called the* downward-closure *of* $m \in M$.
- $(M, \preccurlyeq)$ *is called* total ordering*, if for all* $m, m' \in M$ *either* $m \preccurlyeq m'$ *or* $m' \preccurlyeq m$.
- $(M, \preccurlyeq)$ *is called* partial ordering*, if* $\preccurlyeq$ *is also antisymmetric.*
- *A total and partial ordering is called* linear ordering*. When not stated otherwise, let* $[k], k \in \mathbb{N}$ *denote the linear ordering* $([k], \leq)$ *of the first* k *natural numbers.*

Definition 2.6 (Graph representation of QO)
Let $(M, \preccurlyeq)$ *be a quasi-ordering. The directed graph* $G = (V, E)$ *with* $V = M$ *and* $E = \{(m, m') \mid m' \preccurlyeq m\}$ *is the graph of* $(M, \preccurlyeq)$*. The subgraph induced by the set of vertices reachable from* $m \in M$ *is equivalent to the downward-closure* $\mathsf{cl}(m)$.

Reflexive and transitive edges are implied and may be omitted in the figures for better reading.

Conversely, given a directed graph $G = (V, E)$*, the induced quasi-ordering* $(V, \preccurlyeq)$ *is constructed with* $\preccurlyeq := \{(y, x) \mid x, y \in V : (x, y) \in E^*\}$*,* E^* *denoting the edges of the reflexive and transitive closure of the graph* G.

In figure 2.1 you can see examples for different types of orderings represented as graphs with edges pointing to the direction of the smaller element, e.g. if $a \preccurlyeq c$, there is an edge going from c to a. If the direction of the edges seems odd, consider that it leads to a

natural representation of downward-closures. Reflexive and transitive edges are omitted, as the graph would soon look very confusing. Whenever there is a path in the graph, you can assume an invisible direct edge between the endpoints, so the underlying ordering can be thought of as the *reflexive and transitive closure* of the depicted graph. This graph notation will be used in multiple examples depicting some kind of ordering throughout the thesis.

The ordering (a) in figure 2.1 is partial, as it has no symmetric edges, but not total, because e.g. g and f can not be compared. Ordering (b) is total, as there is an (implied) edge between all elements, but not partial, as there are multiple two-way edges, meaning that e.g. $a \preccurlyeq c$ and also $c \preccurlyeq a$. In (c) you see a quasi-ordering that is neither partial nor total, while (d) is an example for an ordering which is both partial and total, resulting in the only possible structure that can fulfil both properties, a linear chain of elements (thus the name *linear ordering*).

Definition 2.7 (Tree-Quasi-Orderings)
Let $(A, \preccurlyeq)$ *be a quasi-ordering.*

- *If the downward-closure* $A_x := \mathsf{cl}(x)$ *of each element* $x \in A$ *is total, i.e.* $(A_x, \preccurlyeq \cap (A_x \times A_x))$ *is a total quasi-ordering, then* $(A, \preccurlyeq)$ *is a* tree-quasi-ordering.
- *If all downward-closures of a tree-quasi-ordering are linear, the tree-quasi-ordering is just called* tree ordering.
- *The minimal elements of a tree ordering are called* roots*, the maximal elements are called* leaves *and the downward-closures can be called* paths.
- *The* depth *(or* height*)* $\mathsf{ht}(A)$ *of a tree ordering* A *is defined as the maximal length of strictly increasing sequences* $x_1 \prec x_2 \ldots \prec x_k$ *of attributes in* A*, the depth* $\mathsf{ht}(x) = |\mathsf{cl}(x)|$ *of an element* $x \in A$ *is the length of the unique linear path to a root.*

In figure 2.2 you can see a tree-quasi-ordering that is not a tree ordering, as some downward closures are not linear, e.g. in the downward closure $\mathsf{cl}(x_3)$ you can see that $x_2 \preccurlyeq x_3$ and $x_3 \preccurlyeq x_2$, making $\mathsf{cl}(x_3)$ not a partial ordering. Later in this chapter we will see a simple method how to get a tree ordering from a tree-quasi-ordering by collapsing the strongly connected components (highlighted in gray). The root is x_1, the leaves are x_3 and x_8 and the depth is 4, because a strictly increasing sequence with maximal length is $x_1 \prec x_4 \prec x_5 \prec x_8$.

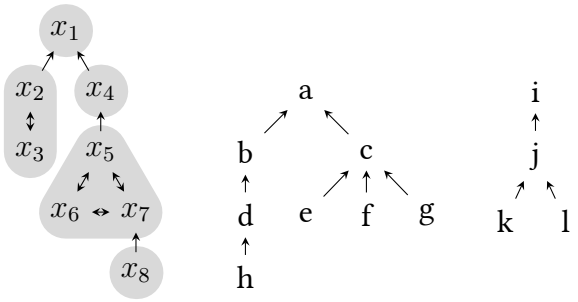


Figure 2.2: A tree-quasi-ordering and a tree ordering

On the right you can see a tree ordering, as all downward-closures are linear. Furthermore, it is an example for a tree ordering consisting of multiple sub-trees – tree orderings can also be forests. The roots are a and i, while the leaves are h, e, f, g, k and l. The depth of this tree ordering is 4 because of the sequence $a \prec b \prec d \prec h$.

Definition 2.8 (Data Words)
Let Σ be a finite alphabet, Δ an infinite domain of data values *and A a finite set of* attributes. *Then the finite sequence $w = (a_1, \mathbf{d}_1)(a_2, \mathbf{d}_2) \cdots (a_n, \mathbf{d}_n) \in (\Sigma \times \Delta^A)^+$ is an* A-attributed data word *with length $|w| = n$, consisting of tuples of* letters $a_i \in \Sigma$ *and* data valuations $\mathbf{d}_i \in \Delta^A$ *which map each attribute to some data value. We use $w_i, i \in [n]$ to denote the letter at the i-th position $(a_i, \mathbf{d}_i)$ of the word.*

If $A = [k]$, the valuations may be called vectors *and represented as k-tuples $(x_1, \ldots, x_k)$ with $x_i \in \Delta$ so that $\mathbf{d}(i) = x_i$ for all $i \in [k]$. If A is a different linear ordering with k elements, it can be treated like $[k]$, identifying each element $a \in A$ with the unique size of its downward-closure $|\mathsf{cl}(a)|$, which is an element of $[k]$.*

In figure 2.3 you can see two examples for data words with a common finite alphabet $\Sigma = \{a, b, c\}$ and infinite value domain $\Delta = (\mathbb{N}, =)$, but different sets of attributes. In the first word we have a set of attributes $A_1 = \{x_1, x_2\}$, in the second word we have $A_2 = \{x_1, x_2, x_3, x_4\}$. As you can see, in each position of the word the value assigned to any attribute can change. For example, in the first word in the second position we have a valuation $\mathbf{d}_2$ with $\mathbf{d}_2(x_1) = 5, \mathbf{d}_2(x_2) = 3$, while in the next position we have a valuation $\mathbf{d}_3$ with $\mathbf{d}_3(x_1) = 8, \mathbf{d}_3(x_2) = 3$.

A noteworthy aspect you can see is that in both examples the attributes exhibit a quasi-ordered structure – in the first word the set A_1 has the ordering relation $\preccurlyeq_1 = \{(x_1, x_2)\}$,

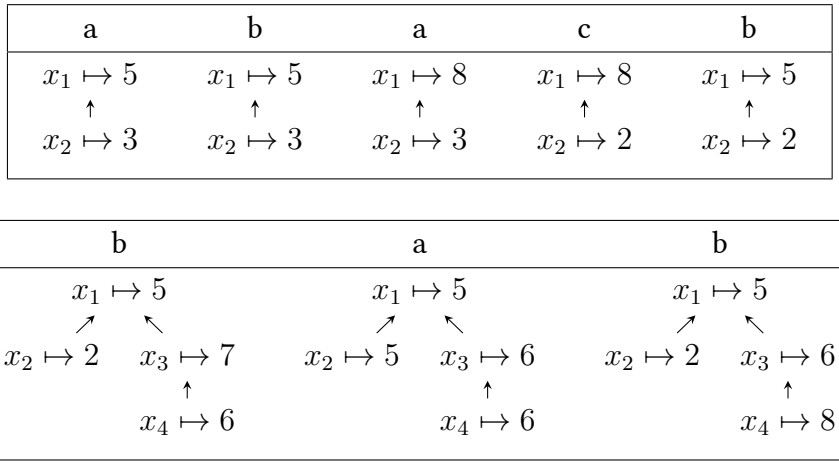


Figure 2.3: Examples for data words

making $(A_1, \preccurlyeq_1)$ a linear ordering. In the second word the set A_2 has the relation $\preccurlyeq_2 = \{(x_1, x_2), (x_1, x_3), (x_1, x_4), (x_3, x_4)\}$, making $(A_2, \preccurlyeq_2)$ an example for a tree ordering. Exactly such types of orderings for multi-attributed data words are the kind we are going to work with.

Unlike in the first word of figure 2.3, for linear-ordered attribute sets that are isomorphic to $([k], \leq)$ the attribute names in all following examples will be omitted and just addressed with a number, i.e. the smallest attribute is just called 1, the second-smallest is called 2 and so on, up to the maximal element addressed with $k, k \in \mathbb{N}$. The graph representation of a linear-ordered data valuation will then just contain the values associated with the corresponding attribute, without mentioning the actual attribute.

Definition 2.9 (Data Valuations)
Let A be a quasi-ordered set and $\mathbf{d}, \mathbf{d}' \in \Delta_\perp^A$ some partial data valuations. $\mathbf{d}$ *is called* equivalent *to* $\mathbf{d}'$ *(written:* $\mathbf{d} \simeq \mathbf{d}'$*) if and only if there is a bijection* $h : \mathsf{dom}(\mathbf{d}) \to \mathsf{dom}(\mathbf{d}')$ *so that for all* $a, a' \in \mathsf{dom}(\mathbf{d}) : a \preccurlyeq a' \Leftrightarrow h(a) \preccurlyeq h(a') \wedge \mathbf{d}(a) = \mathbf{d}'(h(a))$.

In figure 2.4 you can see the same attribute set multiple times, each time with different subsets highlighted, denoting the attributes which are in the domain of the corresponding partial data valuation. The previous definition tells us, that two partial valuations are equivalent if the domain and ordering of both is isomorphic and the corresponding attributes hold the same values.

Although $\mathbf{d}_1$ and $\mathbf{d}_2$ are isomorphic – you can map x_2 to x_3 and x_4 to x_5 and have a linear

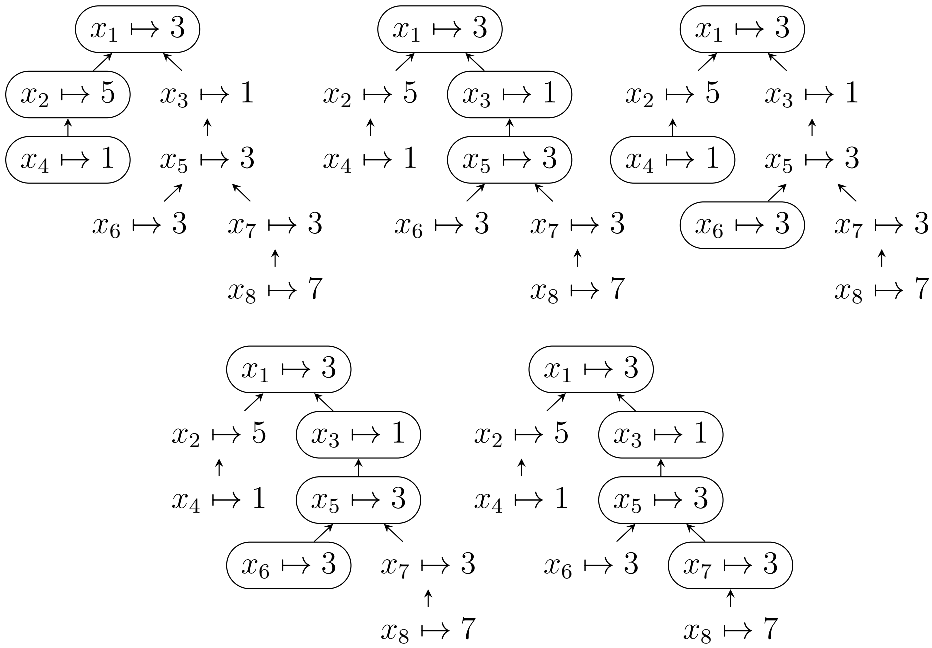


Figure 2.4: Examples of different partial data valuations ($\mathbf{d}_1, \ldots, \mathbf{d}_5$) of the same attribute set

ordering of size 3 in both cases – the values do not match, as $5 = \mathbf{d}_1(x_2) \neq \mathbf{d}_2(x_3) = 1$ and $1 = \mathbf{d}_1(x_4) \neq \mathbf{d}_2(x_5) = 3$, so $\mathbf{d}_1 \not\simeq \mathbf{d}_2$.

