Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences

Ying Sheng[1], Andres Nötzli[1], Andrew Reynolds[2], Yoni Zohar[3], David Dill[4], Wolfgang Grieskamp[4], Junkil Park[4], Shaz Qadeer[4], Clark Barrett[1] and Cesare Tinelli[2]

[1]Stanford University.
[2]The University of Iowa.
[3]Bar-Ilan University.
[4]Meta Novi.

Abstract

Dynamic arrays, also referred to as vectors, are fundamental data structures used in many programs. Modeling their semantics efficiently is crucial when reasoning about such programs. The theory of arrays is widely supported but is not ideal, because the number of elements is fixed (determined by its index sort) and cannot be adjusted, which is a problem, given that the length of vectors often plays an important role when reasoning about vector programs. In this paper, we propose reasoning about vectors using a theory of sequences. We introduce the theory, propose a basic calculus adapted from one for the theory of strings, and extend it to efficiently handle common vector operations. We prove that our calculus is sound and show how to construct a model when it terminates with a saturated configuration. Finally, we describe an implementation of the calculus in cvc5 and demonstrate its efficacy by evaluating it on verification conditions for smart contracts and benchmarks derived from existing array benchmarks.

1 Introduction

Generic vectors are used in many programming languages. For example, in C++'s standard library, they are provided by `std::vector`. Automated verification of software systems that manipulate vectors requires an efficient and automated way of reasoning about them. Desirable characteristics of any approach for reasoning about vectors include: (*i*) expressiveness—operations that are commonly performed on vectors should be supported; (*ii*) generality—vectors are always vectors *of elements* of some type (e.g., vectors of integers), and so it is desirable that vector reasoning be integrated within a more general framework; solvers for satisfiability modulo theories (SMT) provide such a framework and are widely used in verification tools (see [6] for a recent survey); (*iii*) efficiency—fast and efficient reasoning is essential for usability in a verification context, especially as verification tools are increasingly used by non-experts and in continuous integration.

Despite the ubiquity of vectors in software on the one hand and the effectiveness of SMT solvers for software verification on the other hand, there is not currently a clean way to represent vectors using operators from the SMT-LIB standard [3]. While the theory of arrays can be used, it is not a great fit because SMT-LIB arrays have a fixed size (possibly even infinite) determined by their index type. Representing a dynamic array thus requires additional modeling work expressed by extending an SMT problem with an axiomatization of relevant properties of vectors. This involves, for expressiveness, the use of quantified formulas, which often makes the reasoning engine less efficient and robust. Indeed, part of the motivation for this work was frustration with array-based modeling in the Move Prover, a verification framework for smart contracts [28] (see Section 7 for more information about the Move Prover and its use of vectors). The current paper bridges this gap by studying and implementing a native theory of *sequences* in the SMT framework, which satisfies the desirable properties for vector reasoning listed above.

We present two SMT-based calculi for determining satisfiability of quantifier-free formulas in the theory of sequences, and obtain solving procedures for that theory as rule application strategies for those calculi. Since the decidability of the satisfiability problem for quantifier-free formulas in even weaker theories is unknown (see, e.g., [10, 18]), we do not aim for a decision procedure. Rather, we prove model and solution soundness (entailing that, when our procedures terminate, their answers are correct). Our first calculus leverages reasoning techniques for the theory of strings from the SMT-LIB standard, which can be seen as a theory of sequences of Unicode characters. We generalize these techniques by lifting rules specific to sequences of characters to more general rules for arbitrary element types. By itself, this base calculus is already quite effective. However, it lacks rules to perform high-level vector-specific reasoning. For example, both reading from and updating a vector are very common operations in programming, and reasoning efficiently about the corresponding sequence operators is thus crucial. Our second calculus addresses this gap by integrating reasoning methods from array solvers, which handle reads and updates efficiently, into the first procedure. Notice, however, that this integration is not trivial, as it must handle novel combinations of operators (such as the combination of update and read operators with concatenation) as well as out-of-bounds cases that do not occur with SMT-LIB arrays. We have implemented both calculi in the cvc5 SMT solver [2] and evaluated them on benchmarks originating from the Move prover, as well as benchmarks that were translated from SMT-LIB array benchmarks.

Consistent with the choice to consider a theory of *generic* sequences, both of our calculi are agnostic about the sort of the elements in the sequence. However, for combination purposes, we consider here only element sorts that are infinite. This makes the theory *stably infinite*, allowing for a simple, Nelson-Oppen-style [22] combination of our procedures for it with those for other theories. More sophisticated combination methods require a stronger property, *politeness* [20, 24], which we expect to investigate in future work.

The rest of the paper is organized as follows. Section 2 includes basic notions from first-order logic. Section 3 introduces the theory of sequences and shows how it can be used to model vectors. Section 4 presents calculi for this theory. Section 5 proves that the calculi are correct for the theory. Section 6 describes the implementation of these calculi in cvc5. Section 7 presents an evaluation comparing Z3 and several variations of the sequence solver in cvc5. We conclude in Section 8 with directions for further research.

Related work: Our work crucially builds on a proposal by Bjørner et al. [9], but extends it in several key ways. First, their implementation (for a logic they call `QF_BVRE`) restricts the generality of the theory by allowing only bit-vector elements (representing characters) and assuming that sequences are bounded. In contrast, our calculus maintains full generality, allowing unbounded sequences and elements of arbitrary sort. Second, while our core calculus focuses only on a subset of the operators in [9], our implementation supports the remaining operators by reducing them to the core operators, and also adds native support for the `update` operator, which is not included in [9].

The base calculus that we present for sequences builds on similar work for the theory of strings [7, 21]. We extend our base calculus to support array-like reasoning based on the weak-equivalence approach [12]. Though there exists prior work on extending the theory of arrays with more operators and reasoning about length [1, 11, 15, 17, 19], this work does not consider many of the other sequence operators we consider here. Most notably, it does not consider concatenation.

The SMT solver Z3 [13] provides a theory solver for sequences. However, its documentation is limited [8], it does not support `update` directly, and its internal algorithms are not described in the literature. Furthermore, as we show in Section 7, the performance of the Z3 implementation is generally inferior to our implementation in cvc5.

A preliminary version of this work was published in the proceedings of IJCAR 2022 [27]. The current article extends the original version with complete proofs. Further, while [27] only sketched the model construction method (when a saturated configuration in our calculi is found), here, model construction is formalized, thoroughly described, and proven correct.

2 Preliminaries

We assume the usual notions and terminology of many-sorted first-order logic with equality (see, e.g., [16] for a complete presentation). We consider many-sorted signatures Σ, each containing a set of sort symbols (including a Boolean sort Bool), a family of logical symbols $\approx$ for equality, with sort $\sigma \times \sigma \to \mathsf{Bool}$ for all sorts σ in Σ and interpreted as the identity relation, and a set of interpreted (and sorted) function symbols. We fix a set $\mathcal{X}$ of infinitely-many variables of sort σ, for each sort σ of Σ, and adopt the usual definitions of well-sorted terms with variables from $\mathcal{X}$, and of well-sorted literals and formulas as terms of sort Bool. Given a set of terms S, we write $\mathcal{T}(S)$ to denote the set of all subterms of S. A literal is *flat* if it has the form $\bot$, $p(x_1, \ldots, x_n)$, $\neg p(x_1, \ldots, x_n)$, $x \approx y$, $\neg x \approx y$, or $x \approx f(x_1, \ldots, x_n)$, where p and f are function symbols and x, y, and $x_1, \ldots, x_n$ are variables. By convention and unless otherwise stated, we use letters w, x, y, z to denote variables and s, t, u, v to denote terms.

A Σ-interpretation $\mathcal{M}$ is defined as usual and assigns: false to $\mathcal{M}(\bot)$; a set $\mathcal{M}(\sigma)$ to every sort σ of Σ; a function $\mathcal{M}(f) : \mathcal{M}(\sigma_1) \times \ldots \times \mathcal{M}(\sigma_n) \to \mathcal{M}(\sigma)$ to every function symbol f of Σ with arity $\sigma_1 \times \ldots \times \sigma_n \to \sigma$; and an element $\mathcal{M}(x) \in \mathcal{M}(\sigma)$ to every variable x of sort σ. The satisfaction relation between interpretations and formulas is defined as usual and is denoted by $\models$.

A *theory* is a pair $T = (\Sigma, \mathbf{I})$, in which Σ is a signature and $\mathbf{I}$ is a class of Σ-interpretations, closed under variable reassignment. A Σ-formula φ is *satisfiable* (resp., *unsatisfiable*) *in* T, or T*-(un)satisfiable*, if it is satisfied by some (resp., no) interpretation in $\mathbf{I}$. Two Σ-formulas φ and ψ are T*-equisatisfiable* if φ is T-satisfiable iff ψ is T-satisfiable. For a theory $T = (\Sigma, \mathbf{I})$, a set Γ of Σ-formulas and a Σ-formula φ, we say that Γ *entails* φ *in* T (or T*-entails* φ) and write $\Gamma \models_T \varphi$ if every interpretation $\mathcal{M} \in \mathbf{I}$ that satisfies every formula of Γ also satisfies φ. We write just $\models_T \varphi$ when Γ is empty. For a signature Σ, the *empty theory of* Σ, also known as the theory of uninterpreted functions (UF), is $(\Sigma, \mathbf{I})$, where $\mathbf{I}$ is the class of all Σ-interpretations. We often drop Σ when it is clear from the context.

The theory $T_{\mathsf{LIA}} = (\Sigma_{\mathsf{LIA}}, \mathbf{I}_{T_{\mathsf{LIA}}})$ of *linear integer arithmetic* is based on the signature Σ_{LIA} that includes a single sort Int, all natural numbers as constant symbols, the unary $-$ symbol, the binary $+$ symbol, and the binary $\leq$ relation, all with the expected rank. For $k \in \mathbb{N}$, we use the notation $k \cdot x$, inductively defined by $0 \cdot x = 0$ and $(m+1) \cdot x = x + m \cdot x$. In turn, $\mathbf{I}_{T_{\mathsf{LIA}}}$ consists of all interpretations $\mathcal{M}$ for Σ_{LIA} in which the domain $\mathcal{M}(\mathsf{Int})$ is the set of integer numbers, for every constant symbol $n \in \mathbb{N}$, $\mathcal{M}(n) = n$, and $+$, $-$, and $\leq$ are interpreted as usual. We use standard notation for integer intervals (e.g., $[a, b]$ for the set of integers i, where $a \leq i \leq b$ and $[a, b)$ for the set where $a \leq i < b$).

3 A Theory of Sequences

In this section, we define the theory T_{Seq} of sequences. Its signature Σ_{Seq} is given in Figure 1. It includes the sorts Seq, Elem, Int, and Bool, intuitively denoting sequences, sequence elements, integers, and Booleans, respectively. The first

Symbol	**Arity**	**SMT-LIB**	**Description**
n	Int	n	All constants $\mathtt{n} \in \mathbb{N}$
$+$	$\mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$	`+`	Integer addition
$-$	$\mathsf{Int} \to \mathsf{Int}$	`-`	Integer negation
$\leq$	$\mathsf{Int} \times \mathsf{Int} \to \mathsf{Bool}$	`<=`	Integer inequality
ϵ	Seq	`seq.empty`	The empty sequence
unit	$\mathsf{Elem} \to \mathsf{Seq}$	`seq.unit`	Sequence constructor
$\lvert\_\rvert$	$\mathsf{Seq} \to \mathsf{Int}$	`seq.len`	Sequence length
nth	$\mathsf{Seq} \times \mathsf{Int} \to \mathsf{Elem}$	`seq.nth`	Element access
update	$\mathsf{Seq} \times \mathsf{Int} \times \mathsf{Elem} \to \mathsf{Seq}$	`seq.update`	Element update
$\mathsf{extract}$	$\mathsf{Seq} \times \mathsf{Int} \times \mathsf{Int} \to \mathsf{Seq}$	`seq.extract`	Extraction (subsequence)
$\_ \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} \_$	$\mathsf{Seq} \times \cdots \times \mathsf{Seq} \to \mathsf{Seq}$	`seq.concat`	Concatenation

Fig. 1: Signature for the theory of sequences.

four lines include symbols of Σ_{LIA}. We write $t_1 \bowtie t_2$, with $\bowtie \in \{>, <, \geq\}$, as syntactic sugar for the equivalent literal expressed using $\leq$ and possibly $\neg$. For example, $x < y$ is expressed as $\neg(y \leq x)$. The sequence symbols are given on the remaining lines, using mixfix notation for the length and concatenation operators. Figure 1 also provides the *arity* of each function symbol, that is, the number and sorts of its input arguments (if any) and the sort of its result. Notice that $\_ \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} \_$ is a variadic function symbol; we require that it takes at least two arguments.

3.1 Semantics

The theory T_{Seq} consists of all the Σ_{Seq}-interpretations that interpret:

- Int as the set of all integers;
- Elem as some non-empty set E;
- Seq as the set of finite sequences whose elements are from E, that is, the set E^* of all words over the alphabet E;
- each numeral as the corresponding integer;
- $+$,$-$, and $\leq$ as integer addition, negation, and comparison, respectively;
- ϵ as the empty sequence;
- unit as the function that maps every element of E to the sequence containing only that element;
- $\lvert\_\rvert$ as the function len that maps every sequence of E^* to its length;
- nth as a function that maps every sequence $s \in E^*$ and integer n to the n-th element of s when n *is in bounds*, i.e., $0 \leq n < \mathrm{len}(s)$,[1] and to an arbitrary integer when n is out of bounds;
- update as the function that maps every sequence $s \in E^*$, integer i, and element $e \in E$ to s itself if i is out of bounds and to the sequence obtained from s by replacing its i-th element by e if i is in bounds;

[1]We number elements in a sequence starting at 0.

- $\mathsf{extract}$ as the function that maps every sequence $s \in E^*$ and integers i and l to the maximal sub-sequence of s starting at index i and having length at most l if both i and l are non-negative and $i < \mathrm{len}(x)$;[2] the function returns the empty sequence in all other cases;
- $\_ \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} \_$ as the function that maps two or more sequences $s_1, s_2, \ldots, s_n$ of E^* to their concatenation $s_1 s_2 \cdots s_n$.

Notice that the interpretations of Elem and nth are not completely fixed by the theory: different models of T_{Seq} may differ on the set they associate with Elem and on the value returned by nth when its second argument is out of bounds. Also notice that T_{Seq} is a *conservative extension* of the theory T_{LIA} of linear integer arithmetic introduced earlier, that is, every Σ_{LIA}-formula is T_{Seq}-satisfiable iff it is T_{LIA}-satisfiable.

3.2 Vectors as Sequences

We show the applicability of T_{Seq} by using it for a simple verification task. Consider the C++ function `swap` at the top of Figure 2. This function swaps two elements in a vector. The comments above the function include a partial specification for it: if both indexes are in bounds and the indexed elements are equal, then the function should not change the vector (this is expressed by `s_out == s`). We now consider how to encode the verification condition induced by the code and the specification. The function variables a, b, i, and j can be encoded as variables of sort Int with the same names. We include two copies of s: s for its value at the beginning, and s_{out} for its value at the end. But what should the sorts of s and s_{out} be? In the following examples, we consider two options, one based on arrays (Example 1) and the other on sequences (Example 2). Figure 2 summarizes the encodings of the two alternatives.

Example 1 (Arrays) *The theory of arrays includes three sorts: index, element (in this case, both are* Int*), and an array sort* Arr*, as well as two operators:* $x[i]$*, interpreted as the* i*th element of* x*; and* $x[i \leftarrow a]$*, interpreted as the array obtained from* x *by setting the element at index* i *to* a*. The variables and formulas used for this example are given on the left-hand side of the table of in Figure 2. We declare* s *and* s_{out} *as variables of an uninterpreted sort* Vec *and declare two functions* ℓ *and* $\mathbf{c}$*, which, given* v *of sort* Vec*, return its length (of sort* Int*) and content (of sort* Arr*), respectively.*[3]

Next, we introduce functions to model vector operations: $\approx_{\mathbf{A}}$ *for comparing vectors,* $\mathsf{nth}_{\mathbf{A}}$ *for reading from them, and* $\mathsf{update}_{\mathbf{A}}$ *for updating them. These functions need to be axiomatized. We include two axioms (see bottom of Figure 2):* Ax_1 *states that two vectors are equal iff they have the same length and content.* Ax_2 *axiomatizes the update operator relying on the definition of array updates: the updated vector has the same length as the original one, and if the update index* i *is in bounds, the updated content has the update value at index* i *and is otherwise identical to the original content. These axioms are not meant to be complete but are strong enough for the example.*

[2] In Bjørner et al.[9], the second argument j denotes the end index, while here it denotes the length of the sub-sequence, in order to be consistent with the theory of strings in the SMT-LIB standard.