If we compare $\mathbf{d}_2$ and $\mathbf{d}_3$, we see that although the number of attributes in both subsets is the same and we could map attributes with identical values to each other, this does not suffice. If we would map x_5 to x_6 and x_3 to x_4, we still would have the problem that $x_3 \preccurlyeq x_5$, but $x_4 \not\preccurlyeq x_6$, so the relation is not preserved in the mapping and therefore $\mathbf{d}_2 \not\simeq \mathbf{d}_3$.

When comparing e.g. $\mathbf{d}_2$ and $\mathbf{d}_4$ we already can see just from the structure that $\mathbf{d}_2 \not\simeq \mathbf{d}_4$, because we can not create a bijection between two subsets with different size.

Finally, for a positive example, we can conclude that $\mathbf{d}_4 \simeq \mathbf{d}_5$, because we can map x_6 to x_7 and have in both cases a linear order of size 4 and also the corresponding values are the same – trivially for the shared attributes and because $\mathbf{d}_4(x_6) = \mathbf{d}_5(x_7) = 3$.

2.2 Syntax and semantics

Now we have all the pieces that we need to construct our logic:

Definition 2.10 (Syntax)
Let A be some finite set of attributes and AP a finite set of atomic propositions, $p \in AP$ and $x \in A$.

The following grammar describes syntactically valid formulae in $\mathsf{LTL}_A^{\downarrow}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall_{\leq}^{x}]$ *:*

$$
\begin{array}{rcccccccc}
\varphi & ::= & p & | & \neg\psi & | & \varphi \wedge \varphi & | & \varphi \vee \varphi \\
 & | & \mathbf{X}\varphi & | & \varphi \mathbf{U} \varphi & | & \downarrow^x \varphi & | & \uparrow^x \\
 & | & & \exists_{\geq} \varphi & & | & & \forall^x_{\leq, \psi} \varphi & \\
\psi & ::= & p & | & \neg\psi & | & \psi \wedge \psi & | & \psi \vee \psi \\
 & | & \mathbf{X}\psi & | & \psi \mathbf{U} \psi & | & \downarrow^x \psi & | & \uparrow^x
\end{array}
$$

Parentheses may be used freely to represent the structure of a formula.

Definition 2.11 (Syntactic sugar)
The following additional operators are regarded as syntactic sugar and can be used when appropriate:

$$
\begin{array}{rclcrcl}
\mathsf{true} & := & p \vee \neg p \quad p \in AP & & \mathsf{false} & := & \neg\, \mathsf{true} \\
\varphi \Rightarrow \psi & := & \neg\varphi \vee \psi & & \varphi \Leftrightarrow \psi & := & (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi) \\
\mathbf{F}\varphi & := & \mathsf{true}\, \mathbf{U} \varphi & & \mathbf{G}\varphi & := & \neg\mathbf{F}\neg\varphi \\
\overline{\mathbf{X}}\varphi & := & \neg\mathbf{X}\neg\varphi & & \varphi\, \mathbf{R} \psi & := & \neg(\neg\varphi \mathbf{U} \neg\psi)
\end{array}
$$

The definition 2.10 provides a minimal set of common LTL operators, as you can get most familiar operators from well-known equivalences defined in 2.11, which use the minimal set of operations to define everything else. By choosing this path, the core syntax is kept small and easy to handle, while still allowing us to use convenience operators to make formulae more readable.

Definition 2.12 (Semantics)
Let $(A, \preccurlyeq)$ be a finite tree-quasi-ordered set of attributes, AP a finite set of atomic propositions and $\Sigma = \mathcal{P}(AP)$ a finite alphabet encoding subsets of AP, $w = (a_1, \mathbf{d}_1) \dots (a_n, \mathbf{d}_n) \in$

$(\Sigma \times \Delta^A)^+$ *a data word of length* $n \geq 1$, $\mathbf{d} \in \Delta_\perp^A$ *a partial data valuation,* $i \in [n]$ *a position in* w *and* $x \in A$.

The following satisfaction relation inductively defines the semantics of $\mathsf{LTL}_A^{\downarrow}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall_{\leq}^x]$ *:*

$$
\begin{aligned}
(w,i,\mathbf{d}) \models\ & p && :\Leftrightarrow\ p \in a_i \\
(w,i,\mathbf{d}) \models\ & \neg\varphi && :\Leftrightarrow\ (w,i,\mathbf{d}) \not\models \varphi \\
(w,i,\mathbf{d}) \models\ & \varphi \wedge \psi && :\Leftrightarrow\ (w,i,\mathbf{d}) \models \varphi \text{ and } (w,i,\mathbf{d}) \models \psi \\
(w,i,\mathbf{d}) \models\ & \varphi \vee \psi && :\Leftrightarrow\ (w,i,\mathbf{d}) \models \varphi \text{ or } (w,i,\mathbf{d}) \models \psi \\
(w,i,\mathbf{d}) \models\ & \mathbf{X}\varphi && :\Leftrightarrow\ i+1 \leq n \text{ and } (w,i+1,\mathbf{d}) \models \varphi \\
(w,i,\mathbf{d}) \models\ & \varphi \mathbf{U} \psi && :\Leftrightarrow\ \exists_{i \leq k \leq n} : (w,k,\mathbf{d}) \models \psi \text{ and } \forall_{i \leq j < k} : (w,j,\mathbf{d}) \models \varphi \\
(w,i,\mathbf{d}) \models\ & \downarrow^x \varphi && :\Leftrightarrow\ (w,i,\mathbf{d}_i|_{\mathsf{cl}(x)}) \models \varphi \\
(w,i,\mathbf{d}) \models\ & \uparrow^x \varphi && :\Leftrightarrow\ \exists_{y \in A} : \mathbf{d}_i|_{\mathsf{cl}(x)} \simeq \mathbf{d}|_{\mathsf{cl}(y)} \\
(w,i,\mathbf{d}) \models\ & \exists_{\geq} \varphi && :\Leftrightarrow\ \exists_{\mathbf{d}' \in \Delta^A, x \in A} : (w,i,\mathbf{d}'|_{\mathsf{cl}(x)}) \models \varphi \\
(w,i,\mathbf{d}) \models\ & \forall_{\leq,\psi}^x \varphi && :\Leftrightarrow\ \forall_{j \leq i} : (w,j,\mathbf{d}_j) \models \psi \Rightarrow (w,i,\mathbf{d}_j|_{\mathsf{cl}(x)}) \models \varphi
\end{aligned}
$$

Notice, that our storing capability is limited – we can not store and compare arbitrary attributes by themselves, but only downward-closures of an attribute, or put in a different way, we can only store complete paths from an attribute to a root element and compare them to other paths. So we can only compare attributes together with all other "ancestor" attributes they depend on.

We gain a bit of flexibility from the definition of the check-operator $\uparrow^x$, as we can also compare non-isomorphic paths. This is possible, if we previously stored an attribute with a downward-closure with a bigger size and then check an attribute with a downward-closure with a smaller size. Explained visually, we can store an attribute at some deep position in the tree and then check an attribute in a higher position. The check-operator can just ignore the additional values, as it looks for a compatible, isomorphic downward-closure. Obviously this does not work in the other direction, though, as we can not extend our closure afterwards in any meaningful way. So storing a less deep attribute and checking a deeper attribute afterwards will always fail.

For illustration, consider figure 2.5. The formula φ_1 checks that a holds in the first position and at the same time stores the downward-closure of the attribute x_4. In the next

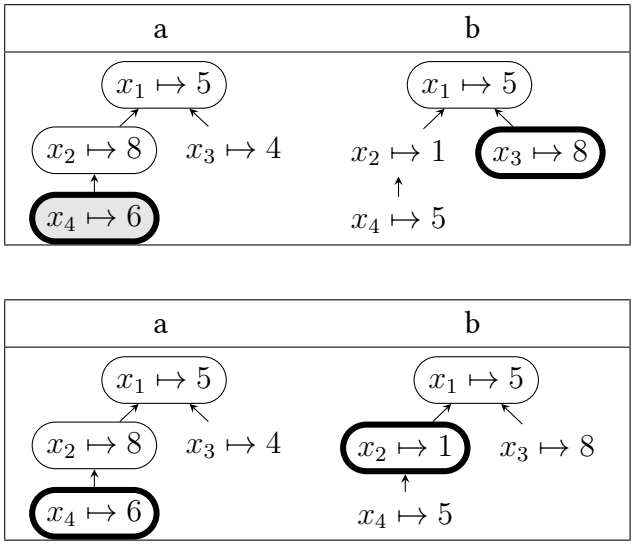


Figure 2.5: An example usage of the "forgetful" check operator

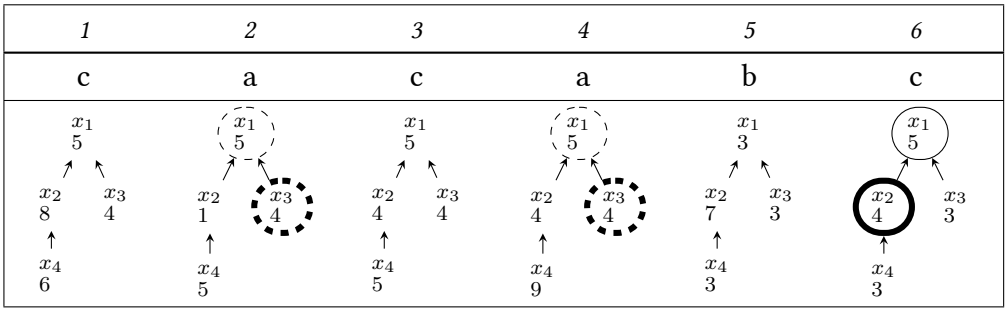


Figure 2.6: An example usage of the $\forall^{x}_{\leq,\psi}$ operator

position it checks that b holds and compares the stored values to the downward-closure of the attribute x_3. The formula is satisfied, as the check restricts the downward-closure of the stored attribute to x_1 and x_2, ignoring x_4 and getting an isomorphic linear ordering that can be compared and indeed has the same values. In φ_2 on the contrary, there is no restriction of the downward-closure of x_4 that has the same values as the downward-closure of x_2, so the formula is not satisfied.

For the sake of simplicity, in all examples the alphabet that is used in data words is assumed to encode sets that contain just a single proposition with the same symbol as the letter, so when you see an a in a data word, it really means the set containing the proposition a, that is $\{a\}$.

b	a	b	a	a	b	a	a
1	1	3	1	3	2	2	1
↑	↑	↑	↑	↑	↑	↑	↑
2	2	2	2	2	3	3	3
↑	↑	↑	↑	↑	↑	↑	↑
3	3	1	3	1	1	1	2

$$\varphi = \exists_{\geq}((b \Rightarrow \neg \uparrow^3)\mathbf{U}(a \wedge \uparrow^3))$$

Figure 2.7: An example usage of the $\exists_{\geq}$ operator

The $\exists_{\geq}$-quantifier allows us to guess and store an arbitrary attribute with an arbitrary data value in the register and check that the following sub-formula is satisfied. This can be used to express the notion that there exists some valuation in the future for which some property holds. The $\forall^x_{\leq,\psi}$-quantifier allows us to express that the following sub-formula should be satisfied for all valuations of an attribute x up to the current position, for all positions where ψ held. The ψ constraint is required for the linearisation which will be described later and allows us to filter the set of positions to be quantified over. If no filtering is desired, the ψ can be just set to true. In that case, the ψ may be omitted completely, so that $\forall^x_{\leq}$ denotes $\forall^x_{\leq,\mathsf{true}}$. Also, for linear orderings the x parameter can be omitted, because there is no ambiguity with regard to the branch to be stored (as there is only one) and the correct depth can be obtained by the automatic restriction of $\uparrow^x$, as described above.

In figure 2.6 you can see the $\forall^x_{\leq,\psi}$ operator in action — the formula checks that finally b holds and then checks, that the data value stored for x_3 in all previous positions where a held is the same as the value stored in x_2 in the next position. For the given data word this is indeed the case — a held in positions 2 and 4, so these positions are quantified over. b holds finally in position 5 and the values of $\mathbf{d}_2$ and $\mathbf{d}_4$ restricted to x_3 are indeed the same as the value $\mathbf{d}_6$ restricted to x_2.

In figure 2.7 the formula says that there exists a position where a holds and that the value of x_3 at that position is never seen at previous positions at which b held. This is a translation of the example given in Figueira (2012) of a property that can not be expressed without the $\exists_{\geq}$ operator. You can verify that the formula is satisfied by observing that the valuation in the last position of the word is never used before. Now it is clear how the logic defined in this chapter works, but we have yet to prove that it is decidable. For this proof we first need the automata model presented in the next chapter.

3 Nested Register Automata

In this chapter I will present the automata model that is used in the next chapter to prove decidability of $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$. The automata are a generalization of *alternating register automata (ARA)* as defined in (Figueira, 2012, Def. 3.1), adjusted to work with ordered data. I will extend this automata model with the two opearations guess and spread from Figueira (2012) that were originally missing in the generalization, but are required for the translation of $\exists_{\geq}$ and $\forall^{x}_{\leq}$ operators in the next chapter.

Next, I will present necessary terminology from the framework of *well-structured transition systems*, which was introduced in Finkel and Schnoebelen (2001) as a generalization of multiple notions, with the aim to simplify and unify the concepts required to prove decidability in different contexts.

Finally I will present a proof by Decker and Thoma [1] that emptiness is decidable for NRA, which is based on this framework. The proof originally did not include the necessary cases for guess and spread, so I will consider these cases where it is relevant, thereby adapting it to NRA that include these two additional operations.

3.1 Definition of NRA

The k-NRA automaton we will define now is a nondeterministic one-way automaton that can be thought of having a set of synchronized threads, working independently, with the restriction that all threads move to the next position of the word simultaniously. Each thread has a register, having the ability to store the data vector of a position and check the saved value for equality at some later point.

We will extend it with the capability to guess an arbitrary data value with guess and use spread to create new threads for each already existing thread in some specific state.

[1] private communication

As our automata will run on finite data words and will be able to see just one position at a time, it is desireble to know when we have reached the end. The following definition offers us a clean way to express this information:

Definition 3.1
Let $w \in (\Sigma \times \Delta^A)^+$ *be a data word with* $|w| = n$. *Let* $\mathsf{type}_w(i) : [n] \to \{\triangleright, \overline{\triangleright}\}$ *be the* word type *of* w, *mapping each position of the word to a symbol indicating whether there is a next position in the word:* $\forall_{i \in [n-1]} : \mathsf{type}_w(i) = \triangleright, \mathsf{type}_w(n) = \overline{\triangleright}$.

Using type we can access this metainformation for some word and position in the states of our automata and $\overline{\triangleright}$ will indicate that we are at the end of the word. In the next chapter we will need this to correctly encode the weak next operator $\overline{\mathbf{X}}$ from some formula into a NRA.

Definition 3.2 (k-NRA)
Let $\Sigma = \mathcal{P}(AP)$ *be a finite alphabet encoding subsets of atomic propositions* AP, Q *a finite set of states,* q_1 *the initial state,* $k \in \mathbb{N}$ *the maximum register depth and* $\delta : Q \to \Phi$ *a transition function from states to expressions defined by the grammar*

$$\varphi := p \mid \overline{p} \mid \odot? \mid \mathsf{store}(q) \mid \mathsf{eq}_i \mid \overline{\mathsf{eq}_i} \mid q \wedge q' \mid q \vee q' \mid \triangleright q \mid \mathsf{guess}(q) \mid \mathsf{spread}(q, q')$$

with $p \in AP, q, q' \in Q, \odot \in \{\triangleright, \overline{\triangleright}\}, i \in [k]$.

The tuple $\mathcal{A} = (\Sigma, k, Q, q_1, \delta)$ *is called an* alternating k-nested register automaton (NRA).

Let $w \in (\Sigma \times \Delta^{[k]})^+, |w| = n$ *be a* $[k]$*-attributed data word of length* $n \geq 1$.

A configuration *of* $\mathcal{A}$ *is a tuple* (i, α, γ, T), *where* $i \in [n]$ *denotes the position in* w, $\alpha = \mathsf{type}_w(i)$ *is the word type of the current position,* $\gamma = w_i$ *is the current input letter and* $T \in \mathcal{P}_{<\infty}(\Delta^{[k]} \times Q)$ *is the set of active* threads, *where a single thread* $(\mathbf{d}, q) \in T$ *consists of a vector of data values* $\mathbf{d}$ *stored in its register and its current state* q. *The set of all configurations of a* k*-NRA is denoted with* $\mathcal{C}^k_{\mathsf{NRA}}$.

A state $q \in Q$ *is called* moving, *if* $\delta(q) = \triangleright q'$ *for some* $q' \in Q$ *and a configuration is called moving, if for all* $(\mathbf{d}, q) \in T$ *the state* q *is moving.*

Let $\rho = (i, \alpha, (a, \mathbf{d}), (\mathbf{d}', q) \cup T)$ *be a configuration. The* non-moving transition relation $\to_\epsilon \subseteq \mathcal{C}^k_{\mathsf{NRA}} \times \mathcal{C}^k_{\mathsf{NRA}}$ *is defined as follows:*

$$
\begin{aligned}
\rho \to_\epsilon \quad & (i, \alpha, (a, \mathbf{d}), \{(\mathbf{d}', q_i)\} \cup T) && :\Leftrightarrow \delta(q) = q_1 \vee q_2, i \in \{1, 2\} \\
\rho \to_\epsilon \quad & (i, \alpha, (a, \mathbf{d}), \{(\mathbf{d}', q_1), (\mathbf{d}', q_2)\} \cup T) && :\Leftrightarrow \delta(q) = q_1 \wedge q_2 \\
\rho \to_\epsilon \quad & (i, \alpha, (a, \mathbf{d}), \{(\mathbf{d}, q')\} \cup T) && :\Leftrightarrow \delta(q) = \mathsf{store}(q') \\
\rho \to_\epsilon \quad & (i, \alpha, (a, \mathbf{d}), T) && :\Leftrightarrow \delta(q) = \mathsf{eq}_i \text{ and } \forall_{1 \leq j \leq i} : \mathbf{d}(j) = \mathbf{d}'(j) \\
\rho \to_\epsilon \quad & (i, \alpha, (a, \mathbf{d}), T) && :\Leftrightarrow \delta(q) = \overline{\mathsf{eq}_i} \text{ and } \exists_{1 \leq j \leq i} : \mathbf{d}(j) \neq \mathbf{d}'(j) \\
\rho \to_\epsilon \quad & (i, \alpha, (a, \mathbf{d}), T) && :\Leftrightarrow \delta(q) = \beta? \text{ and } \alpha = \beta, \quad \beta \in \{\triangleright, \overline{\triangleright}\} \\
\rho \to_\epsilon \quad & (i, \alpha, (a, \mathbf{d}), T) && :\Leftrightarrow \delta(q) = p \text{ and } p \in a \\
\rho \to_\epsilon \quad & (i, \alpha, (a, \mathbf{d}), T) && :\Leftrightarrow \delta(q) = \overline{p} \text{ and } p \notin a \\
\rho \to_\epsilon \quad & (i, \alpha, (a, \mathbf{d}), \{(\mathbf{e}, q')\} \cup T) && :\Leftrightarrow \delta(q) = \mathsf{guess}(q'), \mathbf{e} \in \Delta^{[k]} \\
\rho \to_\epsilon \quad & (i, \alpha, (a, \mathbf{d}), \{(\mathbf{d}'', q_1) \mid (\mathbf{d}'', q_2) \in T\} \cup T) && :\Leftrightarrow \delta(q) = \mathsf{spread}(q_2, q_1) \text{ and } (*)
\end{aligned}
$$

$(*)$: *For* $\mathsf{spread}(\ldots)$ *it is demanded, that all other possible* $\to_\epsilon$ *transitions are already executed, in order to take into account all new data values that were possibly introduced in these transitions.*

The moving transition relation $\to_\triangleright$ *is defined as:*

$$(i, \triangleright, \gamma, T) \to_\triangleright (i + 1, \alpha', \gamma', T')$$

with $\alpha' = \mathsf{type}_w(i + 1), \gamma' = w_{i+1}, T' = \{(\mathbf{d}, q') \mid (\mathbf{d}, q) \in T, \delta(q) = \triangleright q'\}$ *iff the configuration* $(i, \triangleright, \gamma, T)$ *is moving.*

Finally, we define the transition relation *between the configurations as* $\to := \to_\epsilon \cup \to_\triangleright$.

A run *on a data word* $w \in (\Sigma \times \Delta^{[k]})^+$ *is a non-empty sequence* $\mathcal{C}_1 \to \ldots \to \mathcal{C}_n$ *with* $\mathcal{C}_1 = (1, \alpha_1, \gamma_1, \{\mathbf{d}_1, q_1\})$. *A run is* accepting, *iff* $\mathcal{C}_n = (i, \alpha, \gamma, \emptyset)$ *contains an empty set of threads. If for an automaton* $\mathcal{A}$ *there is some word* $w \in (\Sigma \times \Delta^{[k]})^*$ *for which* $\mathcal{A}$ *has an accepting run, we say that* $\mathcal{A}$ *is* non-empty.

You can see in the definition of the transition relation that the different kinds of expressions can be classified in different groups. Some of them *introduce* new threads, like $\wedge$ and spread. For example, in the case of $\wedge$ both subformulae must be satisfied, so two threads are created to take care of each.

Some expressions *modify* a thread in some way, e.g. in the case of $\vee$, the next state is nondeterministically chosen from two given possibilities, in the case of guess the thread

$$\mathcal{A} = (\{a, b\}, 3, \{q_1, \ldots, q_8\}, q_1, \delta)$$

q_i	1	2	3	4	5	6	7	8
$\delta(q_i)$	$q_2 \wedge q_3$	a	$\triangleright q_4$	$\mathsf{store}(q_5)$	$\triangleright q_6$	$q_7 \wedge q_8$	b	eq_2

$w =$

a	a	b
4	5	5
↑	↑	↑
5	1	1
↑	↑	↑
7	2	9

Figure 3.1: Example of a 3-NRA and an accepted word

nondeterministically saves a new vector in the register and continues the evaluation. So there can only be an accepting run over a word, if one of these choices leads to success, allowing these threads to terminate.

Some expressions *eliminate* threads. This happens by definition only, if the according expression is successful, e.g. a thread with eq_i only can be eliminated, if the comparison of the data vector stored in the thread and the vector at the current position of the word is successful. If this is not the case, this thread can not be removed, therefore the configuration can never become moving and the automaton can not continue to read the word.

A run always begins from an initial thread with an initial state at the beginning of the word. In each position threads are introduced, modified or eliminated, first evaluating all threads of non-moving expressions except spread, then the spread operation, which depends on the other threads and therefore waits for them to include all candidates. Finally, when all threads in the current position are at a $\triangleright$-expression (the configuration is moving), the threads move on to the next position of the word. This goes on until no more transition is possible.

Directly from this transition semantics naturally comes the definition of an accepting run – if a run over a word stops, because there are no more threads, it means that we successfully checked all properties encoded in the states, ultimately leading to the termination of all threads. If a run ends with some threads still present in the configuration, it means that some assumption did not hold for the word, leaving the automaton in a stuck state, unable to continue because of no more applicable transition.

In figure 3.1 you can see an NRA with 8 states depicted in tabular form. When the automaton starts the run on w, it is in the configuration $\mathcal{C}_1 = (1, \triangleright, (a, (4, 5, 7)), \{((4, 5, 7), q_1)\})$. Due to the associated expression $\delta(q_1)$ the initial thread gets replaced with two threads $\{((4, 5, 7), q_2), ((4, 5, 7), q_3)\}$. The expression of q_2 checks that a holds, which is true, so

this thread gets eliminated. Now the configuration is moving and so the expression of q_3 moves the thread to the next position of the word, changing to state q_4. Next, due to the store the thread saves the vector $(5, 1, 2)$ of the current position into the register and changes state to q_5. Again, we are in a moving state and as this is the only action left to do, the thread moves to the next position and changes into the state q_6. Now the thread again gets replaced with two new threads $\{((5, 1, 2), q_7), ((5, 1, 2), q_8)\}$. The thread in state q_7 verifies that b holds, becoming successfully eliminated. The last thread that is left is in state q_8 and also gets eliminated after successfully verifying that the first two values of the vector $(5, 1, 9)$ at the current position are equal to the first two values of the stored vector. Now we are left with no threads, therefore the automaton halts, accepting the word w.

As you can easily see, the automaton discussed above represents the $\mathsf{LTL}^{\downarrow}_{[k]}$ formula $\varphi = a \wedge \mathbf{X} \downarrow^2 \mathbf{X}(b \wedge \uparrow^2)$. In the next chapter a general way will be presented to translate arbitrary $\mathsf{LTL}^{\downarrow}_{[k]}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^x_{\leq}]$ formulae into k-NRA, so that the set of accepted words of the automaton exactly characterizes the models of the underlying formulae. But first we need to establish the fact that is is possible to verify whether a given NRA will accept anything at all, i.e. that the emptiness problem is decidable, as we need this property to show the decidability of $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^x_{\leq}]$ in the next chapter.

3.2 Emptiness of NRA

3.2.1 Preliminaries

The general idea of the upcoming proof is as follows. First we will show that the possible configurations of our automaton can be seen as a *well-quasi-ordering*, which is an ordering fulfilling some good properties. As a NRA configuration is quite complex, this will be done in multiple steps building upon each other, relying on known results from order theory. Then we will show that the transition relation of the automaton and the well-quasi-ordering harmonize in a certain way, making the configuration graph of NRA a *well-structured transition system*. Using this fact we will be able to imply that emptiness must be decidable, by applying an according result from Finkel and Schnoebelen (2001). Before we can start, let us formalize these concepts:

Definition 3.3 (Well-Quasi-Ordering (WQO))
Let $(M, \preccurlyeq)$ *be a quasi-ordering.*

$(M, \preccurlyeq)$ *is called* well-quasi-ordering*, if every infinite sequence of elements* $m_1 m_2 m_3 \cdots$ *from M contains two elements* m_i, m_j*, so that* $m_i \preccurlyeq m_j$ *and* $i < j$*.*

The definition of a well-quasi-ordering basically says, that it is not possible to construct infinite strictly decreasing sequences or infinite *antichains* – sequences of incomparable elements.