[3] It is possible to obtain a similar encoding using the theory of datatypes (see, e.g., [5]); however, here we use uninterpreted functions which are simpler and more widely supported by SMT solvers.

```
// @pre:  0 <= i,j < s.size() and s[i] == s[j]
// @post: s_out == s
void swap(std::vector<int>& s, int i, int j) {
  int a = s[i];
  int b = s[j];
  s[i] = b;
  s[j] = a;
}
```

	Arrays	**Sequences**
Problem Variables	$a, b, i, j : \mathsf{Int} \quad s, s_{\text{out}} : \mathsf{Vec}$	$a, b, i, j : \mathsf{Int} \quad s, s_{\text{out}} : \mathsf{Seq}$
Auxiliary Variables	$\ell : \mathsf{Vec} \to \mathsf{Int} \quad \mathbf{c} : \mathsf{Vec} \to \mathsf{Arr}$ $\approx_{\mathbf{A}} : \mathsf{Vec} \times \mathsf{Vec} \to \mathsf{Bool}$ $\mathsf{nth}_{\mathbf{A}} : \mathsf{Vec} \times \mathsf{Int} \to \mathsf{Int}$ $\mathsf{update}_{\mathbf{A}} : \mathsf{Vec} \times \mathsf{Int} \times \mathsf{Int} \to \mathsf{Vec}$	
Axioms	$Ax_1 \wedge Ax_2$	
Program	$a \approx \mathsf{nth}_{\mathbf{A}}(s, i) \wedge b \approx \mathsf{nth}_{\mathbf{A}}(s, j)$ $s_{\text{out}} \approx_{\mathbf{A}} \mathsf{update}_{\mathbf{A}}(\mathsf{update}_{\mathbf{A}}(s, i, b), j, a)$	$a \approx \mathsf{nth}(s, i) \wedge b \approx \mathsf{nth}(s, j)$ $s_{\text{out}} \approx \mathsf{update}(\mathsf{update}(s, i, b), j, a)$
Spec.	$0 \leq i, j < \ell(s) \wedge \mathsf{nth}_{\mathbf{A}}(s, i) \approx \mathsf{nth}_{\mathbf{A}}(s, j)$ $\neg(s_{\text{out}} \approx_{\mathbf{A}} s)$	$0 \leq i, j < \lvert s \rvert \wedge \mathsf{nth}(s, i) \approx \mathsf{nth}(s, j)$ $\neg(s_{\text{out}} \approx s)$

$$
\boxed{
\begin{aligned}
Ax_1 &:= \forall\, x, y : \mathsf{Vec}.\ x \approx_{\mathbf{A}} y \Leftrightarrow (\ell(x) \approx \ell(y) \wedge \forall\, i : \mathsf{Int}.\ 0 \leq i < \ell(x) \Rightarrow \mathbf{c}(x)[i] \approx \mathbf{c}(y)[i]) \\
Ax_2 &:= \forall\, x, y : \mathsf{Vec}.\ \forall\, i, a : \mathsf{Int}. \\
&\qquad y \approx_{\mathbf{A}} \mathsf{update}_{\mathbf{A}}(x, i, a) \Rightarrow (\ell(x) \approx \ell(y) \wedge (0 \leq i < \ell(x) \Rightarrow \mathbf{c}(y) \approx \mathbf{c}(x)[i \leftarrow a]))
\end{aligned}
}
$$

Fig. 2: An example using T_{Seq}.

The first two lines of the `swap` *function are encoded as equalities using* $\mathsf{nth}_{\mathbf{A}}$*, and the last two lines are combined into one nested constraint using* $\mathsf{update}_{\mathbf{A}}$*. The precondition of the specification is naturally modeled using* $\mathsf{nth}_{\mathbf{A}}$*, and the post-condition is negated, so that the unsatisfiability of the formula entails the correctness of the function w.r.t. the specification. Indeed, the conjunction of all formulas in this encoding is unsatisfiable in the combined theories of arrays, integers, and uninterpreted functions.*

The above encoding, while being good enough to prove the verification condition, has two main shortcomings: it introduces auxiliary symbols and it uses quantifiers, reducing clarity and efficiency. The next example illustrates how the theory of sequences allows for a more natural and succinct encoding.

Example 2 (Sequences) *In the sequences encoding, given in the right-hand side of the table in Figure 2,* s *and* s_{out} *have sort* Seq*. No auxiliary sorts or functions are needed, as the theory symbols can be used directly. Further, these symbols do not need to be axiomatized as their semantics is fixed by the theory. The resulting formula is much shorter than in Example 1 and has no quantifiers. It is also unsatisfiable in* T_{Seq} *and can be proven so with our calculus.*

4 Calculi

After introducing some definitions and assumptions (Section 4.1), we describe a basic calculus for the theory of sequences, which adapts techniques from previous procedures for the theory of strings (Section 4.2). In particular, the basic calculus reduces the operators nth and update by introducing concatenation terms. We then show how to extend that calculus with additional rules inspired by solvers for the theory of arrays (Section 4.3); the modified calculus can often reason about nth and update terms directly, avoiding the introduction of concatenation terms (which are typically expensive to reason about).

4.1 Basic Definitions

For conciseness, we use a vector of sequence terms $\bar{\boldsymbol{t}} = (t_1, \ldots, t_n)$ to denote the term corresponding to the concatenation of $t_1, \ldots, t_n$. More precisely, $\bar{\boldsymbol{t}}$ denotes ϵ if $n = 0$, denotes t_1 if $n = 1$, and denotes $t_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} t_n$ otherwise.

Definition 1 *A Σ_{Seq}-formula φ is a sequence constraint if it has the form $s \approx t$ or $s \not\approx t$. It is an arithmetic constraint if*
1. it has the form $s \approx t$, $s \geq t$, $s \not\approx t$, or $s < t$ where s, t are terms of sort Int; or
2. it is a disjunction $c_1 \vee c_2$ of two arithmetic constraints.

Notice that sequence constraints do not have to contain sequence terms (e.g., $x \approx y$ where x, y are Elem-variables). Also, equalities and disequalities between terms of sort Int are both sequence and arithmetic constraints. In this paper we focus on sequence constraints and arithmetic constraints. This is justified by the following lemma.

Lemma 1 *For every quantifier-free Σ_{Seq}-formula φ, there are sets $\mathsf{S}_1, \ldots, \mathsf{S}_n$ of sequence constraints and sets $\mathsf{A}_1, \ldots, \mathsf{A}_n$ of arithmetic constraints such that: (i) φ is T_{Seq}-satisfiable iff $\mathsf{S}_i \cup \mathsf{A}_i$ is T_{Seq}-satisfiable for some $i \in [1, n]$; and (ii) for every T_{Seq}-interpretation $\mathcal{M}$ and $i \in [1, n]$, if $\mathcal{M} \models \mathsf{S}_i \cup \mathsf{A}_i$, then $\mathcal{M} \models \varphi$.*

Proof Using standard transformations, φ can be transformed into a disjunction $\varphi' = \varphi_1 \vee \ldots \vee \varphi_n$, where for each $i \in [1, n]$, φ_i is a conjunction of flat literals $\varphi_i^1, \ldots, \varphi_i^{n_i}$, such that (i) φ and φ' are T_{Seq}-equisatisfiable; and (ii) for every T_{Seq}-interpretation $\mathcal{M}$, if $\mathcal{M} \models \varphi'$, then $\mathcal{M} \models \varphi$. Each flat literal in φ_i is either a sequence constraint or an arithmetic constraint. For each $i \in [1, n]$ we set $\mathsf{S}_i = \{\varphi_i^j \mid$ such that φ_i^j is a sequence constraint $\}$ and $\mathsf{A}_i = \{\varphi_i^j \mid$ such that φ_i^j is an arithmetic constraint $\}$. Both (i) and (ii) follow easily. $\square$

We present the calculi with the following simplifying assumptions.

Assumption 1 *Whenever we refer to a set S of sequence constraints, we assume:*
1. for every non-variable term $t \in \mathcal{T}(\mathsf{S})$, there exists a variable x such that $x \approx t \in \mathsf{S}$;
2. for every variable x of sort Seq, there exists a variable ℓ_x such that $\ell_x \approx |x| \in \mathsf{S}$;

$$
\begin{array}{ll}
|\epsilon| \to 0 & |\mathsf{unit}(t)| \to 1 \\
|\mathsf{update}(s,i,t)| \to |s| & |s_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} s_n| \to |s_1| + \cdots + |s_n| \\
& \\
\overline{\boldsymbol{u}} \mathbin{+\!\!+} \epsilon \mathbin{+\!\!+} \overline{\boldsymbol{v}} \to \overline{\boldsymbol{u}} \mathbin{+\!\!+} \overline{\boldsymbol{v}} & \overline{\boldsymbol{u}} \mathbin{+\!\!+} (s_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} s_n) \mathbin{+\!\!+} \overline{\boldsymbol{v}} \to \overline{\boldsymbol{u}} \mathbin{+\!\!+} s_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} s_n \mathbin{+\!\!+} \overline{\boldsymbol{v}}
\end{array}
$$

Fig. 3: Rewrite rules for the reduced form $t{\downarrow}$ of a term t, obtained from t by applying these rules to completion.

3. all literals in S *are flat.*
Whenever we refer to a set of arithmetic constraints, we assume all its literals are flat.

These assumptions are without loss of generality as any set can be easily transformed into a T_{Seq}-equisatisfiable set satisfying the assumptions by the addition of fresh variables and equalities. Some of the derivation rules we introduce later, which operate on a set S of sequence constraints and a set A of arithmetic constraints, introduce non-flat literals in those sets. In such cases, we assume that similar transformations are done immediately after applying the rule to maintain the invariant that all literals in $\mathsf{S} \cup \mathsf{A}$ are flat. Similarly, rules may introduce arithmetic literals, such as $\ell_x > 0$, that are not arithmetic constraints according to Definition 1. Every one of those literals, however, can be converted to an equivalent set of arithmetic constraints, and so we assume that they are, to guarantee that A remains a set of arithmetic constraints. Rules may also introduce fresh variables k of sort Seq. We further assume for brevity that in such cases, a corresponding constraint $\ell_k \approx |k|$ is added to S, where ℓ_k is a fresh variable of sort Int.

Definition 2 *Let* C *be a set of constraints. We write* $\mathsf{C} \models \varphi$ *to denote that* C *entails formula* φ *in the empty theory, and write* $\equiv_{\mathsf{C}}$ *to denote the binary relation over* $\mathcal{T}(\mathsf{C})$ *such that* $s \equiv_{\mathsf{C}} t$ *iff* $\mathsf{C} \models s \approx t$.

Lemma 2 *For every set* S *of sequence constraints,* $\equiv_{\mathsf{S}}$ *is an equivalence relation; furthermore, every equivalence class of* $\equiv_{\mathsf{S}}$ *contains at least one variable.*

We denote the equivalence class of a term s according to $\equiv_{\mathsf{S}}$ by $[s]_{\equiv_{\mathsf{S}}}$ and drop the $\equiv_{\mathsf{S}}$ subscript when it is clear from the context.

In the presentation of the calculus, it will often be useful to normalize terms to what we call a *reduced form.*

Definition 3 *Let* t *be a* Σ_{Seq}*-term. The* reduced form *of* t*, denoted by* $t{\downarrow}$*, is the term obtained by applying to it the rewrite rules listed in Figure 3 to completion.*

Observe that $t{\downarrow}$ is well defined because the given rewrite rules form a terminating and confluent rewrite system. Termination can be seen by noting that each rule reduces the number of applications of sequence operators in the left-hand side term or keeps that number the same but reduces the size of the term. It is confluent because the reduction rules for length constraints have no ambiguity, and the last two rules about concatenation have a unique final form, which is the result of removing all the empty sequences and nested concatenations. It is not difficult to show that $\models_{T_{\mathsf{Seq}}} t \approx t{\downarrow}$.

We now introduce some basic definitions related to concatenation terms. The goal is to be able to define when such terms are made up of basic building blocks that cannot be further decomposed.

Definition 4 *A concatenation term is a term of the form $s_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} s_n$ with $n \geq 2$. If each s_i is a variable, it is a variable concatenation term. For a set S of sequence constraints, a variable concatenation term $x_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} x_n$ is singular in S if $\mathsf{S} \not\models x_i \approx \epsilon$ for at most one variable x_i with $i \in [1, n]$. A sequence variable x is atomic in S if $\mathsf{S} \not\models x \approx \epsilon$ and for all variable concatenation terms $s \in \mathcal{T}(\mathsf{S})$ such that $\mathsf{S} \models x \approx s$, s is singular in S.*

Intuitively, a variable concatenation term is singular if it contains at most one variable that is not equivalent to the empty sequence; a sequence variable is atomic if it is inequivalent to the empty sequence and every concatenation term it is equivalent to is singular. Thus, an atomic variable cannot be further decomposed into a concatenation of more than one (non-empty) term.

We now lift the concept of atomic variables to representatives of equivalence classes.

Definition 5 *Let S be a set of sequence constraints. Assume a choice function $\alpha : \mathcal{T}(\mathsf{S})/{\equiv_{\mathsf{S}}} \rightarrow \mathcal{T}(\mathsf{S})$ that chooses a variable from each equivalence class of $\equiv_{\mathsf{S}}$. A sequence variable x is an atomic representative in S if it is atomic in S and $x = \alpha([x]_{\equiv_{\mathsf{S}}})$.*

Finally, we define a relation between a sequence variable and its furthest expansion into concatenations. For example, from $x \approx x_1 \mathbin{+\!\!+} u$ and $u \approx x_2 \mathbin{+\!\!+} x_3$, we can expand x to get $x \approx x_1 \mathbin{+\!\!+} x_2 \mathbin{+\!\!+} x_3$. Informally, under the right conditions on a set of atoms of the form $x \approx x_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} x_n$, we can apply such expansions to completion and obtain a unique representation for each sequence variable which we can then treat as the variable's normal form (defined formally in Lemma 6 in the next section).

Definition 6 *Let S be a set of sequence constraints. We inductively define a relation $\mathsf{S} \models_{+\!\!+} x \approx s$, where x is a sequence variable in S and s is a sequence term whose variables are in $\mathcal{T}(\mathsf{S})$, as follows:*

1. *$\mathsf{S} \models_{+\!\!+} x \approx x$ for all sequence variables $x \in \mathcal{T}(\mathsf{S})$.*
2. *$\mathsf{S} \models_{+\!\!+} x \approx t$ for all sequence variables $x \in \mathcal{T}(\mathsf{S})$ and variable concatenation terms t, where $x \approx t \in \mathsf{S}$.*
3. *If $\mathsf{S} \models_{+\!\!+} x \approx (\overline{\boldsymbol{w}} \mathbin{+\!\!+} y \mathbin{+\!\!+} \overline{\boldsymbol{z}}){\downarrow}$ and $\mathsf{S} \models y \approx t$ and t is ϵ or a variable concatenation term in S that is not singular in S, then $\mathsf{S} \models_{+\!\!+} x \approx (\overline{\boldsymbol{w}} \mathbin{+\!\!+} t \mathbin{+\!\!+} \overline{\boldsymbol{z}}){\downarrow}$.*

Let α be a choice function for S *as defined in Definition 5. We additionally define the entailment relation* $\mathsf{S} \models^*_{++} x \approx \overline{\boldsymbol{y}}$*, with* $\overline{\boldsymbol{y}} = (y_1, \ldots, y_n)$ $(n \geq 0)$*, to hold if each element of* $\overline{\boldsymbol{y}}$ *is an atomic representative in* S *and there exists* $\overline{\boldsymbol{z}}$ *of length n such that* $\mathsf{S} \models_{++} x \approx \overline{\boldsymbol{z}}$ *and* $\mathsf{S} \models y_i \approx z_i$ *for* $i \in [1, n]$.