Consider the partial ordering of natural numbers $(\mathbb{N}, \leq)$. This is an example of a well-quasi-ordering, because starting at any number $n \in \mathbb{N}$, there are only finitely many numbers smaller than n, so it is impossible to create an infinite sequence that is strictly decreasing all the time – we can start the sequence decrementing n one by one, but eventually we reach the minimal element 1 and then we can not go down anymore.

Now consider the partial ordering of whole numbers $(\mathbb{Z}, \leq)$ – if we start counting down, we will never run out of smaller numbers, because there is no minimal element we will ever run into. So the whole numbers are not *well-founded* and therefore $(\mathbb{Z}, \leq)$ is not a well-quasi-ordering. Also the ordering $(\mathbb{N}, |)$ is not a well-quasi-ordering, where $|$ is the divisibility relation – we know that there are infinitely many prime numbers, none of them being the divisor of any other, so the sequence of prime numbers would give us an infinite antichain.

Lemma 3.4 (Erdös & Rado)
Let $\leq$ *be a well-quasi-ordering. Then any infinite sequence contains an infinite increasing subsequence* $x_{i_0} \leq x_{i_1} \leq \ldots$ *with* $i_0 < i_1 < \ldots$.

Proof. See (Finkel and Schnoebelen, 2001, Lemma 2.2)

□

This fact is just a simple consequence from the definition of well-quasi-orderings – as decreasing elements are finite, at some point they are exhausted. Therefore, regardless of the element we start with, an increasing element must be taken after a finite amount of decreasing elements in between.

Lemma 3.5 (Dickson)
Let $\leq_k \subseteq \mathbb{N}_0^k \times \mathbb{N}_0^k$ be a product ordering, i.e. such that

$$(x_1, \ldots, x_k) \leq_k (y_1, \ldots, y_k) :\Leftrightarrow \forall i \in [k] : x_i \leq y_i$$

For all $k \in \mathbb{N}_0$, $(\mathbb{N}_0^k, \leq_k)$ is a well-quasi-ordering.

Proof. See Dickson (1913).

□

Dicksons lemma basically says that in every subset of k-tuples of natural numbers there exists a finite set of minimal elements with regard to the described ordering, thereby ensuring that no infinitely decreasing sequence in such subsets is possible, giving us a well-quasi-ordering. Originally, this lemma was used by Dickson to prove a number-theoretic statement about perfect numbers, but the statement holds for other product orderings based on well-quasi-ordered elements as well.

Definition 3.6 (Embedding ordering)
Let $(S, \preccurlyeq)$ be a quasi-ordering and $x = x_1 \ldots x_n, y = y_1 \ldots y_m \in S^*, n, m \in \mathbb{N}$ be finite sequences of elements of S. The relation $\sqsubseteq \subseteq S^* \times S^*$, such that

$$x \sqsubseteq y :\Leftrightarrow \exists_{1 \leq i_1 < \ldots < i_n \leq m} \forall_{j \in [n]} : x_j \preccurlyeq y_{i_j}$$

is called the embedding ordering *over S^*.*

Lemma 3.7 (Higman)
Let $(S, \preccurlyeq)$ be a well-quasi-ordering and $\sqsubseteq \subseteq S^* \times S^*$ be the embedding ordering over S^*. Then $(S^*, \sqsubseteq)$ is a well-quasi-ordering.

Proof. See Higman (1952).

□

Lemma 3.8 (Finite Multiset WQO)
Let $(S, \preccurlyeq)$ be a WQO. $(\mathcal{M}(S), \preccurlyeq^{\mathcal{M}})$ is called the finite multiset WQO *of $(S, \preccurlyeq)$ and is a WQO for $\preccurlyeq^{\mathcal{M}}$ such that for all finite multisets $M = \{m_1, \ldots, m_p\}, M' = \{m'_1, \ldots, m'_r\} \in \mathcal{M}(S), m_i, m'_j \in S$:*

$$M \preccurlyeq^{\mathcal{M}} M' \textit{ iff there is an injection } h : [p] \to [r] \textit{ such that } \forall_{1 \leq i \leq p} : m_i \preccurlyeq m'_{h(i)}$$

Proof. From each finite multiset from $M = \{m_1, \ldots, m_n\} \in \mathcal{M}(S)$ it is possible to construct a finite sequence of the elements $X = x_1 x_2 \ldots x_n \in S^*$ by linearising the multisets in an arbitrary order using a bijection $b : [n] \to [n]$ with $b(i) = j$ such that $m_i = x_j$, mapping each element of the multiset to a position in the corresponding sequence.

Now consider an infinite sequence of multisets $M_1 M_2 \ldots, M_i \in \mathcal{M}(S)$. For each such sequence let $X_1 X_2 \ldots, X_i \in S^*$ be the corresponding infinite sequence of finite sequences, where each X_i corresponds to a multiset M_i by such a bijection b_i. From Higmans Lemma (3.7) we know, that the embedding ordering over finite sequences $(S^*, \sqsubseteq)$ is a well-quasi-ordering, therefore in every such infinite sequence there are some X_i, X_j so that $X_i \sqsubseteq X_j$ and $i < j$, which means that for each $x_k \in X_i$ there is some $x_l \in X_j$, such that $x_k \preccurlyeq x_l$.

Let $p = |X_i|, q = |X_j|$. By construction, we know that each x_k corresponds to a unique $m_{f(k)}$ in the according multiset M_i and each x_l corresponds to a unique $m_{g(l)}$ in the multiset M_j by some bijections f and g. Therefore we have for all $k \in [p]$ that $m_{f(k)} \preccurlyeq m_{g(l)}$ for some $l \in [q]$. We can easily construct an injection $h : [p] \to [q]$ with $h(f(k)) = g(l)$, making sure that $m_r \preccurlyeq m_{h(r)}$ for all $r \in [p]$ and therefore have by definition $M_i \preccurlyeq^{\mathcal{M}} M_j$. We conclude, that $(\mathcal{M}(S), \preccurlyeq^{\mathcal{M}})$ is also a well-quasi-ordering.

□

The embedding ordering is a construction on top of sequences of quasi-ordered sets, for example, we can define the embedding ordering over finite sequences of natural numbers. In figure 3.2 you can see that there is a subsequence of y so that each element of x is smaller than the corresponding element of the subsequence of y, so $x \sqsubseteq y$. In the case of z you can not find a subsequence of y that fulfills that condition, we would have to start at the 3 which is trivially smaller or equal than the 3 in z, but then we have not enough positions in y to find corresponding elements to each z_i, so $z \not\sqsubseteq y$. As we have to match every position of the left sequence to positions of the right sequence, which is not possible if the left sequence is longer, it is clear that $y \not\sqsubseteq x$ and $y \not\sqsubseteq z$ and that in general this relation can only be symmetrical in cases where both sequences have the same length. Higmans lemma tells us, that such embedding orderings are always also well-quasi-orderings, if the underlying ordering is a well-quasi-ordering.

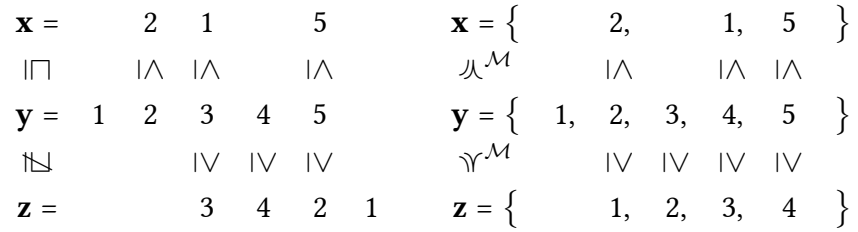


Figure 3.2: Comparison with embedding ordering $\sqsubseteq$ over $(\mathbb{N}^*, \leq)$ and finite multiset ordering $\preccurlyeq^{\mathcal{M}}$ over $(\mathcal{M}(\mathbb{N}), \leq)$

If you remove the information about positions from some sequence, you get a set containing all these elements in an arbitrary order, possibly containing duplicates – a multiset. In the upcoming proofs a well-quasi-ordering on multisets is sufficient and sequencing is not required, so Lemma 3.8 applies Higman's Lemma to multisets, removing the structure imposed by the sequencing. In figure 3.2 you can see, that the same values as in the embedding order, when interpreted as a multiset, can be reordered and permit more ways to compare elements. Therefore the multiset ordering is more liberal than the embedding order.

Definition 3.9 (Transition systems)
Let S be a set of states and $\rightarrow \subseteq S \times S$ be a transition relation. Then $(S, \rightarrow)$ is a transition system.

- $\mathsf{Succ}(s)$ *denotes the set of direct successors and* $\mathsf{Pred}(s)$ *the set of direct predcessors of state $s \in S$.*
- *If* $\mathsf{Succ}(s)$ *is finite for all $s \in S$, the transition system is* finitely branching.
- *If* $\mathsf{Succ}(s)$ *is computable for all $s \in S$, the transition system is* effective.
- *A transition system with a well-quasi-order relation $\leq \subset S \times S$ is called* reflexive downward compatible *with regard to $\leq$, if and only if for all $a_1, a_2, a_1' \in S$ with $a_1 \rightarrow a_2$ and $a_1' \leq a_1$ there exists a_2' with $a_2' \leq a_2$ and either $a_1' \rightarrow a_2'$ or $a_1' = a_2'$.*

The notion of a transition system is a concept unifying different constructions that use a kind of states and transitions between them, regardless of the additional structure, like initial and accepting states, labels and other details specific to the construction. This way it is possible to talk about the behaviour and properties of different systems using the same language. In our case, the configurations of the NRA automaton from the

$$C = (i, \alpha, (a, \mathbf{d}), T),\ T = \{((4,3,2), q_0), ((4,3,6), q_2), ((4,1,9), q_1), \ldots\} \in P_3$$

$$T = \left\{ \begin{array}{ccccc} 4 & 4 & 4 & 6 & 6 \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ 3 & 3 & 1 & 1 & 7 \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ 2 & 6 & 9 & 9 & 4 \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \boxed{q_0} & \boxed{q_2} & \boxed{q_1} & \boxed{q_1} & \boxed{q_5} \end{array} \right\} = \left\{ \begin{array}{ccccc} & & \textcircled{4} & & \textcircled{6} \\ & \nearrow & & \nwarrow & \uparrow \\ & 3 & & 1 & 1 \quad 7 \\ & \nearrow & \nwarrow & \uparrow & \uparrow \quad \uparrow \\ 2 & & 6 & 9 & 9 \quad 4 \\ \uparrow & & \uparrow & \uparrow & \uparrow \quad \uparrow \\ \boxed{q_0} & & \boxed{q_2} & \boxed{q_1} & \boxed{q_1} \quad \boxed{q_5} \end{array} \right\}$$

$$= \{(d_1, t_1), (d_2, t_2)\} \quad d_i \in \Delta,\ t_i \in P_2$$

$$\ldots = \{(4, \{(3, \{(2, q_0), (6, q_2)\}), (1, \{(9, q_1)\})\}), (6, \{(1, \{(9, q_1)\}), (7, \{(4, q_5)\})\})\}$$

Figure 3.3: Forest representation of the threads of a 3-NRA configuration

previous chapter yield the transition system that we will talk about. Now, to show that emptiness is decidable, the main task is to prove that NRA configurations give rise to a well-quasi-ordering and their transition relation is reflexive downward compatible with regard to their well-quasi-ordering, making NRA a *well-structured transition system.*

3.2.2 Proof of decidability

Definition 3.10 (Thread forests)
Let $P_k := \mathcal{P}_{\leq\infty}(\Delta^{[k]} \times Q), k > 0$ *denote the set of finite subsets of* $[k]$*-attributed NRA-threads and* $P_0 := \mathcal{P}(Q)$*.*

An element $T \in P_k$ *can be viewed as a forest and represented as a set of tuples* $\{(d_1, t_1), \ldots, (d_n, t_n)\}$*, where* $d_i \in \Delta$ *are the roots and* $t_i \in P_{k-1}$ *are the sets of corresponding subtrees. Let* $\mathsf{sub}(T) = \{t_1, \ldots, t_n\}$ *denote the multiset of sets of subtrees.*

In figure 3.3 you can see how a set of NRA threads can be seen as a forest – a thread consists of a state and a valuation for the linear ordered attributes. We can build a forest out of threads using a shared data valuation prefix, because we know that the values in a linear data vector are depending on each other. This way we obtain a compact representation of all values which are present in the current configuration that we can better reason about. Note however, that these forests have *no relation at all* to the tree-quasi-ordered attributes from some possible underlying $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ formula, being based on data equality on prefixes of $[k]$-vectors. This notion is formalized in the following definition:

Definition 3.11 (Nested permutation)
Let $f : \Delta^{[k]} \to \Delta^{[k]}, k \in \mathbb{N}$ be a bijection. f is a nested permutation*, iff for all tuples $\mathbf{d}, \mathbf{e} \in \Delta^{[k]}$ the length of their longest common prefix is invariant under f.*

This means, for $\mathbf{d} = (d_1, \ldots, d_k), \mathbf{e} = (e_1, \ldots, e_k), f(\mathbf{d}) = (d'_1, \ldots, d'_k), f(\mathbf{e}) = (e'_1, \ldots, e'_k)$ and for all $1 \leq i \leq k$:

$$(d_1, \ldots, d_i) = (e_1, \ldots, e_i) \Leftrightarrow (d'_1, \ldots, d'_i) = (e'_1, \ldots, e'_i)$$

The extension *of f with regard to some set Q is defined for $d \in \Delta^{[k]}, q \in Q$ as:*

$$\hat{f} : \Delta^{[k]} \times Q \to \Delta^{[k]} \times Q, \hat{f}(\mathbf{d}, q) = (f(\mathbf{d}), q)$$

Using the definition of nested permutations we now will define multiple relations that treat different threads as equal modulo consistent renaming of the data values. As we will see, this allows us to basically collapse the infinite data domain into a finite set of structurally equivalent cases with the same relationships and treat them as the same thing.