In other words, $\mathsf{S} \models^*_{++} x \approx t$ holds when t is a concatenation of atomic representatives and is entailed to be equal to x by S. In practice, t is determined by recursively expanding concatenations using equalities in S up to a fixpoint. Note that our rules ensure that cyclic equations (i.e., of the form $x \approx t[x]$) either collapse into $x \approx x$ or create an inconsistency in the arithmetic theory (see Lemma 4).

Example 3 *Suppose* $\mathsf{S} = \{x \approx y +\!\!+ z, y \approx w +\!\!+ u, u \approx v\}$ *(we omit the additional constraints required by Assumption 1, part 2 for brevity). It is easy to see that u, v, w, and z are atomic in* S*, but x and y are not. Furthermore, w and z (and one of u or v) must also be atomic representatives. By Item 2 of Definition 6, we have* $\mathsf{S} \models_{++} x \approx y +\!\!+ z$*. Then, since* $\mathsf{S} \models y \approx w +\!\!+ u$ *and* $w +\!\!+ u$ *is a variable concatenation term not singular in* S*, we get that* $\mathsf{S} \models_{++} x \approx ((w +\!\!+ u) +\!\!+ z){\downarrow}$*, and so* $\mathsf{S} \models_{++} x \approx w +\!\!+ u +\!\!+ z$*. Now, assume that* $v = \alpha([v]_{\equiv_\mathsf{S}}) = \alpha(\{v, u\})$*. Then,* $\mathsf{S} \models^*_{++} x \approx w +\!\!+ v +\!\!+ z$.

Our calculi can be understood as modeling abstractly a cooperation between an *arithmetic subsolver* and a *sequence subsolver*. Many of the derivation rules in these calculi lift those in the string calculus of Liang et al. [21] to sequences of elements of an arbitrary sort. We describe them similarly as rules that modify *configurations*.

Definition 7 *A configuration is either the distinguished configuration* unsat *or a pair* (S, A) *of a set* S *of sequence constraints and a set* A *of arithmetic constraints.*

The derivation rules are given in *guarded assignment form*, where the rule premises describe the conditions on the current configuration under which the rule can be applied, and the conclusion is either unsat, or otherwise describes the resulting modifications to the configuration, with a syntax of the form X, y abbreviating $X \cup \{y\}$. A rule may have multiple alternative conclusions separated by $\|$. In the rules, some of the premises have the form $\mathsf{S} \models s \approx t$ (see Definition 2) or $\mathsf{S} \models_{\mathsf{LIA}} s \approx t$ where $\models_{\mathsf{LIA}}$ abbreviates $\models_{T_{\mathsf{LIA}}}$. The former entailment can be checked with standard algorithms for congruence closure, and the latter can be checked by solvers for linear integer arithmetic.

An application of a rule is *redundant* if it has a conclusion that is not unsat and where each component in the derived configuration is a subset of the corresponding component in the premise configuration. We assume that for rules that introduce fresh variables, the introduced variables are identical whenever the premises triggering the rule are the same (i.e., we cannot generate an infinite sequence of rule applications by continuously using the same premises to introduce fresh variables).[4] A configuration other than unsat is *saturated* with respect to a set R of derivation rules if every possible application of a rule in R to it is redundant. Notice that, in particular, the rules A-Conf and S-Conf cannot be applied to a configuration that is saturated with respect to those rules, as this would result in unsat and would thus not be redundant.

A *derivation tree* is a tree where each node is a configuration and its children, if any, are obtained by a non-redundant application of a rule of the calculus. A derivation tree is *closed* if all of its leaves are unsat. As we show later, a closed derivation tree with root node (S, A) is a proof that $\mathsf{A} \cup \mathsf{S}$ is unsatisfiable in T_{Seq}. In contrast, a derivation tree with root node (S, A) and a saturated leaf with respect to all the rules of the calculus is a witness that $\mathsf{A} \cup \mathsf{S}$ is satisfiable in T_{Seq}.

Our two calculi are built out of the derivation rules listed in Figures 4 to 6. Based on what we said above about the implicit postprocessing of the sets of constraints derived by those rules, one can prove that all of them transform configurations to configurations, leading to the following lemma which we will implicitly rely on when proving later results.

Lemma 3 *For every rule from Figures 4 to 6 that does not derive* unsat*, if* S' *and* A' *are the sets resulting from applying the rule to a configuration* (S, A)*, then* $(\mathsf{S}', \mathsf{A}')$ *is a configuration as well.*

A configuration (S, A) is *satisfied* by an interpretation if $\mathsf{S} \cup \mathsf{A}$ is satisfied by that interpretation. In contrast, the configuration unsat is satisfied by no interpretation. A derivation rule is *sound* if for every model of T_{Seq} that satisfies the configuration in the rule's premise there is one that satisfies one of the configurations in the rule's conclusion.

4.2 Core Calculus

We now present the first calculus for solving T_{Seq}-formulas.

Definition 8 *The calculus* BASE *consists of the derivation rules in Figures 4 and 5.*

Some of the rules are adapted from previous work on string solvers [21, 26]. Compared to that work, our presentation of the rules is noticeably simpler, due to our use of the relation $\models^*_{++}$ from Definition 6. In particular, our configurations consist only of pairs of sets of formulas, without any auxiliary data-structures.

The rules in Figure 4 form the core of the calculus. For greater clarity, some of the conclusions of the rules include terms before they are flattened. First, either subsolver can report that the current set of constraints is unsatisfiable by using the rules A-Conf or S-Conf. The latter corresponds to a situation where

[4]In practice, this is implemented by associating each introduced variable with a *witness term* as described in Reynolds et al. [25].

$$\textsf{A-Conf}\ \frac{\mathsf{A} \models_{\mathsf{LIA}} \bot}{\mathsf{unsat}} \qquad \textsf{A-Prop}\ \frac{\mathsf{A} \models_{\mathsf{LIA}} s \approx t \qquad s,t \in \mathcal{T}(\mathsf{S})}{\mathsf{S} := \mathsf{S}, s \approx t}$$

$$\textsf{S-Conf}\ \frac{\mathsf{S} \models \bot}{\mathsf{unsat}} \qquad \textsf{S-Prop}\ \frac{\mathsf{S} \models s \approx t \qquad s,t \in \mathcal{T}(\mathsf{S}) \qquad s,t \text{ are } \Sigma_{\mathsf{LIA}}\text{-terms}}{\mathsf{A} := \mathsf{A}, s \approx t}$$

$$\textsf{S-A}\ \frac{x,y \in \mathcal{T}(\mathsf{S}) \cap \mathcal{T}(\mathsf{A}) \qquad x,y : \mathsf{Int}}{\mathsf{A} := \mathsf{A}, x \approx y \quad \| \quad \mathsf{A} := \mathsf{A}, x \not\approx y}$$

$$\textsf{L-Intro}\ \frac{s \in \mathcal{T}(\mathsf{S}) \qquad s : \mathsf{Seq}}{\mathsf{S} := \mathsf{S}, |s| \approx (|s|){\downarrow}} \qquad \textsf{L-Valid}\ \frac{x \in \mathcal{T}(\mathsf{S}) \qquad x : \mathsf{Seq}}{\mathsf{S} := \mathsf{S}, x \approx \epsilon \quad \| \quad \mathsf{A} := \mathsf{A}, \ell_x > 0}$$

$$\textsf{U-Eq}\ \frac{\mathsf{S} \models \mathsf{unit}(x) \approx \mathsf{unit}(y)}{\mathsf{S} := \mathsf{S}, x \approx y} \qquad \textsf{C-Eq}\ \frac{\mathsf{S} \models^*_{++} x \approx \overline{\boldsymbol{z}} \qquad \mathsf{S} \models^*_{++} y \approx \overline{\boldsymbol{z}}}{\mathsf{S} := \mathsf{S}, x \approx y}$$

$$\textsf{C-Split}\ \frac{\mathsf{S} \models^*_{++} x \approx (\overline{\boldsymbol{w}} \mathbin{++} y \mathbin{++} \overline{\boldsymbol{z}}){\downarrow} \qquad \mathsf{S} \models^*_{++} x \approx (\overline{\boldsymbol{w}} \mathbin{++} y' \mathbin{++} \overline{\boldsymbol{z}}'){\downarrow}}{\begin{array}{c} \mathsf{A} := \mathsf{A}, \ell_y > \ell_{y'} \quad \mathsf{S} := \mathsf{S}, y \approx y' \mathbin{++} k \quad \| \\ \mathsf{A} := \mathsf{A}, \ell_y < \ell_{y'} \quad \mathsf{S} := \mathsf{S}, y' \approx y \mathbin{++} k \quad \| \\ \mathsf{A} := \mathsf{A}, \ell_y \approx \ell_{y'} \quad \mathsf{S} := \mathsf{S}, y \approx y' \end{array}}$$

$$\textsf{Deq-Ext}\ \frac{x \not\approx y \in \mathsf{S} \qquad x,y : \mathsf{Seq}}{\begin{array}{c} \mathsf{A} := \mathsf{A}, \ell_x \not\approx \ell_y \quad \| \\ \mathsf{A} := \mathsf{A}, \ell_x \approx \ell_y, 0 \leq i, i < \ell_x \quad \mathsf{S} := \mathsf{S}, w_1 \approx \mathsf{nth}(x,i), w_2 \approx \mathsf{nth}(y,i), w_1 \not\approx w_2 \end{array}}$$

Fig. 4: The core derivation rules, where k and i denote fresh variables of sequence and integer sort, respectively, and w_1, w_2 are fresh element variables.

congruence closure detects a conflict between an equality and a disequality. The rules A-Prop, S-Prop, and S-A correspond to a form of Nelson-Oppen-style theory combination[5] of the two sub-solvers. The first two rules communicate equalities between terms, while the third guesses arrangements for shared variables of sort Int. Rule L-Intro ensures that the length term $|s|$ for each sequence term s is equal to its reduced form $(|s|){\downarrow}$. Rule L-Valid restricts sequence lengths to be non-negative, splitting on whether each sequence is empty or has a length greater than 0. Rule U-Eq captures the injectivity of the unit operator. We will introduce the definition of normal form later in Lemma 6. For now, it can be intuitively treated as a unique representation of each sequence variable introduced by $\models^*_{++}$. In view of this, rule C-Eq concludes that two sequence terms are equal if they have the same normal form. If a sequence variable has two different normal forms, rule C-Split takes the first differing components y and y' from the two normal forms and splits on their length relationship. Note that C-Split is a source of non-termination of the calculus, and in fact the only one (see, e.g., [21, 26]). Finally, rule Deq-Ext handles disequalities between sequences x and y by either asserting that their lengths are different or by choosing an index i at which they differ.

[5]Note that this goes beyond Nelson-Oppen combination because the theories T_{LIA} and T_{Seq} are not disjoint. As a consequence, the exchanged (dis)equalities are not limited to shared variables.

$$\textsf{R-Extract}\ \frac{x \approx \mathsf{extract}(y,i,j) \in \mathsf{S}}{\begin{array}{c}\mathsf{A} := \mathsf{A}, i < 0 \vee i \geq \ell_y \vee j \leq 0 \qquad \mathsf{S} := \mathsf{S}, x \approx \epsilon \quad \| \\ \mathsf{A} := \mathsf{A}, 0 \leq i < \ell_y, j > 0, \ell_k \approx i, \ell_x \approx \min(j, \ell_y - i) \\ \mathsf{S} := \mathsf{S}, y \approx k \mathbin{+\!\!+} x \mathbin{+\!\!+} k'\end{array}}$$

$$\textsf{R-Nth}\ \frac{x \approx \mathsf{nth}(y,i) \in \mathsf{S}}{\begin{array}{c}\mathsf{A} := \mathsf{A}, i < 0 \vee i \geq \ell_y \quad \| \\ \mathsf{A} := \mathsf{A}, 0 \leq i < \ell_y, \ell_k \approx i \qquad \mathsf{S} := \mathsf{S}, y \approx k \mathbin{+\!\!+} \mathsf{unit}(x) \mathbin{+\!\!+} k'\end{array}}$$

$$\textsf{R-Update}\ \frac{x \approx \mathsf{update}(y,i,z) \in \mathsf{S}}{\begin{array}{c}\mathsf{A} := \mathsf{A}, i < 0 \vee i \geq \ell_y \qquad \mathsf{S} := \mathsf{S}, x \approx y \quad \| \\ \mathsf{A} := \mathsf{A}, 0 \leq i < \ell_y, \ell_k \approx i, \ell_{k'} \approx 1 \\ \mathsf{S} := \mathsf{S}, y \approx k \mathbin{+\!\!+} k' \mathbin{+\!\!+} k'', x \approx k \mathbin{+\!\!+} \mathsf{unit}(z) \mathbin{+\!\!+} k''\end{array}}$$

Fig. 5: Reduction rules for extract, nth, and update. The rules use k, k', and k'' to denote fresh sequence variables. We write $s \approx \min(t, u)$ as an abbreviation for $s \approx t \vee s \approx u, s \leq t, s \leq u$.

Figure 5 includes a set of reduction rules for handling operators that are not directly handled by the core rules. These reduction rules capture the semantics of these operators by reduction to concatenation. Rule R-Extract splits into two cases: either the extraction uses an out-of-bounds index or a non-positive length, in which case the result is the empty sequence, or the original sequence can be described as a concatenation that includes the extracted sub-sequence. Rule R-Nth creates an equation between y and a concatenation term with $\mathsf{unit}(x)$ as one of its components, as long as i is not out of bounds. Rule R-Update considers two cases. If i is out of bounds, then the update term is equal to y. Otherwise, y is equal to a concatenation, with the middle component (k') representing the part of y that is updated. In the update term, k' is replaced by $\mathsf{unit}(z)$.

Example 4 *Consider a configuration* (S, A)*, where* S *contains the formulas* $x \approx y \mathbin{+\!\!+} z$, $z \approx v \mathbin{+\!\!+} x \mathbin{+\!\!+} w$*, and* $v \approx \mathsf{unit}(u)$*, and* A *is empty. Hence,* $\mathsf{S} \models |x| \approx |y \mathbin{+\!\!+} z|$*. By L-Intro, we have* $\mathsf{S} \models |y \mathbin{+\!\!+} z| \approx |y| + |z|$*. Together with Assumption 1, we have* $\mathsf{S} \models \ell_x \approx \ell_y + \ell_z$*, and then with S-Prop, we have* $\ell_x \approx \ell_y + \ell_z \in \mathsf{A}$*. Similarly, we can derive* $\ell_z \approx \ell_v + \ell_x + \ell_w, \ell_v \approx 1 \in \mathsf{S}$*, and so*

$$\mathsf{A} \models_{\mathsf{LIA}} \ell_z \approx 1 + \ell_y + \ell_z + \ell_w. \tag{1}$$

Notice that for any variable k *of sort* Seq*, we can apply L-Valid, L-Intro, and S-Prop to add to* A *either* $\ell_k > 0$ *or* $\ell_k = 0$*. Applying this to* y, z, w*, we have that* $\mathsf{A} \models_{\mathsf{LIA}} \bot$ *in each branch thanks to (1), and so A-Conf applies and we get* unsat*.*

Before moving forward, we provide the following helper lemma which relates concatenation terms to their lengths.

Lemma 4 *Let* S *be a set of sequence constraints and* A *a set of arithmetic constraints. Suppose* (S, A) *is saturated w.r.t. S-Prop, L-Intro and L-Valid. If* $x_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} x_n \in \mathcal{T}(\mathsf{S})$ *is a variable concatenation term of size* n *not singular in* S*, then*

1. $\mathsf{A} \models_{\mathsf{LIA}} \Sigma_{k=1}^{n} \ell_{x_k} \geq 2$; *and*
2. *for each* $m \in [1, n]$, $\mathsf{A} \models_{\mathsf{LIA}} \Sigma_{k=1}^{n} \ell_{x_k} > \ell_{x_m}$.

Proof Let $i, j \in [1, n], i \neq j$, such that $\mathsf{S} \not\models x_i \approx \epsilon$ and $\mathsf{S} \not\models x_j \approx \epsilon$. We know that $x_i, x_j \in \mathcal{T}(S)$, so by saturation of L-Valid, we have $\mathsf{A} \models \ell_{x_i} > 0$ and $\mathsf{A} \models \ell_{x_j} > 0$. Also, by saturation of L-Valid, L-Intro, and S-Prop, together with Assumption 1, we know that $\mathsf{A} \models_{\mathsf{LIA}} \ell_{x_k} \geq 0$ for $k \in [1, n]$. It follows that $\mathsf{A} \models_{\mathsf{LIA}} \Sigma_{k=1}^{n} \ell_{x_k} \geq 2$. Furthermore, for each $m \in [1, n]$, $\mathsf{A} \models_{\mathsf{LIA}} \Sigma_{k=1}^{n} \ell_{x_k} > \ell_{x_m}$. □

The calculus uses judgments of the form $\mathsf{S} \models_{++}^{*} x \approx t$. The following lemma shows that it is possible to compute whether those judgments hold.

Lemma 5 *Let* S *be a set of sequence constraints and* A *a set of arithmetic constraints. If* (S, A) *is saturated w.r.t. A-Conf, S-Prop, L-Intro and L-Valid, the problem of determining whether* $\mathsf{S} \models_{++}^{*} x \approx s$ *for given* x *and* s *is decidable.*

Proof We show that the set of pairs (x, s) for which $\mathsf{S} \models_{++} x \approx s$ is finite (it is then easy to see that the set of pairs (y, t) for which $\mathsf{S} \models_{++}^{*} y \approx t$ is also finite). Consider a tree whose root is obtained by Item 1 or Item 2 of Definition 6 where the children of a node are all possible results of applying Item 3 of Definition 6. Note that each node can have only finitely many children as there are only finitely many pairs (y, t) where y is a variable in S and t is ϵ or a variable concatenation term in $\mathcal{T}(\mathsf{S})$. Below, we show that every path in the tree is finite, from which it follows that the tree is finite. Since there are only finitely many such trees, it follows that the set of (x, s) for which $\mathsf{S} \models_{++} x \approx s$ is finite.

Define a partial order $\prec$ over terms of sort Seq in $\mathcal{T}(\mathsf{S})$, by $s \prec t$ iff $\mathsf{A} \models_{\mathsf{LIA}} \ell_{x_s} < \ell_{x_t}$ for some variables x_s and x_t such that $x_s \equiv_{\mathsf{S}} s$ and $x_t \equiv_{\mathsf{S}} t$. We show that $\prec$ is well defined. First, by Assumption 1, such x_s and x_t exist. Next, if $x_s \equiv_{\mathsf{S}} s$, $x_t \equiv_{\mathsf{S}} t$, $x'_s \equiv_{\mathsf{S}} s$ and $x'_t \equiv_{\mathsf{S}} t$, then $\mathsf{A} \models_{\mathsf{LIA}} \ell_{x_s} < \ell_{x_t}$ iff $\mathsf{A} \models_{\mathsf{LIA}} \ell_{x'_s} < \ell_{x'_t}$, since $\mathsf{S} \models x_s \approx x'_s$ and $\mathsf{S} \models x_t \approx x'_t$ by saturation of S-Prop. Finally, by saturation with respect to A-Conf, $\prec$ is irreflexive, asymmetric, and transitive. Let $\prec^*$ be the (well-founded) Dershowitz-Manna multiset ordering induced by $\prec$ [14], that is, the minimal transitive relation that satisfies: $A \prec^* B$ whenever A is obtained from B by removing a single element b, and possibly adding any finite number of elements a such that $a \prec b$ for every such new element. Now, for each term t of sort Seq in $\mathcal{T}(\mathsf{S})$, let $m(t)$ be the multiset of the terms occurring in it (the multiplicity of each element in this multiset is the number of times it occurs in t). We prove the following claim, which establishes that every path in the tree mentioned above is finite: if $\mathsf{S} \models_{++} x \approx (\overline{\boldsymbol{w}} ++ y ++ \overline{\boldsymbol{z}})\!\downarrow$, $\mathsf{S} \models y \approx t$ and t is ϵ or a variable concatenation term in S that is not singular in S, then $m((\overline{\boldsymbol{w}} ++ y ++ \overline{\boldsymbol{z}})\!\downarrow) \succ^* m((\overline{\boldsymbol{w}} ++ t ++ \overline{\boldsymbol{z}})\!\downarrow)$.

We consider two cases: t is ϵ or t is a variable concatenation term in $\mathcal{T}(\mathsf{S})$ not singular in S. In the first case, $(\overline{\boldsymbol{w}} ++ t ++ \overline{\boldsymbol{z}})\!\downarrow = (\overline{\boldsymbol{w}} ++ \overline{\boldsymbol{z}})\!\downarrow$. Note that the only role of $\downarrow$ here is to flatten nested concatenations. Hence, y is removed from the multiset of sub-terms and is not replaced by anything, so $m((\overline{\boldsymbol{w}} ++ y ++ \overline{\boldsymbol{z}})\!\downarrow) \succ^* m((\overline{\boldsymbol{w}} ++ t ++ \overline{\boldsymbol{z}})\!\downarrow)$.

In the second case, t is a variable concatenation term in S not singular in S. Let $t = t_1 ++ \ldots ++ t_n$. Now, $(\overline{\boldsymbol{w}} ++ (t_1 ++ \cdots ++ t_n) ++ \overline{\boldsymbol{z}})\!\downarrow$ is a flat concatenation in which y was removed, and $t_1, \ldots, t_n$ were added. To prove a decrease in $\prec^*$ from $m((\overline{\boldsymbol{w}} ++ y ++ \overline{\boldsymbol{z}})\!\downarrow)$ to $m((\overline{\boldsymbol{w}} ++ t ++ \overline{\boldsymbol{z}})\!\downarrow)$, we show that for every $k \in [1, n]$, $t_k \prec y$, that is, $\mathsf{A} \models_{\mathsf{LIA}} \ell_{t_k} < \ell_y$ (notice that t_k and y are variables). By Lemma 4, we know that for each $k \in [1, n]$, $\mathsf{A} \models_{\mathsf{LIA}} \Sigma_{i=1}^{n} \ell_{t_i} > \ell_{t_k}$. Since $\mathsf{S} \models y \approx t$, we have $\mathsf{S} \models |y| = |t|$. By saturation of L-Intro and S-Prop, and by Assumption 1, it follows that $\mathsf{A} \models \ell_y = \Sigma_{i=1}^{n} \ell_{t_i}$. Thus, $\mathsf{A} \models_{\mathsf{LIA}} \ell_y > \ell_{t_k}$. □

Lemma 5 assumes saturation with respect to certain rules. Accordingly, our proof strategy, described in Section 6, will ensure such saturation before attempting to apply rules relying on $\models_{++}^{*}$.

The following lemma shows that the relation $\models_{++}^{*}$ induces a normal form for each equivalence class of $\equiv_{\mathsf{S}}$.

Lemma 6 *Let* S *be a set of sequence constraints and* A *a set of arithmetic constraints. Suppose* (S, A) *is saturated w.r.t.* A-Conf, S-Prop, L-Intro, L-Valid, *and* C-Split. *Then, for every equivalence class* e *of* $\equiv_{\mathsf{S}}$ *whose terms are of sort* Seq, *there exists a unique (possibly empty)* $\overline{\boldsymbol{s}}$ *such that whenever* $\mathsf{S} \models_{++}^{*} x \approx \overline{\boldsymbol{s}}'$ *for* $x \in e$, *then* $\overline{\boldsymbol{s}}' = \overline{\boldsymbol{s}}$. *In this case, we call* $\overline{\boldsymbol{s}}$ *the* normal form *of* e *(and of* x*).*

Proof Let e be an equivalence class of $\equiv_{\mathsf{S}}$ whose terms are of sort Seq, and let $x \in e$ (we know every equivalence class has at least one variable by Lemma 2). We show existence and uniqueness of a normal form for e.

Existence: We show that for some $\overline{\boldsymbol{s}}$, $\mathsf{S} \models_{++}^{*} x \approx \overline{\boldsymbol{s}}$. Consider the tree construction of Lemma 5. If we follow some path in the tree, we will reach a node $\mathsf{S} \models_{++} x \approx \overline{\boldsymbol{t}}$ for which no children can be derived (i.e., Item 3 of Definition 6 doesn't apply). It is not hard to see that $\overline{\boldsymbol{t}} = (t_1, \dots, t_n)$, where $n \geq 0$ and each t_i is a variable for $i \in [1, n]$.

Let $\overline{\boldsymbol{s}}$ have the same length as $\overline{\boldsymbol{t}}$ and let $s_i := \alpha(t_i)$ for $i \in [1, n]$. We prove that $\mathsf{S} \models_{++}^{*} x \approx \overline{\boldsymbol{s}}$. We know that $\mathsf{S} \models_{++} x \approx \overline{\boldsymbol{t}}$. If $n = 0$, then trivially, $\mathsf{S} \models_{++}^{*} x \approx \overline{\boldsymbol{s}}$. Otherwise, for each $i \in [1, n]$, we further know that $\mathsf{S} \models t_i \approx s_i$ and $s_i = \alpha([s_i])$, so it only remains to show that s_i is atomic in S. Assume it is not. Then either $\mathsf{S} \models s_i \approx \epsilon$ or there exists a variable concatenation term $\overline{\boldsymbol{u}} \in \mathcal{T}(\mathsf{S})$ not singular in S such that $\mathsf{S} \models s_i \approx \overline{\boldsymbol{u}}$. In either case, this would imply that Item 3 of Definition 6 is applicable to $\mathsf{S} \models_{++} x \approx \overline{\boldsymbol{t}}$, contradicting our assumption.

Uniqueness: By the existence argument above, there exists some $\overline{\boldsymbol{s}}$ such that $\mathsf{S} \models_{++}^{*} x \approx \overline{\boldsymbol{s}}$. Now, suppose $\mathsf{S} \models_{++}^{*} x \approx \overline{\boldsymbol{s}}'$, and assume that $\overline{\boldsymbol{s}} \neq \overline{\boldsymbol{s}}'$. Then there must be $\overline{\boldsymbol{w}}, \overline{\boldsymbol{z}}, \overline{\boldsymbol{z}}', y, y'$, each containing only variables that are atomic representatives, such that $s = (\overline{\boldsymbol{w}} ++ y ++ \overline{\boldsymbol{z}})\!\downarrow$ and $s' = (\overline{\boldsymbol{w}} ++ y' ++ \overline{\boldsymbol{z}}')\!\downarrow$, with $y \neq y'$. By saturation w.r.t. C-Split, there are three possibilities: $\mathsf{A} \models \ell_y > \ell_{y'}$ and $y \approx y' ++ k \in \mathsf{S}$ for some k; $\mathsf{A} \models \ell_y < \ell_{y'}$ and $y' \approx y ++ k \in \mathsf{S}$ for some k; or $\mathsf{A} \models \ell_y = \ell_{y'}$ and $y \approx y' \in \mathsf{S}$. In the first case, notice that since $\mathsf{S} \models y \approx y' ++ k$, it follows that $\mathsf{S} \models |y| \approx |y' ++ k|$. Also, by saturation of L-Intro, $\mathsf{S} \models |y' ++ k| \approx |y'| + |k|$. So, $\mathsf{S} \models |y| \approx |y'| + |k|$. And, by saturation of S-Prop and Assumption 1 also $\mathsf{A} \models \ell_y \approx \ell_{y'} + \ell_k$. It follows that $\mathsf{S} \not\models k \approx \epsilon$; otherwise, we would have $\mathsf{A} \models \ell_k \approx 0$ by L-Intro and S-Prop, which together with $\mathsf{A} \models \ell_y > \ell_{y'}$ contradicts saturation of A-Conf. Also, y' is atomic and hence, $\mathsf{S} \not\models y' \approx \epsilon$. Thus, $\mathsf{S} \models y \approx y' ++ k$ but $y' ++ k$ is not singular in S as $\mathsf{S} \not\models k \approx \epsilon$ and $\mathsf{S} \not\models y' \approx \epsilon$. In particular, this means that y is not atomic, which contradicts our assumption, so the first case is impossible. The second case is analogous to the first. In the third case, we have $y \approx y' \in \mathsf{S}$, but y and y' are both equivalence class representatives, so $y = y'$, which contradicts our assumption that $y \neq y'$. □

4.3 Extended Calculus

Next, we present a variant of the BASE calculus that combines array reasoning with the core rules of that calculus and the R-Extract rule.

Definition 9 *The calculus* EXT *is comprised of the derivation rules in Figures 4 and 6, with the addition of rule R-Extract from Figure 5.*

Unlike in BASE, the rules in Figure 6 do not reduce nth and update to concatenation operations. Instead, they reason about those operators directly and handle their combination with concatenation. Rule Nth-Concat identifies the i-th element of sequence y with the corresponding element selected from its normal form (see Lemma 6). Rule Update-Concat operates similarly, applying update to all the components. Rule Update-Concat-Inv operates similarly on the updated sequence rather than on the original sequence. Rule Nth-Unit captures the semantics of nth when applied to a unit term. RuleUpdate-Unit is similar and distinguishes an update on an out-of-bounds index (different from 0) from an update within the bound. Rule Nth-Intro is meant to ensure that rules Nth-Update (explained below) and Nth-Unit (explained above) are applicable whenever an update term exists in the constraints. Rule Nth-Update captures the read-over-write axioms of arrays, adapted to consider their lengths as in Christ and Hoenicke [12]. It distinguishes three cases. In the first, the update index is out of bounds. In the second, it is not out of bounds, and the corresponding nth term accesses the same index that was updated. In the third case, the index used in the nth term is different from the updated index. Rule Update-Bound splits on two cases: either the update changes the sequence, or the sequence remains the same. Finally, rule Nth-Split introduces a case split on the equality between two sequence variables x and x' whenever they appear as arguments to nth with equivalent second arguments. This is needed to ensure that we detect all cases where the arguments of two nth terms must be equal.

5 Correctness

In this section, we prove that the calculi presented in Section 4 are correct. This is formalized by following theorem:

Theorem 1 *Let* $X \in \{\mathsf{BASE}, \mathsf{EXT}\}$ *and* $(\mathsf{S}_0, \mathsf{A}_0)$ *be a configuration. Assume without loss of generality that* A_0 *contains only arithmetic constraints that are not sequence constraints. Let* T *be a derivation tree obtained by applying the rules of* X *with* $(\mathsf{S}_0, \mathsf{A}_0)$ *as the initial configuration.*