Definition 3.12 ($\preccurlyeq_k$)
Let $\preccurlyeq_k \subseteq P_k \times P_k$ be an ordering relation on subsets of k-NRA threads such that for $T, T' \in P_k$, $T \preccurlyeq_k T'$ iff there is a nested permutation f, so that $T \subseteq \hat{f}(T')$.

Definition 3.13 ($\sim$)
Let $\sim \subseteq \mathcal{C}^k_{\mathsf{NRA}} \times \mathcal{C}^k_{\mathsf{NRA}}$ be an equivalence relation on k-NRA configurations such that $(i, \alpha, (a, \mathbf{d}), T) \sim (i', \alpha', (a', \mathbf{d}'), T')$ iff $\alpha = \alpha', a = a'$ and there is a nested permutation f such that $\hat{f}(T) = T' \wedge f(\mathbf{d}) = \mathbf{d}'$.

Definition 3.14 ($\preceq$)
Let $\preceq \subseteq \mathcal{C}^k_{\mathsf{NRA}} \times \mathcal{C}^k_{\mathsf{NRA}}$ be an ordering relation on k-NRA configurations such that $(i, \alpha, (a, \mathbf{d}), T) \preceq (i', \alpha', (a', \mathbf{d}'), T')$ iff $\alpha = \alpha', a = a', d = d'$ and $T \subseteq T'$.

Definition 3.15 ($\precsim$)
Let $\precsim \subseteq \mathcal{C}^k_{\mathsf{NRA}} \times \mathcal{C}^k_{\mathsf{NRA}}$ be an ordering relation on k-NRA configurations such that $c \precsim c'$ iff there is a $c'' \sim c'$ with $c \preceq c''$.

The $\preccurlyeq_k$ relation compares raw sets of threads with each other, establishing a kind of

indirect subset relation and $\sim$ relation is an equivalence relation on configurations. The $\preceq$ relation is an exact relation in the sense that it only compares configurations with sets of threads that are truly in direct subset relation, giving a partial ordering on configurations. Finally $\precsim$ combines $\sim$ with $\preceq$ to regain the indirect semantics and compare configurations with similar, but not necessarily equal thread sets, basically lifting $\preccurlyeq_k$ from sets of threads to configurations.

This is sensible, because what matters for acceptance is not particular values, but how the values that are present in some data word and configuration relate to each other with regard to attribute equality, as defined by our semantics. More importantly, we will see that $\precsim$ has the properties we need to make NRA a well-structured transition system.

Lemma 3.16
$(P_k, \preccurlyeq_k)$ is a well-quasi-ordering.

Proof. By induction on k. For $k = 0$ we have by definition $P_0 = Q$ and then $\preccurlyeq_0 = \subseteq$ is the subset ordering on the finite set Q, which is trivially a WQO.

Now let $k > 0$ and assume that $(P_{k-1}, \preccurlyeq_{k-1})$ is a WQO. We show that $(P_k, \preccurlyeq_k)$ is a WQO as well.

Consider an infinite sequence $T_1 T_2 \dots$ of elements of P_k. In the corresponding sequence $\mathsf{sub}(T_1)\mathsf{sub}(T_2)\dots$ we find two positions $i < j$ with $\mathsf{sub}(T_i) \preccurlyeq^{\mathcal{M}}_{k-1} \mathsf{sub}(T_j)$, since $(\mathcal{M}(P_{k-1}), \preccurlyeq^{\mathcal{M}}_{k-1})$ is a finite multiset WQO by Lemma 3.8. Assume in the following $R := T_i = \{(d_1, r_1), \dots, (d_n, r_n)\}$ and $S := T_j = \{(e_1, s_1), \dots, (e_m, s_m)\}$. We construct a nested permutation witnessing that $R \preccurlyeq_k S$.

Since $\mathsf{sub}(R) \preccurlyeq^{\mathcal{M}}_{k-1} \mathsf{sub}(S)$, there is an injection $h : [n] \to [m]$ with $r_i \preccurlyeq_{k-1} s_{h(i)}$ for all $1 \leq i \leq n$. This in turn means that for each r_i there is an (extension of a) nested permutation $f_i : P_{k-1} \to P_{k-1}$ with $r_i \subseteq f(s_{h(i)})$ and for each of them we define $\hat{f}_i : (\{e_{h(i)}\} \times \Delta^{[k-1]} \times Q) \to (\{d_i\} \times \Delta^{[k-1]} \times Q)$ with $\hat{f}_i(e_{h(i)}, \mathbf{d}) = (d_i, f_i(\mathbf{d}))$ for all $\mathbf{d} \in \Delta^{[k-1]} \times Q$.

That is, $\hat{f}_i$ simply lifts the witnessing permutation for the $r_i \preccurlyeq_{k-1} s_{h(i)}$ in the unique possible way to the corresponding trees $(d_i, r_i) \in R$ and $(e_{h(i)}, s_{h(i)}) \in S$. Since h is injective, the data values $e_{h(i)}$ for all $1 \leq i \leq n$ are all different. We therefore can choose an (extension of a) nested permutation $\hat{f} : P_k \to P_k$ such that $\hat{f}(e_{h(i)}, \mathbf{d}) = \hat{f}_i(e_{h(i)}, \mathbf{d}) = (d_i, f_i(\mathbf{d}))$.

By construction we have that $R \subseteq \hat{f}(S)$ since the nested permutations f_i guarantee that property on the subtrees and h is injective meaning that each tree in R corresponds to a distinct subset of $\mathsf{img}(\hat{f})$. We conclude that $R \preccurlyeq_k S$.

□

Lemma 3.17
For all configurations $c_1, c_1', c_2 \in \mathcal{C}^k_{\mathsf{NRA}}$ for a fixed $k \in \mathbb{N}$ with $c_1 \sim c_1'$ and $c_1 \to c_2$ there is a configuration $c_2' \in \mathcal{C}^k_{\mathsf{NRA}}$ with $c_2' \sim c_2$ and $c_1' \to c_2'$.

Proof. Let f be a nested permutation witnessing the equivalence of c_1 and c_1'. The proof follows by a case analysis of the transition relation $\to$ (Definition 3.2).

In case of eq_i, if two vectors $\mathbf{d}, \mathbf{d}'$ are identical at every position $j \leq i$, their counterparts $f(\mathbf{d}), f(\mathbf{d}')$ will be as well by Definition 3.11, leading to identical behaviour in the transformed thread set – the elimination of the according thread. The case for $\overline{\mathsf{eq}_i}$ is similar.

All other cases do not depend on the data vectors in the configuration at all and therefore the corresponding action still can be performed in the transformed configuration c_1' and in every case the permutation f still witnesses the equivalence of the respective following states c_2 and c_2', hence $c_2 \sim c_2'$.

In the case of guess a new valuation can be guessed and stored in the register regardless of the previous value stored, so this action can be performed from any c_1' and leads to a corresponding state c_2', where the register of the according thread is overwritten with the guessed value.

Similarily, in the case of spread the action does not depend on the stored vector, and as there exists the same number of threads in state q_1 in c_1' as in c_1, according threads in state q_2 can be created leading to the configuration c_2'.

□

Lemma 3.18
For a fixed $k \in \mathbb{N}$, $(\mathcal{C}^k_{\mathsf{NRA}}, \precsim)$ is a well-quasi-ordering.

Proof. For some fixed k, every infinite sequence of configurations contains an infinite subsequence of configurations $c_1 c_2 \ldots \in \mathcal{C}^k_{\mathsf{NRA}}$ with $c_j = (i_j, \alpha, (a, \mathbf{d}_j), T_j)$ where α and

a are fixed, as the possible values for α and a are finite. Due to Lemma 3.16 and Lemma of Erdös and Rado (3.4) we can assume that the T_j are increasing, i.e. $T_j \preccurlyeq_k T_{j+1}$.

Let $T_{j,l} := \{\mathbf{d}_j(1), \ldots, \mathbf{d}_j(l), \ldots, \mathbf{d}_j(k), q) \in T_j\}$ be the subtree of T_j starting with $\mathbf{d}_j(1), \ldots, \mathbf{d}_j(l)$ for any $1 \leq l \leq k$. Consider now the sequence of tuples $S_1 S_2 \ldots$ with $S_j = (T_j, T_{j,1}, \ldots, T_{j,k})$. Due to Lemma 3.16 we have a WQO on every component and by Dicksons Lemma (3.5) a WQO on the tuples.

Thus, this sequence contains two tuples S_m and S_n with $m < n$. Due to the fact that subtrees in T_m starting with prefixes of $\mathbf{d}_m$ are smaller with respect to $\preccurlyeq_k$ than subtrees in T_n starting with prefixes of $\mathbf{d}_n$, $\mathbf{d}_n$ can be mapped to $\mathbf{d}_m$ by the nested permutation underlying $\precsim$. It follows that $c_m \precsim c_n$ and therefore $(\mathcal{C}^k_{\mathsf{NRA}}, \precsim)$ is a WQO.

□

The construction of the tuples S_j with the series of the $T_{j,l}$ subtrees is a means to abstract away the actual data values, while preserving the structural relationship between the data valuation $\mathbf{d}_j$ at some position i_j and the valuations stored inside of the threads. This finally allows us to argue that the configurations give rise to a well-quasi-ordering.

Lemma 3.19
For a fixed $k \in \mathbb{N}$, $(\mathcal{C}^k_{\mathsf{NRA}}, \rightarrow)$ is reflexive downward-compatible with respect to $(\mathcal{C}^k_{\mathsf{NRA}}, \precsim)$.

Proof. In Lemma 3.17 it has been shown, that for any two equivalent configurations the same actions can actually be performed, which was just assumed implicitly in Figueira (2012). Other than that it is the same proof as (Figueira, 2012, Lemma 3.10), verifying the property by a case analysis on the transition relation.

We explicitly examine the cases for guess and spread:

Let $k \in \mathbb{N}$ be fixed and $c_1, c_2, c_1' \in \mathcal{C}^k_{\mathsf{NRA}}$ with $c_1 \rightarrow c_2$ and $c_1' \precsim c_1$. We will see that there is a c_2' such that $c_2' \precsim c_2$ and either $c_1' \rightarrow c_2'$ or $c_2' = c_1'$.

As $\precsim$ implies the existence of a nested permutation f witnessing $c_1 \sim c_1'$, without loss of generality we assume now that $c_1' \preceq c_1$.

If a guess is performed, let

$$c_1 = (i, \alpha, (a, \mathbf{d}), \{(\mathbf{d}', q)\} \cup T) \text{ and } c_2 = (i, \alpha, (a, \mathbf{d}), \{(\mathbf{e}, q')\} \cup T)$$

with $\delta(q) = \mathsf{guess}(q')$ and let $c_1' = (i', \alpha, (a, \mathbf{d}), T')$.

Suppose that there is $(\mathbf{d}', q) \in T'$. We take the guess transition from c_1' and obtain a c_2' by guessing $\mathbf{e}$, hence $c_2' \precsim c_2$ holds by the same nested permutation witnessing that $c_1' \precsim c_1$. Otherwise, if $(\mathbf{d}', q) \notin T'$, we set $c_2' = c_1'$ and also have $c_2' \precsim c_2$, because we know that $c_2' = c_1' \precsim c_1 \preceq c_2$.

If a spread is performed, let

$$c_1 = (i, \alpha, (a, \mathbf{d}), \{(\mathbf{d}', q)\} \cup T) \text{ and } c_2 = (i, \alpha, (a, \mathbf{d}), \{(\mathbf{d}, q_1) \mid (\mathbf{d}, q_2) \in T\} \cup T)$$

with $\delta(q) = \mathsf{spread}(q_2, q_1)$ and let $c_1' = (i', \alpha, (a, \mathbf{d}), T')$.