1. *If* T *is closed, then* $\mathsf{S}_0 \cup \mathsf{A}_0$ *is* T_{Seq}*-unsatisfiable.*
2. *If* T *contains a saturated configuration* (S, A) *w.r.t. all the rules of* X*, then* $\mathsf{S} \cup \mathsf{A}$ *is* T_{Seq}*-satisfiable, and so is* $\mathsf{S}_0 \cup \mathsf{A}_0$*.*

$$\textsf{Nth-Concat}\ \frac{x \approx \mathsf{nth}(y,i) \in \mathsf{S} \qquad \mathsf{S} \models^{*}_{++} y \approx w_1 ++ \cdots ++ w_n}{\begin{array}{c}\mathsf{A} := \mathsf{A}, i < 0 \vee i \geq \ell_y \quad \| \\ \mathsf{A} := \mathsf{A}, 0 \leq i, i < \ell_{w_1} \qquad \mathsf{S} := \mathsf{S}, x \approx \mathsf{nth}(w_1, i) \quad \| \quad \ldots \quad \| \\ \mathsf{A} := \mathsf{A}, \sum_{j=1}^{n-1} \ell_{w_j} \leq i, i < \sum_{j=1}^{n} \ell_{w_j} \qquad \mathsf{S} := \mathsf{S}, x \approx \mathsf{nth}(w_n, i - \sum_{j=1}^{n-1} \ell_{w_j})\end{array}}$$

$$\textsf{Update-Concat}\ \frac{x \approx \mathsf{update}(y,i,v) \in \mathsf{S} \qquad \mathsf{S} \models^{*}_{++} y \approx w_1 ++ \cdots ++ w_n}{\begin{array}{c}\mathsf{S} := \mathsf{S}, x \approx z_1 ++ \cdots ++ z_n, \\ z_1 \approx \mathsf{update}(w_1, i, v), \ldots, z_n \approx \mathsf{update}(w_n, i - \sum_{j=1}^{n-1} \ell_{w_j}, v)\end{array}}$$

$$\textsf{Update-Concat-Inv}\ \frac{x \approx \mathsf{update}(y,i,v) \in \mathsf{S} \qquad \mathsf{S} \models^{*}_{++} x \approx w_1 ++ \cdots ++ w_n}{\begin{array}{c}\mathsf{S} := \mathsf{S}, y \approx z_1 ++ \cdots ++ z_n, \\ w_1 \approx \mathsf{update}(z_1, i, v), \ldots, w_n \approx \mathsf{update}(z_n, i - \sum_{j=1}^{n-1} \ell_{w_j}, v)\end{array}}$$

$$\textsf{Nth-Unit}\ \frac{x \approx \mathsf{nth}(y,i) \in \mathsf{S} \qquad \mathsf{S} \models y \approx \mathsf{unit}(u)}{\mathsf{A} := \mathsf{A}, i < 0 \vee i > 0 \quad \| \quad \mathsf{A} := \mathsf{A}, i \approx 0 \qquad \mathsf{S} := \mathsf{S}, x \approx u}$$

$$\textsf{Update-Unit}\ \frac{x \approx \mathsf{update}(y,i,v) \in \mathsf{S} \qquad \mathsf{S} \models y \approx \mathsf{unit}(u)}{\begin{array}{c}\mathsf{A} := \mathsf{A}, i < 0 \vee i > 0 \qquad \mathsf{S} := \mathsf{S}, x \approx \mathsf{unit}(u) \quad \| \\ \mathsf{A} := \mathsf{A}, i \approx 0 \qquad \mathsf{S} := \mathsf{S}, x \approx \mathsf{unit}(v)\end{array}}$$

$$\textsf{Nth-Intro}\ \frac{s' \approx \mathsf{update}(s,i,t) \in \mathsf{S}}{\mathsf{S} := \mathsf{S}, e \approx \mathsf{nth}(s,i), e' \approx \mathsf{nth}(s',i)}$$

$$\textsf{Nth-Update}\ \frac{\mathsf{nth}(x,j) \in \mathcal{T}(\mathsf{S}) \qquad y \approx \mathsf{update}(z,i,v) \in \mathsf{S} \qquad \mathsf{S} \models x \approx y \text{ or } \mathsf{S} \models x \approx z}{\begin{array}{c}\mathsf{A} := \mathsf{A}, j < 0 \vee j \geq \ell_x \quad \| \\ \mathsf{A} := \mathsf{A}, i \approx j, 0 \leq j, j < \ell_x \qquad \mathsf{S} := \mathsf{S}, \mathsf{nth}(y,j) \approx v \quad \| \\ \mathsf{A} := \mathsf{A}, i \not\approx j, 0 \leq j, j < \ell_x \qquad \mathsf{S} := \mathsf{S}, \mathsf{nth}(y,j) \approx \mathsf{nth}(z,j)\end{array}}$$

$$\textsf{Update-Bound}\ \frac{x \approx \mathsf{update}(y,i,v) \in \mathsf{S}}{\mathsf{A} := \mathsf{A}, 0 \leq i, i < \ell_y \qquad \mathsf{S} := \mathsf{S}, \mathsf{nth}(y,i) \not\approx v \quad \| \quad \mathsf{S} := \mathsf{S}, x \approx y}$$

$$\textsf{Nth-Split}\ \frac{\mathsf{nth}(x,i), \mathsf{nth}(x',i') \in \mathcal{T}(\mathsf{S}) \qquad i \approx i' \in \mathsf{A}}{\mathsf{S} := \mathsf{S}, x \approx x' \quad \| \quad \mathsf{S} := \mathsf{S}, x \not\approx x'}$$

Fig. 6: Extended derivation rules. The rules use $z_1, \ldots, z_n$ to denote fresh sequence variables and e, e' to denote fresh element variables.

The theorem states that the calculi are correct in the following sense: if a closed derivation tree is obtained for the constraint set $\mathsf{S}_0 \cup \mathsf{A}_0$ then that is unsatisfiable in T_{Seq}; if a tree with a saturated leaf is obtained, then it is satisfiable. It is possible, however, that neither kind of tree can be derived by the calculi, making them neither refutation-complete nor terminating. This is not surprising since, as mentioned in the introduction, the decidability of even weaker theories is still unknown.

In the remainder of this section, we prove the above theorem for the extended calculus EXT. The simpler case of BASE can be obtained by an adaptation.

5.1 Proof of Theorem 1, Item 1

The proof of Item 1 is routine, and amounts to a local soundness check of each derivation rule. Most rules in our calculi are easily shown to be sound. The only non-routine cases are those involving the $\models^*_{++}$ relation. They rely on the following lemma.

Lemma 7 *Let* S *be a set of sequence constraints and* A *a set of arithmetic constraints. Suppose* (S, A) *is saturated w.r.t.* S-Prop, L-Intro, *and* L-Valid. *If* $\mathsf{S} \models^*_{++} x \approx s$ *then* $\mathsf{S} \models_{T_{\mathsf{Seq}}} x \approx s$.

Proof Suppose $\mathsf{S} \models^*_{++} x \approx \overline{\boldsymbol{s}}$ where $\overline{\boldsymbol{s}} = (s_1, \ldots, s_n)$. Then, for some $\overline{\boldsymbol{s}}' = (s'_1, \ldots, s'_n)$, $\mathsf{S} \models_{++} x \approx \overline{\boldsymbol{s}}'$ and $\mathsf{S} \models s_i \approx s'_i$ for $i \in [1, n]$. To show that $\mathsf{S} \models_{T_{\mathsf{Seq}}} x \approx \overline{\boldsymbol{s}}$, it thus suffices to show that $\mathsf{S} \models_{T_{\mathsf{Seq}}} x \approx \overline{\boldsymbol{s}}'$. As shown in Lemma 5, there is a finite number k of derivation steps in $\models_{++}$ that yield $x \approx \overline{\boldsymbol{s}}'$. We prove the claim by induction on k. For $k = 1$, we either apply Item 1 or Item 2. In the former case we get a trivial identity $\mathsf{S} \models_{++} x \approx x$, and clearly $\mathsf{S} \models_{T_{\mathsf{Seq}}} x \approx x$. In the latter case, we get an identity $\mathsf{S} \models_{++} x \approx t$ such that $\mathsf{S} \models x \approx t$, and so in particular $\mathsf{S} \models_{T_{\mathsf{Seq}}} x \approx t$. Suppose $k > 1$, and that $x \approx \overline{\boldsymbol{s}}'$ was obtained using Item 3 of Definition 6. Hence, $\overline{\boldsymbol{s}}'$ has the form $(\overline{\boldsymbol{w}} ++ t ++ \overline{\boldsymbol{z}})\downarrow$, where $\mathsf{S} \models_{++} x \approx (\overline{\boldsymbol{w}} ++ y ++ \overline{\boldsymbol{z}})\downarrow$ with a shorter derivation, and $\mathsf{S} \models y \approx t$. By the induction hypothesis, $\mathsf{S} \models_{T_{\mathsf{Seq}}} x \approx (\overline{\boldsymbol{w}} ++ y ++ \overline{\boldsymbol{z}})\downarrow$. Clearly, $\mathsf{S} \models_{T_{\mathsf{Seq}}} y \approx t$. Hence, $\mathsf{S} \models_{T_{\mathsf{Seq}}} x \approx \overline{\boldsymbol{s}}'$. $\square$

Using Lemma 7, we can prove the soundness of rules that use $\models^*_{++}$. For example, we can show the soundness of the C-Split rule as follows. Since $\mathsf{S} \models^*_{++} x \approx s$ implies that $x \approx s$ follows from S in T_{Seq}, we have that every model of the premise configuration of C-Split assigns the same value to $(\overline{\boldsymbol{w}} ++ y ++ \overline{\boldsymbol{z}})\downarrow$ and $(\overline{\boldsymbol{w}} ++ y' ++ \overline{\boldsymbol{z}}')\downarrow$. Assuming that the first two branches of the conclusion do not hold in some model, we indeed get that the interpretations of y and y' must be identical, as they are the sub-sequence of the same sequence, that begins at the same index and whose length is the same.

5.2 Proof of Theorem 1, Item 2

Consider a saturated configuration (S, A). We first define an interpretation $\mathcal{M}$, based on (S, A) in Section 5.2.1. Then, we prove that $\mathcal{M}$ is well-defined in Section 5.2.2. Finally, we show in Section 5.2.3 that $\mathcal{M} \models \mathsf{S} \cup \mathsf{A}$. By construction, $\mathcal{M}$ is designed to be a model of T_{Seq}, hence $\mathsf{S} \cup \mathsf{A}$ is T_{Seq}-satisfiable. The T_{Seq}-satisfiability of $\mathsf{S}_0 \cup \mathsf{A}_0$ is an immediate consequence of that fact that $\mathsf{S}_0 \cup \mathsf{A}_0 \subseteq \mathsf{S} \cup \mathsf{A}$ since, except for A-Conf and S-Conf, none of rules removes constraints from the current configuration.

5.2.1 Model Construction

We will construct a satisfying interpretation $\mathcal{M}$ for a saturated configuration (S, A) bottom-up by interpreting Elem as an arbitrary but *countably infinite* set. We can do that because, intuitively, T_{Seq} itself allows such interpretations and is such that no set of Σ_{Seq} constraints can impose an upper bound on the cardinality of Elem. Note, however, that in a theory combination setting, where Elem is replaced by a sort (such as Bool, for instance) that admits only finite interpretations, our construction will not work. In that case, in fact, our calculus needs to be extended with additional rules that take the finiteness of the element sort into account as done, for instance, in Liang et al. [21] for the theory of strings over a finite alphabet.

For our model construction, we start with the lengths of the sequences, obtained from solving the T_{LIA}-constraints. Then the element variables are be assigned with distinct interpretations. For that, we rely on the element domain being infinite. We use a *weak equivalence* graph to capture constraints that are based on update. In the graph, two variables connected by an edge will differ in their interpretation on at most one element. We carefully assign atomic sequence variables in the graph, starting with element-wise assignments deduced by nth. We assign distinct values to all remaining elements for the sequence variables. Non-atomic sequence variables are assigned simply by following the values assigned to their corresponding equivalent concatenation terms.

Unless stated otherwise, by *equivalence class* we mean an equivalence class w.r.t. $\equiv_{\mathsf{S}}$ and may drop the subscript when referring to it (e.g., writing $[x]$ for the equivalence class of x). We say that an equivalence class e is *atomic in* S if its terms are of sort Seq and all variables in it are atomic in S. From now on, we will simply say "atomic" to mean "atomic in S." Note that if e contains any variable that is atomic, then all variables in e are atomic.

The definitions provided below completely define the interpretation $\mathcal{M}$. Definition 10 defines the domains of $\mathcal{M}$; Definition 11 defines the interpretations of the function symbols of Σ_{Seq}, except nth, whose definition is deferred. Definition 12 defines the interpretations of variables of sort Int and Elem; Definition 13 assigns lengths, but not yet values, to the interpretations of variables of sort Seq that are atomic; Definition 14 defines the interpretations of variables of sort Seq that are equivalent to unit terms; Definition 17 defines the interpretations of variables of sort Seq that are atomic but not equivalent to unit terms; Definition 18 defines the interpretations of variables of sort Seq that are not atomic; Finally, Definition 19 fully defines the interpretation of the nth symbol in $\mathcal{M}$.[6]

Definition 10 (Model construction: domains)
1. *$\mathcal{M}(\mathsf{Int}) = \mathbb{Z}$, the set of integers.*
2. *$\mathcal{M}(\mathsf{Elem})$ is some countably infinite set E.*
3. *$\mathcal{M}(\mathsf{Seq})$ is the set E^* (of finite sequences whose elements are taken from E).*

[6]Recall that the meaning of nth is fixed by the theory only for certain inputs.

The domains for Int and Seq were chosen to meet the requirements of T_{Seq}. The exact identity of Elem is not important, and so we set its elements to be arbitrary. In contrast, as argued above, its cardinality is, as we will see later.

Definition 11 (Model construction: function symbols) *The symbols shown in Figure 1, except for* nth*, are interpreted as prescribed by the semantics of T_{Seq} provided in Section 3.1.*

Now we can assign values to variables of sort Int and Elem.

Definition 12 (Model construction: Int and Elem variables)

1. *Let $\mathcal{A}$ be a T_{LIA}-interpretation (with $\mathcal{A}(\mathsf{Int}) = \mathbb{Z}$) that satisfies* A*. Then $\mathcal{M}(x) := \mathcal{A}(x)$ for every variable x of sort* Int*.*
2. *Let $a_1, a_2, \ldots$ be an enumeration of $\mathcal{M}(\mathsf{Elem})$, and let $e_1, \ldots, e_n$ be an enumeration of the equivalence classes of $\equiv_{\mathsf{S}}$ whose variables have sort* Elem*. Then, for every $i \in [1, n]$ and every variable $x \in e_i$, $\mathcal{M}(x) := a_i$.*

Next, we assign values to variables of sort Seq. We will first associate a sequence value with every sequence equivalence class (we write $\mathcal{M}(e)$ for the value associated with equivalence class e). Then, for each equivalence class e and for each $x \in e$, we define $\mathcal{M}(x) := \mathcal{M}(e)$. We start by defining lengths.

Definition 13 (Model construction: length of atomic variables) *For each variable x in an atomic equivalence class e of $\equiv_{\mathsf{S}}$, we constrain $\mathcal{M}(e)$ to be a sequence of length $\mathcal{M}(\ell_x)$.*

Note that because every atomic variable is in an atomic equivalence class, Definition 13 assigns a length to every atomic variable. In what follows, we denote the length assigned to $\mathcal{M}(e)$ in Definition 13 by ℓ_e.

Next, we define model values for atomic variables. We start with equivalence classes that contain unit-terms. We will call an equivalence class e of $\equiv_{\mathsf{S}}$ a *unit equivalence class* if $\mathsf{unit}(x) \in e$ for some x.

Definition 14 (Model construction: atomic variables in unit equivalence classes) *For every atomic variable x such that $x \equiv_{\mathsf{S}} \mathsf{unit}(y)$ for some y, we set $\mathcal{M}([x])$ to be a sequence of length 1 whose only element is $\mathcal{M}(y)$.*

Next, we turn to atomic equivalence classes that are not unit. We begin by defining a graph in order to keep track of constraints that originate from the update symbol. The graph connects atomic equivalence classes by edges constructed from update.

If two vertices are connected by an edge, one of them is an update of the other, and so they differ on at most 1 element. This definition is an adaptation of the weak equivalence graph from Christ and Hoenicke [12] to the context of sequences, where operations are richer. In particular, we take into account not only whether a constraint of the form $y \approx \mathsf{update}(x, i, a)$ appears in S, but also whether x and y are atomic, and whether the interpretation already given to i in Definition 12 is within the range determined by the length assigned to x and y in Definition 13.

Definition 15 (Weak equivalence graph) *Define a graph $G = (V, E, \delta)$ as follows. V is the set of atomic equivalence classes. $E \subseteq V \times V$ is a set of unordered edges, and $\delta : E \to \mathsf{P}(\mathbb{N})$ is a labeling function on edges, such that $(e_1, e_2) \in E$ and $k \in \delta((e_1, e_2))$ iff there are $x \in e_1$ and $y \in e_2$ such that $y \approx \mathsf{update}(x, i, z) \in \mathsf{S}$ or $x \approx \mathsf{update}(y, i, z) \in \mathsf{S}$, where $\mathcal{M}(i) = k$ and $0 \leq k < \mathcal{M}(\ell_{e_1}) = \mathcal{M}(\ell_{e_2})$.*

Definition 16 (Weak equivalence) *Given a weak equivalence graph $G = (V, E, \delta)$, for each $i \in \mathbb{N}$, we define a binary relation $\sim_i$ over V as follows: $e_1 \sim_i e_2$ iff there exists a path p between e_1 and e_2 in G, such that for each edge d in p, $\delta(d) \setminus \{i\} \neq \emptyset$.*

It is routine to verify the following lemma.

Lemma 8 *For each $i \in \mathbb{N}$, $\sim_i$ is an equivalence relation over the atomic equivalence classes of $\equiv_{\mathsf{S}}$.*

Notice that even though $\sim_i$ is defined using update-terms, every atomic equivalence class e satisfies $e \sim_i e$, even if it has no update-terms.

Let ℓ be the maximal sequence length assigned in Definition 13. Let $a_1^1, a_1^2, \ldots, a_1^\ell, a_2^1, \ldots, a_2^\ell, \ldots$ be an enumeration of the elements in $\mathcal{M}(\mathsf{Elem})$ that were not assigned to any variable of Elem sort. For each $i \in [0, \ell)$, let $E_1^i, E_2^i, \ldots, E_{n_i}^i$ be an enumeration of the equivalence classes of $\sim_i$. The following lemma is a consequence of Lemma 8.