Suppose that there is $(\mathbf{d}', q) \in T'$. We take the spread transition and obtain a configuration c_2' with $c_2' \precsim c_2$, because any $(\mathbf{e}, q_1)$ in c_2' generated by the spread must come from a $(\mathbf{e}, q_2)$ in c_1', hence there is some $(\mathbf{e}, q_2)$ in c_1 which leads to $(\mathbf{e}, q_1)$ in c_2. If $(\mathbf{d}', q) \notin T'$, by a similar argument as above we set $c_2' = c_1'$ and also have $c_2' \precsim c_2$.

□

Theorem 3.20 (Emptiness of NRA)
Emptiness is decidable for NRA.

Proof. To decide emptiness of an NRA we have to decide whether an accepting configuration is reachable, i.e. a configuration where the set of threads is empty. By Lemma 3.18 and 3.19, $(\mathcal{C}^k_{\mathsf{NRA}}, \to, \precsim)$ is a *downward-WSTS with reflexive compatibility* in the sense of (Finkel and Schnoebelen, 2001, Definition 5.1). Further, the relation $\precsim$ is decidable and the transition relation $\to$ is effective. Since the set of accepting configurations is downward closed, it follows from (Finkel and Schnoebelen, 2001, Proposition 5.4) that it is decidable whether one of them is reachable.

□

As you can see in comparison with the corresponding proof for ARA, the part requiring the most work is showing that $\precsim$ (as defined for NRA) is a well-quasi-ordering, because of the more complex, nested structure of the data values in each thread. The introduction of guess and spread on the other hand does not require much additional effort in these proofs and their cases in the analysis of the transition relation do not pose any problems and work in exactly the same way as for ARA.

4 Decidability of the logic

In this chapter I will show that it is possible to translate any $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ formula over tree-ordered attribute sets into NRA accepting exactly the same words which satisfy the underlying formula.

As NRA are defined over $[k]$-attributed data words (which are easier to reason about), but we want to show decidability for formulae expressing properties of more general A-attributed data words, first I will present a way to translate arbitrary $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ formulae to equisatisfiable $\mathsf{LTL}^{\downarrow}_{[k]}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ formulae, using the linearisation technique from Decker and Thoma (2015) and extending it with the corresponding cases for the $\exists_{\geq}$ and $\forall^{x}_{\leq}$ operators. Next, I will show how to translate $\mathsf{LTL}^{\downarrow}_{[k]}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ formulae into NRA automata, again extending a proof by Decker and Thoma [1] with the respective cases. This is basically the same procedure as described in (Figueira, 2012, Theorem 4.4), but generalised from ARA to NRA.

Having seen a model-preserving translation and using the fact that NRA emptiness is decidable this will finally prove the decidability of the logic.

4.1 Linearisation from $\mathsf{LTL}^{\downarrow}_{A}$ to $\mathsf{LTL}^{\downarrow}_{[k]}$

When looking at general tree-quasi-ordered attribute sets one can see a possible problem — attributes that depend on each other, i.e. elements in the same strongly connected component in the corresponding graph representation have the same downward-closure. As the logic can only store downward-closures, we have no way to distinguish between such attributes. But as from our perspective these attributes are the same, we can make this explicit and simplify our model by encoding all components into a single attribute:

[1] private communication

Lemma 4.1 (Collapsing of the strongly connected components)
Let A be a tree-quasi-ordered set of attributes. Every $\mathsf{LTL}^{\downarrow}_{A}$ formula φ can be translated to an equisatisfiable $\mathsf{LTL}^{\downarrow}_{A'}$ formula φ', where A' is a tree ordering.

Proof. Let $C_{i,j}$ be all maximal (thus disjoint) strongly connected components of the graph $G = (V, E)$ of A, where i denotes the size $|C_{i,j}|$ of the component and j enumerates components of the same size.

As the downward-closure of each element from the same SCC is identical, all attributes of a SCC can be encoded into a single attribute, because the set of possible values is infinite. To do this, choose for each $C_{i,j}$ an arbitrary representative element $x_{i,j} \in C_{i,j}$ and remove all other elements from the component. Because of transitivity it is assured that any edge between two components exists between any pair of elements from these components. Therefore we can simply restrict the edges to the representatives without affecting the relationships between components:

$$V' = \bigcup_{i,j} \{x_{i,j}\}, \quad E' = E \cap (V' \times V')$$

The graph $G' = (V', E')$ is now an acyclic graph representing a partial tree ordering A'.

In the formula φ replace every attribute $x \in C_{i,j}$ with the chosen representative $x_{i,j}$.

Now, concering the data valuation equivalence $\simeq$ a loss of information happened – if previously the comparison between attributes from components with different sizes would have failed because of a lack of isomorphism, in the formula with the collapsed attributes such comparisons could succeed. To prevent this, we add an additional constraint:

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

This constraint demands that two collapsed attributes from components that had different sizes never have the same valuation, which again is possible to grant due to the infinity of the data domain. Therefore the resulting $\mathsf{LTL}^{\downarrow}_{A'}$ formula φ', consisting of φ with replaced attributes and this constraint γ, is equisatisfiable with regard to the original $\mathsf{LTL}^{\downarrow}_{A}$ formula φ.

□

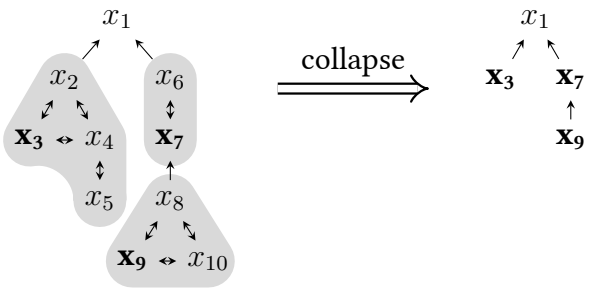


Figure 4.1: Example of the collapsing of a tree-quasi-ordering

In figure 4.1 you can see a tree-quasi-ordering which has been collapsed to a tree ordering. The components with more than one attribute have been merged into a single representative attribute highlighted in bold font. In the original ordering the downward-closures of the attributes x_2, x_3, x_4, x_5 are identical and have the size 5, whereas the downward-closures of the attribues x_6 and x_7 are also identical, but have the size 3, so a valuation comparison for attributes of these two different components can not happen – there is no way for the check-operator to further restrict the bigger component. But in the resulting ordering there is nothing preventing us from comparing the representatives x_3 and x_7, so we need the described constraint that forces us to assign different values to them, so that the comparison can never be successful.

As our goal is to translate formulae over tree-quasi-ordered attributes to formulae over linear ordered attributes, we first need to ensure that all models of the original formula can be also expressed for the resulting formula. We show that this is the case by describing the following encoding, which translates an arbitrary A-attributed data word into an equivalent data word over $[k]$-vectors.

Definition 4.2 (Frame encoding)
Let A be a tree ordering, $k = \mathsf{ht}(A)$ be the depth of A and $w \in (\Sigma \times \Delta^A)^+$ a data word of length m. We can translate the A-attributed data word w to a $[k]$-attributed data word w' in the following way:

First, the tree ordering has to be padded with dummy attributes, so that every leaf of a tree has a path of length k to a root. So for each attribute $x \in A$ with depth $l = \mathsf{ht}(x) < k$ we add a series of additional attributes $x_{l+1}, \ldots, x_k$ and add the relations $x \preccurlyeq x_{l+1}, x_{l+1} \preccurlyeq x_{l+2}, \ldots, x_{k-1} \preccurlyeq x_k$ to our ordering, obtaining the padded ordering A'. As all added attributes are larger than the original attributes, they are never part of the

downward-closures, in no way affecting the semantics of formulae over A. In each position of w these dummy attributes can be assigned arbitrary values without changing the property of being a model of a formula or not.

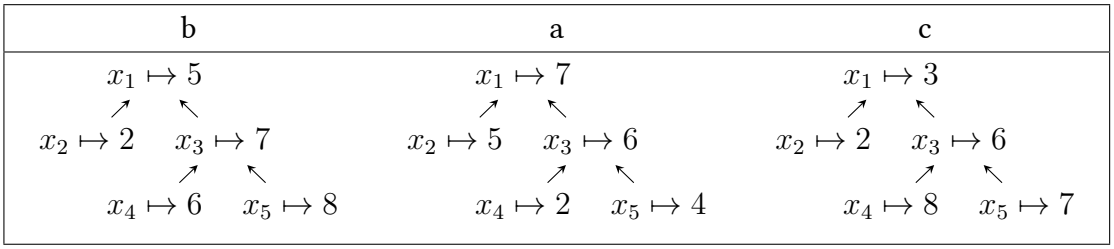
Let $l_1, \ldots, l_n \in A'$ be the leaves of A' and let $x_{j,1} \preccurlyeq \ldots \preccurlyeq x_{j,k} = l_j$ be the path of attributes of $\mathsf{cl}(l_j)$, representing a branch of the tree from the leaf l_j to a root. Let the leaves be enumerated in an arbitrary but fixed order such that paths sharing a common prefix are ordered consecutively. We encode each position $(a_i, \mathbf{d}_i)$ of w in a sequence of positions $(a_i, \mathbf{g}_{i,1}) \ldots (a_i, \mathbf{g}_{i,n})$. Each such sequence of n positions is called frame*. In every frame each position holds the same letter as the original position a_i, while each one of the valuations $\mathbf{g}_{i,j} \in \Delta^{[k]}$ encodes one branch $\mathbf{d}_i|_{\mathsf{cl}(l_j)}$ of the padded original valuation $\mathbf{d}_i \in \Delta^{A'}$, assigning $\mathbf{g}_{i,j}(r), r \in [k]$ the value of $\mathbf{d}_i(x_{j,r})$.*

The concatenation of the frames $(a_1, \mathbf{g}_{1,1}) \ldots (a_1, \mathbf{g}_{1,n}) \ldots\ldots (a_m, \mathbf{g}_{m,1}) \ldots (a_m, \mathbf{g}_{m,n})$ is the corresponding data word w'.

Definition 4.3
Let A be a tree ordering, $x \in A$ and $l_1, \ldots, l_n$ be the leaves of A, enumerated in some order in the sense of Definition 4.2.

- $\mathsf{sb}(x) = \min\{1 \leq r \leq n \mid x \preccurlyeq l_r\}$
  denotes the index of the smallest branch of A containing x.
- $\mathsf{lb}(x) = \max\{1 \leq r \leq n \mid x \preccurlyeq l_r\}$
  denotes the index of the largest branch of A containing x.

In figure 4.2 you can see a frame encoding in practice – the tree ordering has the depth 3, so vectors of size 3 are used to encode all branches of the tree and as there are three leaves in this tree (the padding does not change the number of leaves, as only a linear chain is added), there are three positions in each frame, resulting in a word with thrice the size, but having "flattened" the tree structure. The gray values are the arbitrarily assigned values for the padded attributes, because all vectors must have the same size. This procedure works in the same way for attribute sets with an ordering that induces multiple trees, because multiple root elements are allowed. In this case the trees just need a fixed order in which they appear in the frames.



⇓ encode ⇓

b	b	b	a	a	a	c	c	c
5	5	5	7	7	7	3	3	3
↑	↑	↑	↑	↑	↑	↑	↑	↑
2	7	7	5	6	6	2	6	6
↑	↑	↑	↑	↑	↑	↑	↑	↑
1	6	8	9	2	4	5	8	7

Figure 4.2: Example of the frame encoding from words over tree orderings to words over $[k]$-vectors

Now, before coming to the linearisation, we need some equivalences that are required by the transformation of the formula. They were assumed in Decker and Thoma (2015), but are shown here explicitly for the sake of completeness.