Lemma 9 *For every atomic equivalence class e of $\equiv_{\mathsf{S}}$, and for each $i \in [0, \ell_e)$, there exists some j such that $e \in E_j^i$.*

Definition 17 (Model construction: atomic variables in non-unit equivalence classes) *Let e be an atomic equivalence class of $\equiv_{\mathsf{S}}$ that is not a unit equivalence class. We set the ith element of $\mathcal{M}(e)$ for every $i \in [0, \ell_e)$ as follows. By Lemma 9, there exists some j such that $e \in E_j^i$.*

1. *If there are $e' \in E_j^i$, $s \in e'$, x, and y such that $x \equiv_{\mathsf{S}} \mathsf{nth}(s, y)$ and $\mathcal{M}(y) = i$, we set the ith element of $\mathcal{M}(e)$ to be $\mathcal{M}(x)$.*
2. *Otherwise, the ith element of $\mathcal{M}(e)$ is set to a_j^i.*

Next, we set the interpretation of non-atomic variables of sort Seq.

Definition 18 (Model construction: non-atomic sequence variables) *Let x be a non-atomic variable of sort* Seq*, and let $\overline{\boldsymbol{y}}$ be its normal form. $\mathcal{M}([x])$ is set to be the concatenation of the interpretations of the variables in $\overline{\boldsymbol{y}}$. ($\mathcal{M}([x])$ is the empty sequence if $\overline{\boldsymbol{y}}$ is of size 0.)*

Finally, we define the interpretation of nth. $\mathcal{M}(\mathsf{nth})$ is a function from $\mathcal{M}(\mathsf{Seq}) \times \mathcal{M}(\mathsf{Int})$ to $\mathcal{M}(\mathsf{Elem})$. Given $a \in \mathcal{M}(\mathsf{Seq})$ and an integer i, if i is non-negative and smaller than the length of a, the value of $\mathcal{M}(\mathsf{nth})(a, i)$ is fixed by the theory. The following definition assigns a value to $\mathcal{M}(\mathsf{nth})(a, i)$ for other i's.

Definition 19 (Model construction: nth terms) *For every element $a \in \mathcal{M}(\mathsf{Seq})$ with length n and for every $i \in \mathbb{N}$ such that $i < 0$ or $i \geq n$, if there are s, x, and y such that $x \equiv_{\mathsf{S}} \mathsf{nth}(s, y)$, $\mathcal{M}(s) = a$, and $\mathcal{M}(y) = i$, $\mathcal{M}(\mathsf{nth})(a, i)$ is set to $\mathcal{M}(x)$. Otherwise, it is set arbitrarily.*

This concludes the construction of $\mathcal{M}$. The following is an example of a construction of a model according to the above definitions.

Example 5 *Consider a signature in which* Elem *is* Int*, and a saturated configuration $(\mathsf{S}^*, \mathsf{A}^*)$ w.r.t.* EXT *that includes the following formulas: $y \approx y_1 +\!\!+\, y_2$, $x \approx x_1 +\!\!+\, x_2$, $y_2 \approx x_2$, $y_1 \approx \mathsf{update}(x_1, i, a)$, $|y_1| = |x_1|$, $|y_2| = |x_2|$, $\mathsf{nth}(y, i) \approx a$, $\mathsf{nth}(y_1, i) \approx a$. Following the above construction, a satisfying interpretation $\mathcal{M}$ can be built as follows:*

Definition 10 Set both $\mathcal{M}(\mathsf{Int})$ and $\mathcal{M}(\mathsf{Elem})$ to be the set of integer numbers. $\mathcal{M}(\mathsf{Seq})$ is fixed by the theory.*

Definition 12 First, find an arithmetic model, $\mathcal{M}(\ell_x) = \mathcal{M}(\ell_y) = 4, \mathcal{M}(\ell_{y_1}) = \mathcal{M}(\ell_{x_1}) = 2, \mathcal{M}(\ell_{y_2}) = \mathcal{M}(\ell_{x_2}) = 2, \mathcal{M}(i) = 0$. Further, set $\mathcal{M}(a) = 0$.*

Definition 13 Start assigning values to sequences. First, set the lengths of $\mathcal{M}(x)$ and $\mathcal{M}(y)$ to be 4, and the lengths of $\mathcal{M}(x_1), \mathcal{M}(x_2), \mathcal{M}(y_1), \mathcal{M}(y_2)$ to be 2.*

Definition 14 Skipped as there are no* unit *terms.*

Definition 17 Next, according to Item 1, the 0th element of $\mathcal{M}(y_1)$ is set to 0 (y_1 is atomic, y is not.). According to Item 2, assign fresh values to the remaining indices of atomic variables. The result can be, e.g., $\mathcal{M}(y_1) = [0, 2], \mathcal{M}(x_1) = [1, 2], \mathcal{M}(y_2) = \mathcal{M}(x_2) = [3, 4]$.*

Definition 18 Assign non-atomic sequence variables based on equivalent concatenations: $\mathcal{M}(y) = [0, 2, 3, 4], \mathcal{M}(x) = [1, 2, 3, 4]$.*

Definition 19 No integer variable in the formula was assigned an out-of-bound value, and so the interpretation of* nth *on out-of-bounds cases is set arbitrarily.*

5.2.2 Well-definedness

Note that Definitions 10 and 11 are trivially well-defined. We now go through the remaining definitions.

Lemma 10 *Definition 12 is well-defined.*

Proof To show that Item 1 is well-defined, we note that by saturation of A-Conf, A is T_{LIA}-satisfiable. To show that Item 2 is well-defined, we establish an infinite enumeration $a_1, a_2, \dots$ of $\mathcal{M}(\mathsf{Elem})$, which exists due to Definition 10. $\square$

Lemma 11 *Definition 13 is well-defined.*

Proof Let $x, y \in e$. We show that $\mathcal{M}(\ell_x) = \mathcal{M}(\ell_y)$. Since $x \equiv_{\mathsf{S}} y$, we have $\mathsf{S} \models |x| \approx |y|$. By Assumption 1, we also have $\mathsf{S} \models \ell_x \approx \ell_y$. By saturation of S-Prop, we have $\ell_x \approx \ell_y \in \mathsf{A}$, and hence, by Definition 12, $\mathcal{M}(\ell_x) = \mathcal{M}(\ell_y)$. $\square$

Lemma 12 *Definition 14 is well-defined.*

Proof We first show that $\ell_{[x]}$ is 1. To see this, note first that by saturation of L-Intro (and Figure 3), we have $\mathsf{S} \models |\mathsf{unit}(y)| \approx 1$. We also have $\ell_x \approx |x| \in \mathsf{S}$ by Assumption 1. It follows that $\mathsf{S} \models \ell_x \approx 1$, so $\ell_x \approx 1 \in \mathsf{A}$ by saturation of S-Prop. Thus, we must have $\mathcal{A}(\ell_x) = 1$ in Definition 12, and thus $\mathcal{M}(\ell_x) = 1$. By Definition 13, we then have that the length of $\mathcal{M}([x])$ must be 1.

Next, suppose $\mathsf{unit}(y), \mathsf{unit}(z) \in [x]$. We prove $\mathcal{M}(y) = \mathcal{M}(z)$. Since $\mathsf{S} \models \mathsf{unit}(y) \approx \mathsf{unit}(z)$, we must have $y \approx z \in \mathsf{S}$, by saturation of U-Eq. Hence, $y \equiv_{\mathsf{S}} z$ and so by Definition 12, we have $\mathcal{M}(y) = \mathcal{M}(z)$. $\square$

For the well-definedness of Definition 17, we use the following helper lemma which shows that the model assigns equal lengths to updated sequences.

Lemma 13 *If $y \approx \mathsf{update}(x, i, z) \in \mathsf{S}$, then $\mathcal{M}(\ell_x) = \mathcal{M}(\ell_y)$.*

Proof By congruence, $\mathsf{S} \models |y| \approx |\mathsf{update}(x, i, z)|$, and by saturation of L-Intro and the definition of $\downarrow$, we know that $\mathsf{S} \models |\mathsf{update}(x, i, z)| \approx |x|$. Thus, $\mathsf{S} \models |x| \approx |y|$. By Assumption 1, $\mathsf{S} \models \ell_x \approx \ell_y$, so by saturation of S-Prop, we have $\mathsf{A} \models \ell_x \approx \ell_y$. It follows from Definition 12 that $\mathcal{A}(\ell_x) = \mathcal{A}(\ell_y)$, and thus, $\mathcal{M}(\ell_x) \approx \mathcal{M}(\ell_y)$. $\square$

Lemma 14 *Definition 17 is well-defined.*

Proof Let e be as in Definition 17. Suppose there are $e', e'' \in E^i_j$, with $s' \in e'$, $s'' \in e''$, and x', y', x'', y'' such that $x' \equiv_{\mathsf{S}} \mathsf{nth}(s', y')$ and $x'' \equiv_{\mathsf{S}} \mathsf{nth}(s'', y'')$ and $\mathcal{M}(y') = \mathcal{M}(y'') = i$, with $i \in [0, \ell_e)$. We prove that $\mathcal{M}(x') = \mathcal{M}(x'')$.

We first show that $y' \approx y'' \in \mathsf{S}$. Clearly $y', y'' \in \mathcal{T}(\mathsf{S})$. Also, $y', y'' \in \mathcal{T}(\mathsf{A})$ since $y' \approx y' \in \mathsf{A}$ and $y'' \approx y'' \in \mathsf{A}$ by saturation of S-Prop. It follows by saturation of S-A that either $y' \approx y'' \in \mathsf{A}$ or $y' \not\approx y'' \in \mathsf{A}$. Since $\mathcal{M}(y') = \mathcal{M}(y'')$, the latter cannot hold, and so the former holds. Then, by saturation of A-Prop, we have $y' \approx y'' \in \mathsf{S}$ as well.

Now, notice that $e' \sim_i e''$. Hence, there exist $e_1, \dots, e_n$ such that $e_1 = e'$, $e_n = e''$ and for $k \in [1, n)$, e_k and e_{k+1} are connected by an edge d_k in G where $\delta(d_k) \setminus \{i\} \neq \emptyset$. For every such k, we have that $s_k \approx \mathsf{update}(s'_k, y_k, z_k) \in \mathsf{S}$ or $s'_k \approx \mathsf{update}(s_k, y_k, z_k) \in \mathsf{S}$ for some $s_k \in e_k$, $s'_k \in e_{k+1}$, integer variable y_k, and Elem-variable z_k. By Lemma 13, $\mathcal{M}(\ell_{s_k}) = \mathcal{M}(\ell_{s'_k})$. And by Definition 13, $\ell_{e_k} = \mathcal{M}(\ell_{s_k})$ and $\ell_{e_{k+1}} = \mathcal{M}(\ell_{s'_k})$. It follows that $\ell_{e'} = \ell_{e_1} = \ldots = \ell_{e_n} = \ell_{e''}$. By a similar argument, because $e \sim_i e'$, we have $\ell_{e'} = \ell_e$. We also have: $(*)$ $\mathcal{M}(y_k) \neq i$ and $(**)$ $\mathcal{M}(y_k) \in [0, \ell_e)$. Define s'_0 to be an alias for s' and then notice that for $k \in [1, n)$, $s_k \equiv_{\mathsf{S}} s'_{k-1}$ because s_k, s'_{k-1} are both in e_k.

We prove by induction that for $k \in [0, n)$, $\mathsf{nth}(s', y') \equiv_{\mathsf{S}} \mathsf{nth}(s'_k, y')$. For the base case, we simply note that $\mathsf{nth}(s', y')$ and $\mathsf{nth}(s'_0, y')$ are identical and $\mathsf{nth}(s', y') \in \mathcal{T}(\mathsf{S})$. For the induction step, suppose that $\mathsf{nth}(s', y') \equiv_{\mathsf{S}} \mathsf{nth}(s'_k, y')$, where $k \in [0, n-1)$. This implies $\mathsf{nth}(s'_k, y') \in \mathcal{T}(\mathsf{S})$, and we also know $s'_k \equiv_{\mathsf{S}} s_{k+1}$. Recalling that $s_{k+1} \approx \mathsf{update}(s'_{k+1}, y_{k+1}, z_{k+1}) \in \mathsf{S}$ or $s'_{k+1} \approx \mathsf{update}(s_{k+1}, y_{k+1}, z_{k+1}) \in \mathsf{S}$, we see that the premises of Nth-Update are satisfied. By saturation of Nth-Update, then, there are three possibilities.

1. In the first case, $\mathsf{A} \models y' < 0 \vee y' \geq \ell_{s'_k}$. We know from Definition 12 that $\mathcal{M}(y') = \mathcal{A}(y')$ and $\mathcal{M}(\ell_{s'_k}) = \mathcal{A}(\ell_{s'_k})$, so we must have $\mathcal{M}(y') < 0$ or $\mathcal{M}(y') \geq \mathcal{M}(\ell_{s'_k})$. But $\mathcal{M}(y') = i$ and $i \in [0, \ell_e)$, and we know that $\mathcal{M}(\ell_{s'_k}) = \ell_{e_{k+1}} = \ell_e$, so this case is not possible.
2. In the second case, $\mathsf{A} \models y' \approx y_{k+1}$. This is also not possible because we know that $\mathcal{A}(y') = \mathcal{M}(y') = i \neq \mathcal{M}(y_{k+1}) = \mathcal{A}(y_{k+1})$.
3. In the third case, $\mathsf{nth}(s_{k+1}, y') \equiv_{\mathsf{S}} \mathsf{nth}(s'_{k+1}, y')$. But we know that $s'_k \equiv_{\mathsf{S}} s_{k+1}$, so we also have $\mathsf{nth}(s'_k, y') \equiv_{\mathsf{S}} \mathsf{nth}(s'_{k+1}, y')$. Then, by the induction hypothesis, $\mathsf{nth}(s', y') \equiv_{\mathsf{S}} \mathsf{nth}(s'_{k+1}, y')$, which completes the induction proof.

Letting $k = n-1$, we obtain $\mathsf{nth}(s', y') \equiv_{\mathsf{S}} \mathsf{nth}(s'_{n-1}, y')$. But $s'_{n-1} \in e_n$ and $e_n = e''$, so $s'_{n-1} \equiv_{\mathsf{S}} s''$ and $\mathsf{nth}(s', y') \equiv_{\mathsf{S}} \mathsf{nth}(s'', y')$. Finally, since we showed above that $y' \approx y'' \in \mathsf{S}$, we have $\mathsf{nth}(s', y') \equiv_{\mathsf{S}} \mathsf{nth}(s'', y'')$, and thus $x' \equiv_{\mathsf{S}} x''$, which means that $\mathcal{M}(x') = \mathcal{M}(x'')$ by Definition 12. □

Lemma 15 *Definition 18 is well-defined.*

Proof $\overline{\boldsymbol{y}}$ exists and is unique by Lemma 6. If $\overline{\boldsymbol{y}}$ is a variable or a variable-concatenation term, then uniqueness guarantees well-definedness. Further, each variable that occurs in $\overline{\boldsymbol{y}}$ is atomic by Lemma 6, and hence its value in $\mathcal{M}$ was already defined in Definitions 14 and 17. □

We prove that Definition 19 is well-defined, but first we prove some helper lemmas. Recall that we write $\mathsf{C} \models_{\mathsf{LIA}} \varphi$ to denote that every model of T_{LIA} satisfying C also satisfies φ. Intuitively, if φ can be derived from C using arithmetic reasoning, then $\mathsf{C} \models_{\mathsf{LIA}} \varphi$.

The first lemma states that $\models_{+\!+}$ conforms with the lengths of sequences.

Lemma 16 *If* $\mathsf{S} \models_{+\!+} x \approx \overline{\boldsymbol{z}}$ *and* $\overline{\boldsymbol{z}}$ *is of size* n, *then* $\mathsf{A} \models_{\mathsf{LIA}} \ell_x = \Sigma_{i=1}^{n} \ell_{z_i}$.

Proof The proof is by structural induction using Definition 6. For Item 1, clearly $\mathsf{A} \models \ell_x \approx \ell_x$. For Item 2, we have $\mathsf{S} \models_{+\!+} x \approx t$ for some variable concatenation term $t = t_1 +\!+ \cdots +\!+ t_n$ such that $\mathsf{S} \models x \approx t$. Therefore, $\mathsf{S} \models |x| \approx |t|$. By saturation of L-Intro, $\mathsf{S} \models |x| \approx \Sigma_{i=1}^{n} |t_i|$, and using Assumption 1, we get $\mathsf{S} \models \ell_x \approx \Sigma_{i=1}^{n} \ell_{t_i}$. By saturation of S-Prop, $\mathsf{A} \models \ell_x \approx \Sigma_{i=1}^{n} \ell_{t_i}$.

Now suppose that $\mathsf{S} \models_{+\!+} x \approx (\overline{\boldsymbol{w}} +\!+ y +\!+ \overline{\boldsymbol{z}})\!\downarrow$ and $\mathsf{S} \models y \approx t$, where t is ϵ or a variable concatenation term in $\mathcal{T}(S)$. Let $\overline{\boldsymbol{w}} = (w_1, \ldots, w_m)$ and $\overline{\boldsymbol{z}} = (z_1, \ldots, z_n)$, with $m, n \geq 0$. By the induction hypothesis, we have that $\mathsf{A} \models_{\mathsf{LIA}} \ell_x \approx \Sigma_{i=1}^{m} \ell_{w_i} + \ell_y +$

$\Sigma_{i=1}^{n}\ell_{z_i}$. We consider two cases. (1) $t = \epsilon$: in this case, $\mathsf{S} \models |y| \approx |\epsilon|$. Also, $\mathsf{S} \models |\epsilon| = 0$ by saturation of L-Intro, so $\mathsf{S} \models |y| = 0$, and thus $\mathsf{A} \models \ell_y = 0$ by saturation of S-Prop and Assumption 1. It follows that $\mathsf{A} \models_{\mathsf{LIA}} \ell_x \approx \Sigma_{i=1}^{m}\ell_{w_i} + \Sigma_{i=1}^{n}\ell_{z_i}$. (2) t is a variable concatenation term in $\mathcal{T}(\mathsf{S})$: let $t = t_1 +\!\!+ \ldots +\!\!+ t_k$. We have $\mathsf{S} \models |y| \approx |t|$. Also, $\mathsf{S} \models |t| = \Sigma_{i=1}^{k}|t_i|$ by saturation of L-Intro. By saturation of S-Prop and Assumption 1 (and assuming wlog that $\Sigma_{i=1}^{k}\ell_{t_i}$ is the result of flattening $\Sigma_{i=1}^{k}|t_i|$), it follows that $\mathsf{A} \models \ell_y = \Sigma_{i=1}^{k}\ell_{t_i}$. Thus, $\mathsf{A} \models_{\mathsf{LIA}} \ell_x \approx \Sigma_{i=1}^{m}\ell_{w_i} + \Sigma_{i=1}^{k}\ell_{t_i} + \Sigma_{i=1}^{n}\ell_{z_i}$. □

Next, we show that the model respects the lengths of sequence variables.

Lemma 17 *For every sequence variable $x \in \mathsf{S}$, if ℓ is the length of $\mathcal{M}(x)$, then $\mathcal{M}(\ell_x) = \ell$.*

Proof If x is atomic, then $\ell = \mathcal{M}(\ell_x)$ by Definition 13. Suppose that x is non-atomic. Let $\overline{\boldsymbol{y}}$ be the normal form of x where $\overline{\boldsymbol{y}}$ is of size n. Each element of $\overline{\boldsymbol{y}}$ is atomic, so for $i \in [1, n]$, the length of $\mathcal{M}(y_i)$ is $\mathcal{M}(\ell_{y_i})$ by Definition 13. Then, $\ell = \Sigma_{i=1}^{n}\mathcal{M}(\ell_{y_i})$ by Definition 18. Let $\overline{\boldsymbol{z}}$ of length n be such that $\mathsf{S} \models_{+\!\!+} x \approx \overline{\boldsymbol{z}}$ and $\mathsf{S} \models y_i \approx z_i$ for $i \in [1, n]$, which exists by Definition 6. By Lemma 16, we have that $\mathsf{A} \models_{\mathsf{LIA}} \ell_x \approx \Sigma_{i=1}^{n}\ell_{z_i}$. For each $i \in [1, n]$, we know that because $\mathsf{S} \models y_i \approx z_i$, $\mathsf{S} \models |y_i| \approx |z_i|$, and so by Assumption 1, $\mathsf{S} \models \ell_{y_i} \approx \ell_{z_i}$. Therefore, $\mathsf{A} \models \ell_{y_i} \approx \ell_{z_i}$ by saturation of S-Prop. Then, we can conclude that $\mathsf{A} \models_{\mathsf{LIA}} \ell_x \approx \Sigma_{i=1}^{n}\ell_{y_i}$. Finally, we have $\mathcal{M}(\ell_x) = \mathcal{A}(\ell_x) = \Sigma_{i=1}^{n}\mathcal{A}(\ell_{y_i}) = \Sigma_{i=1}^{n}\mathcal{M}(\ell_{y_i}) = \ell$. □

The next lemmas show that nth terms are evaluated correctly, first for atomic classes and then generally.

Lemma 18 *Suppose $k \equiv_{\mathsf{S}} \mathsf{nth}(x, y)$. Let $i = \mathcal{M}(y)$, $e = [x]$, and let ℓ_e be the length of $\mathcal{M}(x)$. If e is atomic and $i \in [0, \ell_e)$, then the ith element of $\mathcal{M}(x)$ is $\mathcal{M}(k)$.*

Proof We consider the following cases:

1. e is a unit equivalence class: then, for some z, $\mathsf{unit}(z) \in e$. By Definition 14, we have that $i = 0$ and $\mathcal{M}(x)$ contains the single element $\mathcal{M}(z)$. We show that $\mathcal{M}(z) = \mathcal{M}(k)$. Since $\mathsf{nth}(x, y) \in \mathcal{T}(\mathsf{S})$ and $\mathsf{S} \models x \approx \mathsf{unit}(z)$, by saturation of Nth-Unit, there are two cases. In the first case, $\mathsf{A} \models y < 0 \vee y > 0$, which, by Definition 12, is not possible since $\mathcal{M}(y) = i = 0$. In the second case, $\mathsf{nth}(x, y) \approx z \in \mathsf{S}$. It follows that $z \equiv_{\mathsf{S}} k$, so by Definition 12, $\mathcal{M}(z) = \mathcal{M}(k)$.
2. e is atomic but not a unit equivalence class: let j be such that $e \in E_j^i$. Then, by Item 1 of Definition 17, the ith element of $\mathcal{M}(e)$ must be $\mathcal{M}(k)$, and thus the ith element of $\mathcal{M}(x)$ is $\mathcal{M}(k)$. □

Lemma 19 *Suppose $k \equiv_{\mathsf{S}} \mathsf{nth}(x, y)$. Let $i = \mathcal{M}(y)$, $e = [x]$, and let ℓ_e be the length of $\mathcal{M}(x)$. If $i \in [0, \ell_e)$, then the ith element of $\mathcal{M}(x)$ is $\mathcal{M}(k)$.*

Proof If e is atomic, then we have the result by Lemma 18. Suppose e is not atomic. Let $\overline{\boldsymbol{x}}$ be the normal form of x.

1. If $\overline{\boldsymbol{x}}$ is empty, then by Definition 18, $\mathcal{M}(x)$ is the empty sequence, so $i \in [0, \ell_e)$ is always false, and the statement holds vacuously.

2. Suppose $\overline{\boldsymbol{x}}$ has a single element, x_1. By Definition 6, it is clear that $\mathsf{S} \models^*_{++} x_1 \approx x_1$. So, by saturation of $\mathsf{C\text{-}Eq}$, we must have $x \approx x_1 \in \mathsf{S}$. But x_1 is atomic, so this contradicts the assumption that e is not atomic.
3. Otherwise, $\overline{\boldsymbol{x}} = x_1 ++ \cdots ++ x_n$, with $n \geq 2$. Recall that $\mathsf{nth}(x, y) \in \mathcal{T}(\mathsf{S})$. Thus, by saturation of $\mathsf{Nth\text{-}Concat}$, one of its $n+1$ conclusions is applicable. In the first case, we must have $\mathsf{A} \models y < 0 \vee y \geq \ell_x$. But we also know that $\mathcal{M}(y) = i$ is non-negative and is smaller than the length assigned to $M(x)$, which leads to a contradiction using Lemma 17. For the other cases, we have, for some $k \in [1, n]$, (1) $\mathsf{A} \models \Sigma_{j=1}^{k-1} \ell_{x_j} \leq y < \Sigma_{j=1}^{k} \ell_{x_j}$ and (2) $\mathsf{S} \models \mathsf{nth}(x, y) \approx \mathsf{nth}(x_k, y - \Sigma_{j=1}^{k-1} \ell_{x_j})$. By Definition 12, this means that $\Sigma_{j=1}^{k-1} \mathcal{M}(\ell_{x_j}) \leq \mathcal{M}(y) < \Sigma_{j=1}^{k} \mathcal{M}(\ell_{x_j})$. Now, by Definition 18, $\mathcal{M}(x) = \mathcal{M}(x_1) ++ \cdots ++ \mathcal{M}(x_n)$, and by Lemma 17, the length of $\mathcal{M}(x_j)$ is $\mathcal{M}(\ell_{x_j})$ for $j \in [1, n]$. Let $i' = \mathcal{M}(y) - \Sigma_{j=1}^{k-1} \mathcal{M}(\ell_{x_j})$. Clearly, $i' \in [0, \mathcal{M}(\ell_{x_k}))$, and element $\mathcal{M}(y)$ of $\mathcal{M}(x)$ is the same as element i' of $\mathcal{M}(x_k)$. Now, revisiting (2), let α be the term $y - \Sigma_{j=1}^{k-1} \ell_{x_j}$, and let $\hat{\alpha}$ be the variable introduced for α when flattening the term $\mathsf{nth}(x_k, \alpha)$. We have $k \equiv_{\mathsf{S}} \mathsf{nth}(x, y)$, so $k \equiv_{\mathsf{S}} \mathsf{nth}(x_k, \hat{\alpha})$ by (2). Let i'' be $\mathcal{M}(\hat{\alpha})$, and recall that x_k is atomic and that the length of $\mathcal{M}(x_k)$ is $\mathcal{M}(\ell_{x_k})$. By Lemma 18, we have that if $i'' \in [0, \mathcal{M}(\ell_{x_k}))$, then the i''th element of $\mathcal{M}(x_k)$ is $\mathcal{M}(k)$. It remains to show that $i' = i''$. To see this, note that $\mathsf{S} \models \hat{\alpha} \approx \alpha$. So, by saturation of $\mathsf{S\text{-}Prop}$, we have $\mathsf{A} \models \hat{\alpha} \approx \alpha$. Then, by Definition 12, $i'' = \mathcal{M}(\hat{\alpha}) = \mathcal{M}(y) - \Sigma_{j=1}^{k-1} \mathcal{M}(\ell_{x_j}) = i'$. □

The following lemma proves that disequalities are respected.

Lemma 20 *For all x, y, with $x \not\approx y \in \mathsf{S}$, we have $\mathcal{M}(x) \neq \mathcal{M}(y)$.*

Proof By $\mathsf{Deq\text{-}Ext}$ we have two cases. In the first, $\mathsf{A} \models \ell_x \neq \ell_y$. So, by Definition 12, $\mathcal{M}(\ell_x) \neq \mathcal{M}(\ell_y)$. Thus, by Lemma 17, we have that the length of $\mathcal{M}(x)$ is different from the length of $\mathcal{M}(y)$, so $\mathcal{M}(x) \neq \mathcal{M}(y)$.

In the second case, we have (1) $\mathsf{A} \models \ell_x \approx \ell_y \wedge 0 \leq i < \ell_x$ and (2) $w_1 \approx \mathsf{nth}(x, i), w_2 \approx \mathsf{nth}(y, i), w_1 \not\approx w_2 \in \mathsf{S}$, for some i, w_1, w_2. By (2) and saturation of $\mathsf{S\text{-}Conf}$, we know that $w_1 \not\equiv_{\mathsf{S}} w_2$, so by Definition 12, $\mathcal{M}(w_1) \neq \mathcal{M}(w_2)$. Let $n = \mathcal{M}(i)$ and let ℓ be the length of $\mathcal{M}(x)$. By Definition 12, we have $\mathcal{M}(\ell_x) = \mathcal{M}(\ell_y)$ and $0 \leq n < \mathcal{M}(\ell_x)$. So, by Lemma 17, we have that the lengths of $\mathcal{M}(x)$ and $\mathcal{M}(y)$ are both equal to ℓ, and $n \in [0, \ell)$. Looking again at (2), we can apply Lemma 19 twice to get that the nth element of $\mathcal{M}(x)$ is $\mathcal{M}(w_1)$ and the nth element of $\mathcal{M}(y)$ is $\mathcal{M}(w_2)$. We can then conclude that $\mathcal{M}(x) \neq \mathcal{M}(y)$, as we know that $\mathcal{M}(w_1) \neq \mathcal{M}(w_2)$. □

We can now show that Definition 19 is well-defined.

Lemma 21 *Definition 19 is well-defined.*

Proof Suppose there are x, x', s, s', y, y' such that $x \equiv_{\mathsf{S}} \mathsf{nth}(s, y)$, $x' \equiv_{\mathsf{S}} \mathsf{nth}(s', y')$, $\mathcal{M}(s) = \mathcal{M}(s') = a$, and $\mathcal{M}(y) = \mathcal{M}(y') = i$. We prove $\mathcal{M}(x) = \mathcal{M}(x')$. Since $y, y' \in \mathcal{T}(\mathsf{S})$, and they have sort Int, by saturation of $\mathsf{S\text{-}Prop}$, we have that $y \approx y, y' \approx y' \in \mathsf{A}$, and so $y, y' \in \mathcal{T}(\mathsf{S}) \cap \mathcal{T}(\mathsf{A})$. By saturation of $\mathsf{S\text{-}A}$, either $y \not\approx y' \in \mathsf{A}$ or $y \approx y' \in \mathsf{A}$. The first case is impossible since $\mathcal{M}(y) = \mathcal{M}(y')$. In the second case, we have $y \approx y' \in \mathsf{A}$, and so $y \approx y' \in \mathsf{S}$ by saturation of $\mathsf{A\text{-}Prop}$. Now, by saturation of $\mathsf{Nth\text{-}Split}$, there are two options: either $s \not\approx s' \in \mathsf{S}$ or $s \approx s' \in \mathsf{S}$. The first is impossible by Lemma 20, as $\mathcal{M}(s) = \mathcal{M}(s')$. On the other hand, if $s \approx s' \in \mathsf{S}$, then since $x \equiv_{\mathsf{S}} \mathsf{nth}(s, y)$ and $x' \equiv_{\mathsf{S}} \mathsf{nth}(s', y')$, we also have $x \equiv_{\mathsf{S}} x'$. Thus, by Definition 12, $\mathcal{M}(x) = \mathcal{M}(x')$. □

From the well-definedness lemmas above we can then conclude that $\mathcal{M}$ is well-defined.

5.2.3 Satisfaction

In this section, we show that $\mathcal{M} \models \mathsf{S} \cup \mathsf{A}$. The arithmetic constraints (i.e., A) are satisfied by the fact that $\mathcal{M}$ extends a model $\mathcal{A}$ of T_{LIA} that satisfies A (see Definitions 11 and 12).

Lemma 22 *$\mathcal{M}$ satisfies A.*

Showing that $\mathcal{M} \models \mathsf{S}$ is more involved. Roughly speaking, we consider each possible shape of a sequence constraint separately, and prove that $\mathcal{M}$ satisfies constraints of that shape from S. The cases that include nth and update terms heavily rely on the construction of the weak equivalence graph. Constraints that include concatenation are handled by reasoning about $\models_{++}$.

The main result of this section is thus the following.

Lemma 23 *$\mathcal{M}$ satisfies S.*

The proof of this lemma is provided at the end of this section. The following lemmas will be used for the various cases of that proof. The first lemma proves the existence of atomic representatives.

Lemma 24 *A variable x is atomic in S iff $\mathsf{S} \models^*_{++} x \approx y$ for some atomic representative y.*

Proof $\Rightarrow$: Let y be the representative of $[x]$. By Definition 4, and because $x \equiv_{\mathsf{S}} y$ and x is atomic, y must also be atomic. We have $\mathsf{S} \models_{++} x \approx x$ by Definition 6, and thus, also by Definition 6, we have $\mathsf{S} \models^*_{++} x \approx y$.