Lemma 4.4 (Equivalences in Freeze LTL)
The following equivalences apply for arbitrary formulae ψ, ξ, propositions $p \in AP$ and attributes $x, y \in A$:

$$\begin{aligned}
\downarrow^x p &\equiv p \\
\downarrow^x \downarrow^y \psi &\equiv \downarrow^y \psi \\
\downarrow^x \neg\psi &\equiv \neg \downarrow^x \psi \\
\downarrow^x (\psi \wedge \xi) &\equiv (\downarrow^x \psi) \wedge (\downarrow^x \xi) \\
\downarrow^x (\psi \mathbf{U} \xi) &\equiv (\downarrow^x \xi) \vee ((\downarrow^x \psi) \wedge \downarrow^x \mathbf{X}(\psi \mathbf{U} \xi))
\end{aligned}$$

Proof. Let ψ, ξ be arbitrary formulae, $p \in AP$ an arbitrary proposition and $x, y \in A$ arbitrary attributes. The equivalences are proved individually, using the definition of the satisfaction relation:

1.

$$(w,i,\mathbf{d}) \models\downarrow^x p \quad\Leftrightarrow\quad (w,i,\mathbf{d}_i|_{\mathsf{cl}(x)}) \models p \quad\Leftrightarrow\quad p \in a_i \quad\Leftrightarrow\quad (w,i,\mathbf{d}) \models p$$

2.

$$\begin{aligned} (w,i,\mathbf{d}) &\models \downarrow^x\downarrow^y \psi \;\Leftrightarrow\; (w,i,\mathbf{d}_i|_{\mathsf{cl}(x)}) \models \downarrow^y \psi \\ \Leftrightarrow\; (w,i,\mathbf{d}_i|_{\mathsf{cl}(y)}) &\models \psi \;\Leftrightarrow\; (w,i,\mathbf{d}) \models \downarrow^y \psi \end{aligned}$$

3.

$$\begin{aligned} (w,i,\mathbf{d}) &\models \downarrow^x \neg\psi \\ \Leftrightarrow\quad (w,i,\mathbf{d}_i|_{\mathsf{cl}(x)}) &\models \neg\psi \\ \Leftrightarrow\quad (w,i,\mathbf{d}_i|_{\mathsf{cl}(x)}) &\not\models \psi \\ \Leftrightarrow\quad (w,i,\mathbf{d}) &\not\models \downarrow^x \psi \\ \Leftrightarrow\quad (w,i,\mathbf{d}) &\models \neg\downarrow^x \psi \end{aligned}$$

4.

$$\begin{aligned} (w,i,\mathbf{d}) &\models \downarrow^x (\psi\wedge\xi) \\ \Leftrightarrow\quad (w,i,\mathbf{d}_i|_{\mathsf{cl}(x)}) &\models \psi\wedge\xi \\ \Leftrightarrow\quad (w,i,\mathbf{d}_i|_{\mathsf{cl}(x)}) &\models \psi \quad\text{and}\quad (w,i,\mathbf{d}_i|_{\mathsf{cl}(x)}) \models \xi \\ \Leftrightarrow\quad (w,i,\mathbf{d}) &\models \downarrow^x \psi \quad\text{and}\quad (w,i,\mathbf{d}) \models \downarrow^x \xi \\ \Leftrightarrow\quad (w,i,\mathbf{d}) &\models (\downarrow^x \psi)\wedge(\downarrow^x \xi) \end{aligned}$$

5. Using the known unwrapping of $\mathbf{U}$ and equivalence 4:

$$\begin{aligned} (w,i,\mathbf{d}) &\models \downarrow^x (\psi\mathbf{U}\xi) \\ \Leftrightarrow\quad (w,i,\mathbf{d}) &\models\downarrow^x (\xi\vee(\psi\wedge\mathbf{X}(\psi\mathbf{U}\xi))) \\ \Leftrightarrow\quad (w,i,\mathbf{d}) &\models (\downarrow^x \xi)\vee \downarrow^x (\psi\wedge\mathbf{X}(\psi\mathbf{U}\xi)) \\ \Leftrightarrow\quad (w,i,\mathbf{d}) &\models (\downarrow^x \xi)\vee((\downarrow^x \psi)\wedge\downarrow^x \mathbf{X}(\psi\mathbf{U}\xi)) \end{aligned}$$

□

Theorem 4.5 (Linearisation)
If A is a tree-quasi-ordered set of attributes of depth k, then every $\mathsf{LTL}^{\downarrow}_{A}$ formula can be translated into an equisatisfiable $\mathsf{LTL}^{\downarrow}_{[k]}$ formula.

Proof. Using Lemma 4.1 we can first translate the formula into one over a tree-partial-ordered attribute set. Therefore, without loss of generality, we assume now that A is a tree-ordering.

In Definition 4.2 an encoding has been presented that allows the translation of data words over tree-ordered attribute sets into data words over $[k]$-vectors. What is left to show is that we can correctly translate any $\mathsf{LTL}^{\downarrow}_{A}$ formula Φ over data words with tree-ordered attributes into an $\mathsf{LTL}^{\downarrow}_{[k]}$ formula $\hat{\Phi}$ that is checked over the corresponding frame-encoded data words.

Using the equivalences proven in Lemma 4.4 we can assume that $\hat{\Phi}$ is in a normal form where every freeze quantifier $\downarrow^x$ is immediately followed by either $\mathbf{X}$ or $\uparrow^y$ for attributes $x, y \in A$. Further it can be assumed for every formula $\downarrow^x\uparrow^y$ that $\mathsf{sb}(x) \leq \mathsf{sb}(y)$:

If $x \prec y$ or $x \succcurlyeq y$, we can completely remove the formula, replacing it respectively with a contradiction or tautology, e.g. false or true, because if $x \prec y$, the attributes can not be compared in any way and if $x \succcurlyeq y$, because of the specified semantics the comparison will always be true – the check operator can then restrict $\mathbf{d}|_{\mathsf{cl}(x)}$ to $\mathbf{d}|_{\mathsf{cl}(y)}$, essentially comparing y to y. Hence now assume x and y to be incomparable with regard to the attribute ordering A.

If $\mathsf{ht}(x) < \mathsf{ht}(y)$, again there is no way to compare the attributes, making it a contradiction. If $\mathsf{ht}(x) \geq \mathsf{ht}(y)$, there exists a unique attribute $p \prec x$ with $\mathsf{ht}(p) = \mathsf{ht}(x)$ (because the downward-closures are linear) which will be used for comparison by definition of the semantics, so we can safely replace $\downarrow^x\uparrow^y$ with $\downarrow^p\uparrow^y$. Now, as $\mathsf{ht}(p) = \mathsf{ht}(y)$, we can swap p and y, if necessary.

We extend the alphabet to add new propositions $p_1, \ldots, p_n$ that are supposed to indicate the position in the current frame and demand that each letter contains one of these propositions, making these propositions a kind of cyclic counter.

Now we have to enforce that data words indeed have the assumed structure. First we

check that the assumed counter works as expected:

$$\beta_1 := p_1 \wedge \mathbf{G}\left(\bigwedge_{i\in\{1,\dots,n\}} p_i \Rightarrow \left(\bigwedge_{j\in\{1,\dots,n\}\setminus\{i\}} \neg p_j \wedge (\overline{\mathbf{X}} p_{(i \bmod n)+1})\right)\right)$$

Next we check that the letter in positions within the same frame does not change:

$$\beta_2 := \bigwedge_{a\in\Sigma} \mathbf{G}(p_1 \wedge a \Rightarrow a\mathbf{U}(p_n \wedge a))$$

Finally we verify that each attribute is consistently encoded throughout every frame. In the assumed encoding each attribute can be identified by its depth in the linear ordering and the positions that encode it within a frame, so we ensure this property by verifying for each attribute that the subvectors encoding it are equal in all positions of a frame encoding this attribute:

$$\beta_3 := \bigwedge_{x\in A} \mathbf{G}\left(p_{\mathsf{sb}(x)} \Rightarrow \downarrow^{\mathsf{ht}(x)} \left(\uparrow^{\mathsf{ht}(x)} \mathbf{U}(p_{\mathsf{lb}(x)} \wedge \uparrow^{\mathsf{lb}(x)})\right)\right)$$

Now we inductively define the translation $t(\varphi)$ for subformulae ξ, ψ of $\phi, x \in A, a \in \Sigma$:

$$\begin{aligned}
t(a) &:= a \\
t(\neg\psi) &:= \neg t(\psi) \\
t(\psi \wedge \xi) &:= t(\psi) \wedge t(\xi) \\
t(\mathbf{X}\psi) &:= \wedge_{j=1}^{n} p_j \Rightarrow \mathbf{X}^{n-j+1} t(\psi) \\
t(\psi \mathbf{U} \xi) &:= (p_1 \Rightarrow t(\psi))\mathbf{U}(p_1 \wedge t(\xi)) \\
t(\downarrow^x \psi) &:= \mathbf{X}^{\mathsf{sb}(x)-1} \downarrow^{\mathsf{ht}(x)} t(\psi) \\
t(\uparrow^x) &:= \wedge_{j=1}^{n} p_j \Rightarrow \mathbf{X}^{\mathsf{sb}(x)-j} \uparrow^{\mathsf{ht}(x)} \\
t(\exists_{\geq}\psi) &:= \exists_{\geq} t(\psi) \\
t(\forall^x_{\leq,\psi}\xi) &:= \forall^{\mathsf{ht}(x)}_{\leq,\psi\wedge p_{\mathsf{sb}(x)}} t(\xi)
\end{aligned}$$

The formula $\hat{\Phi} = t(\Phi) \wedge \beta_1 \wedge \beta_2 \wedge \beta_2$ exactly characterizes the encodings of models of Φ, because of the invariant that all subformulae are evaluated on the first position in a frame, except those preceded by a freeze quantifier. Those subformulae that follow a

freeze quantifier are either $\mathbf{X}\psi$ or $\uparrow^y$ and are relocated to the first position of the next frame, or the position encoding the branch that needs to be checked, respectively.

□

The direct translation of $\exists_{\geq}$ works, because the definition of data valuation equivalence does not depend on the attribute the data is from, only enforcing that both valuations being compared have an isomorphic structure. As in our encoding the underlying linear ordering $[k]$ is capable of storing any branch from our initial tree ordering A and the codomain (the possible data values) in both encodings and for all attributes is the same, the translated $\exists_{\geq}$-operator can still "guess" any possible valuation for any A-attribute, independently of the current position – we can always choose the $[k]$-attribute isomorphic to the A-attribute we are going to check, as all such isomorphic A-attributes are mapped to the same $[k]$-attribute, just in different positions (as the mapping of a single A-attribute to a single $[k]$-attribute is not injective). Therefore it suffices to guess a $[k]$-vector at the beginning of the frame, as it can be checked against an arbitrary attribute.

The translation of $\forall^x_{\leq,\psi}$ works by translating the A-attribute x from the original formula to the corresponding $[k]$-attribute $\mathsf{ht}(x)$ and extending ψ with the constraint that we only consider the first position in a frame containing a branch with the attribute x. This is necessary, because (as described above) in different positions the same $[k]$-attribute can encode different isomorphic A-attributes of the initial formula, depending on the underlying tree ordering. This is also sufficient, as we can assume (because of the formula β_3) that the valuation of the corresponding $[k]$-attribute stays the same in all positions of the frame where x is encoded, so we do not have to check the attribute in all positions of a frame where it is encoded.

4.2 Translation of formulae to NRA

Now we will translate the linearised formulae into NRA, which are perfectly tailored for our purpose, supporting all operations that are necessary to verify the underlying $\mathsf{LTL}^{\downarrow}_{[k]}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^x_{\leq}]$ formula on a suitable data word. Each state of the automaton essentially will encode the outermost token of a subformula that is left to verify, "branching" up for binary or n-ary operators connecting the subformulae.

Theorem 4.6 (Translation from $\mathsf{LTL}^{\downarrow}_{[k]}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ to k-NRA)
Every $\mathsf{LTL}^{\downarrow}_{[k]}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ *formula* φ *can be translated into a corresponding* k*-NRA* $\mathcal{A}_{\varphi}$*, so that for every non-empty data word* $w \in (\Sigma \times \Delta^{[k]})^{+}$*:*

$$w \textit{ satisfies } \varphi \Leftrightarrow \mathcal{A}_{\varphi} \textit{ accepts } w$$

Proof. Let $\varphi \in \mathsf{LTL}^{\downarrow}_{[k]}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ be an arbitrary formula.

To take care of the $\forall^{x}_{\leq,\psi}$ operator we first need to enforce the creation of threads that collect all values at relavant positions and exist until the end of the word is reached. Let j enumerate the different ψ used to filter the quantification and ψ_j be the j-th such formula.

We define the formulae $\eta_j := \mathbf{G}(\psi_j \Rightarrow \downarrow^{k} \mathbf{G}\ \mathsf{true}\ )$ and define $\varphi' := \bigwedge_j \eta_j \wedge \varphi$. In each position each η_j checks whether ψ_j holds and if yes, stores the complete current valuation in the register. When translated into an automaton, the thread corresponding to the $\mathbf{G}\,\mathsf{true}$ subformula is forced to run until the end of the word, verifying a tautology in every position and successfully terminating at the end of the word, so this construction forces the creation and accumulation of according threads in the configuration, storing these values at each position that satisfies the corresponding ψ_j. Restriction of the valuation to some specific attribute $x \in [k]$ is not necessary, because it does not affect the semantics and additional unnecessary values are simply ignored by definition of eq and $\overline{\mathsf{eq}}$ as well as $\uparrow^{x}$.

Let $q_{save,j}$ denote the *moving* state of a thread that has stored the valuation of a position where ψ_j held. By definition of the transition relation it is guaranteed, that for every value that is to be quantified over by some $\forall^{x}_{\leq,\psi_j}$ at some position, a thread in state $q_{save,j}$ that is storing the according value will exist in the configuration at the time the according spread-operation is executed, as it must be the last non-moving operation executed at some position, especially coming after any $\mathsf{store}(\ldots)$ operations.

Using the syntactic replacements $\overline{\mathbf{X}}$ and $\mathbf{R}$ from 2.11 we transform φ' to negation normal form, where negation only occurs in front of propositions or $\uparrow^{x}$, as the automaton only permits negation in the context of checking a single proposition and comparing valuations for equality. Then we make sure, that the formula contains no $\uparrow^{j}$ in scope of $\downarrow^{i}$ with $i < j$, as this comparison is false in the semantics of the logic. Such subformulae are replaced directly with the contradiction false. We call the modified formula $\hat{\varphi}$.

Let $\mathsf{sub}(\hat{\varphi})$ be the set of all subformulae of $\hat{\varphi}$, including the unrolling of $\xi \mathbf{U} \psi = \psi \vee (\xi \wedge (\xi \mathbf{U} \psi))$, and all their subformulae and let $\langle q \rangle$ denote the state $q \in Q$ to avoid the confusion between states and transition expressions. To take care of the weak next operator $\overline{\mathbf{X}}$, that we introduced to transform the negation before an $\mathbf{X}$ into an expression that the automaton can act on, let $Q_{\overline{\triangleright}} := \{\mathbf{X}\xi \mid \overline{\mathbf{X}}\xi \in \mathsf{sub}(\hat{\varphi})\} \cup \{\langle \overline{\triangleright} \rangle\}$.

Finally, let $\mathcal{A}_\varphi = (\Sigma, k, Q, \langle \hat{\varphi} \rangle, \delta), k \in \mathbb{N}$ be a k-NRA with $Q = Q_{\overline{\triangleright}} \cup \mathsf{sub}(\hat{\varphi})$. The transition function δ is defined as follows:

$$
\begin{aligned}
\delta(p) &:= \langle p \rangle & \delta(\neg p) &:= \langle \overline{p} \rangle \\
\delta(\xi \wedge \psi) &:= \langle \xi \rangle \wedge \langle \psi \rangle & \delta(\xi \vee \psi) &:= \langle \xi \rangle \vee \langle \psi \rangle \\
\delta(\uparrow^i) &:= \mathsf{eq}_i & \delta(\neg \uparrow^i) &:= \overline{\mathsf{eq}_i} \\
\delta(\downarrow^i \xi) &:= \mathsf{store}(\langle \xi \rangle) & \delta(\overline{\triangleright}) &:= \overline{\triangleright}? \\
\delta(\mathbf{X}\xi) &:= \triangleright \langle \xi \rangle & \delta(\overline{\mathbf{X}}\xi) &:= \langle \mathbf{X}\xi \rangle \vee \langle \overline{\triangleright} \rangle \\
\delta(\xi \mathbf{U} \psi) &:= \langle \psi \rangle \vee \langle \xi \wedge \mathbf{X}(\xi \mathbf{U} \psi) \rangle & \delta(\xi \mathbf{R} \psi) &:= \langle \psi \rangle \wedge \langle \xi \vee \overline{\mathbf{X}}(\xi \mathbf{R} \psi) \rangle \\
\delta(\exists_{\geq} \xi) &:= \mathsf{guess}(\langle \xi \rangle) & \delta(\forall^i_{\leq, \psi_j} \xi) &:= \mathsf{spread}(q_{save,j}, \langle \xi \rangle)
\end{aligned}
$$

Given this translation, it is clear that $\mathcal{A}_\varphi$ accepts exactly the models of φ.

□

The translation follows directly the described intuition of the states as to-be-evaluated subformulae and allows for a natural mapping of logical operators to expressions understood by the automaton. As you can see, some special handling is neccessary for formulae containing negations, as the automaton does not have a corresponding universal expression. It can especially not evaluate some arbitrary subformula and then go back to e.g. negate the result, being a one-way automaton and having this thread-eliminating semantics.

The fact that we can not move backwards is also the reason for the construction with the threads that run along the main thread (which is evaluating the actual formula) and collect values in all positions we are interested in, by creating a new thread in each such position that will only terminate at the end of the word. As we can not just go back and compare values, we take them with us in the form of these threads. This gives us the possibility to access these data values using spread, "forking" the according threads with a new state to verify some property for all of them.

The part of the proof describing the construction of the threads that collect the values differs in presentation from (Figueira, 2012, Proposition 4.3), as the description of the construction lacks in precision and abuses notation, assigning complex formulae to the transition function, whereas according to the definitions the described states actually need to be split up. The corresponding step here – the addition of the η_i formulae – fits better into the style of the rest of this proof, describing the same construction as an extension to the formula, before performing the actual translation from the subformulae to states. So the same effect is achieved here in a more concise and readable way.

Now it is clear, that starting with a $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ formula we can translate it down to a NRA, using the collapsing, frame encoding, linearisation and then translation described in this chapter. In figure 4.3 you can see a simple formula and satisfying data word at different stages in the series of transformations. You can see, that in many steps there is another additional term added to the formula. This is necessary to rule out the possibility, that words that previously did not satisfy the formula may do so in the transformed model. So the added correction terms are basically accommodating for the reduction or change of structure in each transformation step, always ensuring that the original meaning of the formula, and thereby the set of models, is preserved.

Finally, using all the previous work, we are able to conclude that the logic described in this thesis is indeed still decidable over tree-quasi-ordered attribute sets, by putting all the pieces together:

Theorem 4.7 (Decidability)
$\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ *is decidable if and only if* A *is a tree-quasi-ordering.*

Proof. For the only-if direction see (Decker and Thoma, 2015, Theorem 2) and observe, that $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ is a superset of the logic described there. Therefore, $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ is not decidable if A is not a tree-quasi-ordering.

The other direction follows from the linearisation described in Theorem 4.5 and translation to NRA described in Theorem 4.6. As shown in Theorem 3.20, we can decide whether an NRA is empty. Because we can construct an NRA that accepts exactly the same words that satisfy the underlying $\mathsf{LTL}^{\downarrow}_{A}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^{x}_{\leq}]$ formula, satisfiability for such formulae is decidable.

□

formula φ | data word w

$a \wedge \downarrow^{x_3} \mathbf{X}(\neg a \wedge \uparrow^{x_4})$

a	b
$x_1 \mapsto 5$ $\updownarrow$ $x_2 \mapsto 2$ $\nearrow \quad \nwarrow$ $x_3 \mapsto 7 \quad x_4 \mapsto 4$ $\uparrow$ $x_5 \mapsto 9$	$x_1 \mapsto 5$ $\updownarrow$ $x_2 \mapsto 2$ $\nearrow \quad \nwarrow$ $x_3 \mapsto 1 \quad x_4 \mapsto 7$ $\uparrow$ $x_5 \mapsto 3$

$\Downarrow$ collapsing $\Downarrow$ $\Downarrow$

$a \wedge \downarrow^{x_2} \mathbf{X}(\neg a \wedge \uparrow^{x_3}) \wedge \gamma$

a	b
$x_1 \mapsto 52$ $\nearrow \quad \nwarrow$ $x_2 \mapsto 7 \quad x_3 \mapsto 4$ $\uparrow$ $x_4 \mapsto 9$	$x_1 \mapsto 52$ $\nearrow \quad \nwarrow$ $x_2 \mapsto 1 \quad x_3 \mapsto 7$ $\uparrow$ $x_4 \mapsto 3$

$\Downarrow$ frame encoding and linearisation $\Downarrow$ $\Downarrow$

$a \wedge \downarrow^{2} \mathbf{XX}(\neg a \wedge \mathbf{X} \uparrow^{2}) \wedge \gamma \wedge \beta_1 \wedge \beta_2 \wedge \beta_3$

a	a	b	b
52	52	52	52
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
7	4	1	7
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
9	99	3	99

$\Downarrow$ translation of formula to equivalent NRA $\Downarrow$

$\mathcal{A} = (\Sigma, 3, Q, q_1, \delta), \quad \delta =$

q_i	1	2	3	4
$\delta(q_i)$	$q_2 \wedge \langle \gamma \wedge \beta_1 \wedge \beta_2 \wedge \beta_3 \rangle$	$q_3 \wedge q_4$	a	$\mathsf{store}(q_5)$

q_i	5	6	7	8	9	10	...
$\delta(q_i)$	$\triangleright q_6$	$\triangleright q_7$	$q_8 \wedge q_9$	$\overline{a}$	$\triangleright q_{10}$	eq_2	...

Figure 4.3: Series of transformations from $\mathsf{LTL}^{\downarrow}_{A}$ to k-NRA

5 Summary and Open Questions

In this thesis you have been introduced to the logic $\mathsf{LTL}_A^{\downarrow}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall_{\leq}^{x}]$ and have seen, how a tree-quasi-ordered attribute set and restricted quantification over data values can be both added to $\mathsf{LTL}_1^{\downarrow}$ in a straight-forward way, while preserving decidability.

Returning to the example given in the introduction, this means that using this logic we can express and decide statements like

$$\exists_{\geq}\mathbf{F}((\mathsf{lock} \wedge \uparrow^{\mathsf{pid}}) \wedge \neg(\mathsf{use} \wedge \uparrow^{\mathsf{res}})\mathbf{U}(\mathsf{unlock} \wedge \uparrow^{\mathsf{pid}}))$$

saying that at some point a resource is locked, but not used, or

$$\mathbf{G}(\mathsf{lock} \Rightarrow \forall_{\leq,\mathsf{lock}}^{\mathsf{pid}}(\uparrow^{\mathsf{res}} \Rightarrow \uparrow^{\mathsf{pid}}))$$

saying that each resource is always locked by the same process.

A few other questions that could be investigated from here would be:

- Lifting the logic to data trees
- Extending the logic with a linear ordering over the data values
- Analyzing the complexity of the logic

In Figueira (2012) all results are given both for data words and *data trees* (which are basically branching data words) and for reasoning about the tree variation, corresponding proofs for *alternating tree register automata* (ATRA, Jurdzinski and Lazic (2011)) are given, which can most likely be adapted to nested registers in the same way as NRA generalize the ARA. As data trees are a natural model to represent hierarchial data like XML documents, an interesting result in Figueira (2012) is that the forward fragment of *Core-Data-XPath* (Bojanczyk et al. (2009)), consisting of the operations describing forward navigation and equality comparison of data, is decidable. A generalization of $\mathsf{LTL}_A^{\downarrow}[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall_{\leq}^{x}]$ to ATRA would bring the additional expressive power of the nested

attributes into the realm of XPath query validation. Furthermore, while in some cases the same property can be expressed using either $\forall_{\leq}$ or $\exists_{\geq}$ because of the linearity of data words, in the context of data trees the distinction between those operators becomes more visible because of the branching structure of data trees – while $\exists_{\geq}$ can guess a valuation that may be satisfied by one of the upcoming branches, we can not simulate the same effect using the pseudo-look-behind given by $\forall_{\leq}$.

Next, in Figueira (2012) it is shown that decidability is preserved even if the data domain has a linear ordering that can be accessed by the logic, replacing eq and $\overline{\mathsf{eq}}$ with a more powerful set of operations $\mathsf{test}(=), \mathsf{test}(\neq), \mathsf{test}(<), \mathsf{test}(>)$. A similar generalization to an ordering over $[k]$-vectors in the nested register model would allow the addition of two new corresponding operators $\uparrow^x_<, \uparrow^x_>$ to the logic, thus extending the possibilities for valuation comparisons and making the logic applicable to problems where such comparisons are necessary. For example, one might want to verify that some variable is monotonically increasing in a non-trivial piece of code – this variable could then directly be encoded as an attribute and verified with e.g. $\mathbf{G}(\downarrow^{\mathsf{var}} \mathbf{X} \uparrow^{\mathsf{var}}_>)$.

In Decker and Thoma (2015) it has been shown, that satisfiability is non-primitive recursive and specifically that $\mathsf{LTL}^{\downarrow}_A[\mathbf{X}, \mathbf{U}]$ is $\mathbf{F}_{\varepsilon_0}$-complete in terms of *fast-growing complexity classes* (Schmitz (2013)). While decidability results are given for $\mathsf{LTL}^{\downarrow}_A[\mathbf{X}, \mathbf{U}, \exists_{\geq}, \forall^x_{\leq}]$, it is still open how the added quantifiers and the other possible extensions proposed here affect the complexity of the logic.

Bibliography

Bojanczyk, M., Muscholl, A., Schwentick, T., and Segoufin, L. (2009). Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3):1–48.

Decker, N. and Thoma, D. (2015). On freeze LTL with ordered attributes. *CoRR*, abs/1504.06355.

Demri, S., Figueira, D., and Praveen, M. (2013). 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.

Demri, S. and Lazic, R. (2009). LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3).

Demri, S., Lazic, R., and Nowak, D. (2005). 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.

Demri, S., Lazic, R., and Sangnier, A. (2008). Model checking freeze LTL over one-counter automata. In *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*, volume 4962, pages 490–504. Springer.

Dickson, L. E. (1913). Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *The American Journal of Mathematics*, 35(4):413–422.

Figueira, D. (2012). Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1):1–43.

Finkel, A. and Schnoebelen, P. (2001). Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92.

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

Higman, G. (1952). Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 2(7):326–336.

Jurdzinski, M. and Lazic, R. (2011). Alternating automata on data trees and xpath satisfiability. *ACM Transactions on Computational Logic*, 12(3):19.

Pnueli, A. (1977). The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society.

Schmitz, S. (2013). Complexity hierarchies beyond elementary. *CoRR*, abs/1312.5686.