$\Leftarrow$: Since y is atomic and $x \equiv_{\mathsf{S}} y$, x is also atomic by Definition 4. □

Next, we show how $\models^*_{++}$ works with concatenation terms.

Lemma 25 *If $\mathsf{S} \models^*_{++} x \approx \overline{\boldsymbol{y}}$, then $\mathcal{M}(x) = \mathcal{M}(y_1) +\!\!+ \ldots +\!\!+ \mathcal{M}(y_n)$, where $\overline{\boldsymbol{y}}$ has size $n \geq 0$.*

Proof If x is not atomic, then the result is immediate by Definition 18. If x is atomic, then by Lemma 24 and uniqueness of normal forms (Lemma 6), $n = 1$. Since $x \equiv_{\mathsf{S}} y_1$, and models are assigned by equivalence class in Definitions 14 and 17, it follows that $\mathcal{M}(x) = \mathcal{M}(y_1)$. □

In the next lemma, we show that the empty sequence obtains length 0.

Lemma 26 $\mathsf{S} \models x \approx \epsilon$ *iff* $\mathcal{M}(x)$ *has length 0.*

Proof $\Rightarrow$: If $\mathsf{S} \models x \approx \epsilon$, then $\mathsf{S} \models |x| \approx |\epsilon|$. By saturation of $\mathsf{L\text{-}Intro}$, $\mathsf{S} \models |\epsilon| = 0$. So, by Assumption 1, $\mathsf{S} \models \ell_x \approx 0$, and so, by saturation of $\mathsf{S\text{-}Prop}$, we have $\ell_x \approx 0 \in \mathsf{A}$. Thus, by Definition 12, $\mathcal{M}(\ell_x) = 0$, and so the length of x is 0 by Lemma 17. $\Leftarrow$: If $\mathcal{M}(x)$ has length 0, then by Lemma 17, $\mathcal{M}(\ell_x) = 0$. By saturation of $\mathsf{L\text{-}Valid}$, either $x \approx \epsilon \in \mathsf{S}$ or $\mathsf{A} \models \ell_x > 0$. But the latter is impossible by Definition 12. □

Next, we show how $\models^*_{++}$ works with update terms.

Lemma 27 *If* $x \approx \mathsf{update}(y, i, v) \in \mathsf{S}$, $\mathcal{M}(i) \in [0, \mathcal{M}(\ell_y))$, *and* $\mathsf{S} \models^*_{++} y \approx w_1 ++ \cdots ++ w_n$, *then:*

1. $x \approx z_1 ++ \cdots ++ z_n \in \mathsf{S}$ *for some atomic* $z_1, \ldots, z_n$*;*
2. *there exists some* $k \in [1, n]$*, such that* $\sum_{j=1}^{k-1} \mathcal{M}(\ell_{w_j}) \leq \mathcal{M}(i) < \sum_{j=1}^{k} \mathcal{M}(\ell_{w_j})$ *and* $z_k = \mathsf{update}(w_k, \alpha_k, v) \in \mathsf{S}$*, where* $\mathsf{A} \models \alpha_k \approx i - \sum_{j=1}^{k-1} \ell_{w_j}$*; and*
3. *for all* $j \in [1, n]$*,* $j \neq k$*,* $z_j \approx w_j \in \mathsf{S}$*.*

Proof By saturation w.r.t. $\mathsf{Update\text{-}Concat}$, there is $x \approx z_1 ++ \cdots ++ z_n \in \mathsf{S}$, for some $z_1, \ldots, z_n$, such that $z_m \approx \mathsf{update}(w_m, \alpha_m, v) \in \mathsf{S}$ for $m \in [1, n]$, where α_m is the variable introduced for the term $i - \sum_{j=1}^{m-1} \ell_{w_j}$ when flattening the term $\mathsf{update}(w_m, i - \sum_{j=1}^{m-1} \ell_{w_j}, v)$. Then, $z_m \approx \mathsf{update}(w_m, \alpha_m, v) \in \mathsf{S}$ and $\alpha_m \equiv_{\mathsf{S}} i - \sum_{l=1}^{m-1} \ell_{w_l}$. By saturation of $\mathsf{S\text{-}Prop}$, we also have $\mathsf{A} \models \alpha_m \approx i - \sum_{j=1}^{m-1} \ell_{w_j}$.

Next we prove that $z_1, ..., z_n$ are atomic. Suppose z_m is not atomic for some $m \in [1, n]$. Note that we cannot have $\mathsf{S} \models z_m \approx \epsilon$: w_m is atomic, so by Lemma 26, $\mathcal{M}(w_m)$ has a nonzero length; then, by Lemma 13, $\mathcal{M}(z_m)$ has nonzero length, so $\mathsf{S} \not\models z_m \approx \epsilon$ by Lemma 26. Let $\overline{\boldsymbol{u}}$ be the normal form of z_m. $\overline{\boldsymbol{u}}$ cannot be empty because then, by Lemma 25, the length of $\mathcal{M}(z_m)$ would be 0, so by Lemma 26, we would have $z_m \approx \epsilon$. $\overline{\boldsymbol{u}}$ cannot be of size 1 as then z_m would be atomic by Lemma 24. Thus, $\overline{\boldsymbol{u}}$ is of size at least 2. Now, by saturation of $\mathsf{Update\text{-}Concat\text{-}Inv}$ applied to $z_m \approx \mathsf{update}(w_m, \alpha_m, v)$, we have $w_m \approx \overline{\boldsymbol{z}}' \in \mathsf{S}$, $u_1 \approx \mathsf{update}(z_1', \alpha_m, v) \in \mathsf{S}$, and $u_2 \approx \mathsf{update}(z_2', \alpha', v) \in \mathsf{S}$, for some α' where $\mathsf{S} \models \alpha' = \alpha_m - \ell_{u_1}$. By Lemma 13, we have $\mathcal{M}(\ell_{u_1}) = \mathcal{M}(\ell_{z_1'})$ and $\mathcal{M}(\ell_{u_2}) \approx \mathcal{M}(\ell_{z_2'})$. But u_1 and u_2 are atomic, so by Lemma 26, their lengths cannot be zero. By Lemma 17, then, the lengths of z_1' and z_2' are also nonzero. So $\mathsf{S} \not\models z_1 \approx \epsilon$ and $\mathsf{S} \not\models z_2 \approx \epsilon$ by Lemma 26. Thus, $\overline{\boldsymbol{z}}$ is not singular, which means that w_m is not atomic, which is a contradiction.

Now, consider the following $n + 2$ constraints: $i < 0$, $\sum_{j=1}^{k-1} \ell_{w_j} \leq i < \sum_{j=1}^{k} \ell_{w_j}$ for $k \in [1, n]$, and $i \geq \sum_{j=1}^{n} \ell_{w_j}$. Exactly one of these holds in $\mathcal{M}$, since it interprets arithmetic symbols in the usual way by Definition 11.

Suppose $\mathcal{M} \models i < 0$ or $\mathcal{M} \models i \geq \sum_{j=1}^{n} \ell_{w_j}$. We know that $\mathcal{M}(i) \in [0, \mathcal{M}(\ell_y))$, so this case is impossible by Lemmas 17 and 25.

Now, suppose that $\mathcal{M} \models \sum_{j=1}^{k-1} \ell_{w_j} \leq i < \sum_{j=1}^{k} \ell_{w_j}$ for some $k \in [1, n]$. Clearly, Item 2 holds for this k. We know that $\mathsf{Update\text{-}Bound}$ is saturated w.r.t. $z_m \approx \mathsf{update}(w_m, \alpha_m, v) \in \mathsf{S}$ for $m \in [1, n]$. Recall also that $\mathsf{A} \models \alpha_m \approx i - \sum_{j=1}^{m-1} \ell_{w_j}$. It is easy to see that the first branch is inconsistent with $\mathcal{M}$ whenever $m \neq k$. Thus, we have $z_m \approx w_m \in \mathsf{S}$ for $m \in [1, n], m \neq k$. □

The next lemma proves that unit classes are atomic.

Lemma 28 *If $x \equiv_{\mathsf{S}} \mathsf{unit}(y)$ then x is atomic.*

Proof Note that $\mathsf{S} \models |\mathsf{unit}(y)| \approx 1$ by saturation of L-Intro, and so we also have $(*)$ $\mathsf{S} \models |x| \approx 1$. Assume x is not atomic. There are two cases. In the first case, $x \equiv_{\mathsf{S}} \epsilon$. But $\mathsf{S} \models |\epsilon| \approx 0$ by saturation of L-Intro, so by $(*)$ and saturation of S-Prop, this implies $\mathsf{A} \models 0 \approx 1$, which contradicts saturation of A-Conf. In the second case, there exists a variable concatenation term $x_1 ++ \cdots ++ x_n \in \mathcal{T}(\mathsf{S})$ such that $\mathsf{S} \models x \approx x_1 ++ \cdots ++ x_n$ and $x_1 ++ \cdots ++ x_n$ is not singular in S. By Lemma 4, we know that $\mathsf{A} \models_{\mathsf{LIA}} \Sigma_{i=1}^{n} \ell_{x_n} \geq 2$. But by $(*)$ and saturation of S-Prop and L-Intro, together with Assumption 1, we also have $\mathsf{A} \models \Sigma_{i=1}^{n} \ell_{x_n} = 1$, which also contradicts saturation of A-Conf. □

Next, we show when Item 2 of Definition 6 can be eliminated.

Definition 20 *Define $\mathsf{S} \models_{++}^{1,3} x \approx t$ if there is a derivation of $\mathsf{S} \models_{++} x \approx t$ without using Item 2 of Definition 6.*

Lemma 29 *If $\mathsf{S} \models_{++}^{*} x \approx \overline{\boldsymbol{y}}$, where y has size n, then for some $\overline{\boldsymbol{z}}$ of size n such that $z_i \equiv_{\mathsf{S}} y_i, i \in [1, n]$, $\mathsf{S} \models_{++}^{1,3} x \approx \overline{\boldsymbol{z}}$.*

Proof Since $\mathsf{S} \models_{++}^{*} x \approx \overline{\boldsymbol{y}}$, there is some $\overline{\boldsymbol{z}}'$ such that $z_i' \equiv_{\mathsf{S}} y_i, i \in [1, n]$ and $\mathsf{S} \models_{++} x \approx \overline{\boldsymbol{z}}'$. Consider the derivation tree described in Lemma 5, and let D be the path through the tree corresponding to the derivation of $\mathsf{S} \models_{++} x \approx \overline{\boldsymbol{z}}'$. If D has no application of Item 2 of Definition 6, then the claim is proved by setting $\overline{\boldsymbol{z}}$ to be $\overline{\boldsymbol{z}}'$. Otherwise, the first node in D must use Item 2 of Definition 6 to derive $\mathsf{S} \models_{++} x \approx t$, where $x \approx t \in \mathsf{S}$. Suppose that t is not singular. Then, it is possible to derive $\mathsf{S} \models_{++}^{1,3} x \approx t$ by starting with $\mathsf{S} \models_{++}^{1,3} x \approx x$ and then applying Item 3 of Definition 6, using $\mathsf{S} \models x \approx t$. We can then replace the root of D with this derivation to get a derivation showing $\mathsf{S} \models_{++}^{1,3} x \approx \overline{\boldsymbol{z}}$.

Suppose, on the other hand, that t is singular and $t = t_1 ++ \ldots ++ t_m$. Without loss of generality, assume that D eagerly applies Item 3 of Definition 6 $m-1$ times, each time using $\mathsf{S} \models t_i \approx \epsilon$ for some $i \in [1, m]$, which is possible because t is singular. The result is a derivation of $\mathsf{S} \models_{++} x \approx t_k$ for some variable t_k. We consider two cases.

1. Suppose that t_k is atomic with atomic representative v. Then, clearly we have $\mathsf{S} \models_{++}^{*} x \approx v$ and $\mathsf{S} \models_{++}^{*} t_k \approx v$, so by saturation of C-Eq, $x \equiv_{\mathsf{S}} t_k$. By Lemma 6, we also have that $\overline{\boldsymbol{y}} = v$. But then, let $\overline{\boldsymbol{z}} = x$. Clearly, we have $\mathsf{S} \models_{++}^{1,3} x \approx \overline{\boldsymbol{z}}$. Furthermore, $x \equiv_{\mathsf{S}} \overline{\boldsymbol{y}}$, proving the claim.
2. Suppose that t_k is not atomic. Then D must continue after $\mathsf{S} \models_{++} x \approx t_k$, and the next step must use Item 3 of Definition 6 using $\mathsf{S} \models t_k \approx t'$ to derive $\mathsf{S} \models_{++} x \approx t'$, where t' is ϵ or a variable concatenation term in S that is not singular in S. But then, note that we can start with $\mathsf{S} \models_{++}^{1,3} t_k \approx t_k$ and apply the same step to get $\mathsf{S} \models_{++}^{1,3} t_k \approx t'$. If we continue using the rest of the steps in derivation D, we can show that $\mathsf{S} \models_{++}^{1,3} t_k \approx \overline{\boldsymbol{z}}'$, and therefore, $\mathsf{S} \models_{++}^{*} t_k \approx \overline{\boldsymbol{y}}$. By saturation of C-Eq, we then have $x \equiv_{\mathsf{S}} t_k$. But, since $x \equiv_{\mathsf{S}} t_k$, this means that $x \equiv_{\mathsf{S}} t'$. So, we can start with $\mathsf{S} \models_{++}^{1,3} x \approx x$ and apply Item 3 of Definition 6 using $\mathsf{S} \models x \approx t'$ to get

$\mathsf{S} \models^{1,3}_{++} x \approx t'$. Using the same steps that appear in D after $t_k \approx t'$, we can show $\mathsf{S} \models^{1,3}_{++} x \approx \overline{\boldsymbol{z}}'$, which proves the claim. □

Lemma 30 *Let x be a sequence variable. Suppose $\mathsf{S} \models^{1,3}_{++} x \approx t$ and $\mathsf{S} \models_{++} z \approx (\overline{\boldsymbol{u}} ++ x ++ \overline{\boldsymbol{v}})\!\downarrow$. Then $\mathsf{S} \models_{++} z \approx (\overline{\boldsymbol{u}} ++ t ++ \overline{\boldsymbol{v}})\!\downarrow$.*

Proof By induction on the number of derivation steps in $\models^{1,3}_{++}$ that yield $\mathsf{S} \models^{1,3}_{++} x \approx t$ (see Definition 6). If this number is 1, then it must be by using Item 1 of Definition 6, so $x = t$ and the result follows trivially. If this number is some $n+1 > 1$, then consider the first n steps of the derivation. Let $\mathsf{S} \models^{1,3}_{++} x \approx s$ be their result. By the induction hypothesis, $\mathsf{S} \models_{++} z \approx (\overline{\boldsymbol{u}} ++ s ++ \overline{\boldsymbol{v}})\!\downarrow$. Now, consider the $n+1$ step of the derivation. It must replace some variable y in s by some term r, which results in t. Performing the same step on $\mathsf{S} \models_{++} z \approx (\overline{\boldsymbol{u}} ++ s ++ \overline{\boldsymbol{v}})\!\downarrow$ results in $\mathsf{S} \models_{++} z \approx (\overline{\boldsymbol{u}} ++ t ++ \overline{\boldsymbol{v}})\!\downarrow$. □

The next lemma shows how normal forms can be joined together.

Lemma 31 *Let $x_1, \ldots, x_k$ be sequence variables. Suppose $\mathsf{S} \models_{++} x \approx x_1 ++ \cdots ++ x_k$, and for every $i \in [1,k]$, $\mathsf{S} \models^{*}_{++} x_i \approx x_i^1 ++ \cdots ++ x_i^{n_i}$. Then $\mathsf{S} \models^{*}_{++} x \approx x_1^1 ++ x_1^2 ++ \cdots ++ x_k^{n_k}$.*

Proof We have $\mathsf{S} \models_{++} x \approx x_1 ++ \cdots ++ x_k$. Also, for every $i \in [1,k]$, since $\mathsf{S} \models^{*}_{++} x_i \approx x_i^1 ++ \cdots ++ x_i^{n_i}$, we also have, by Lemma 29, $\mathsf{S} \models^{1,3}_{++} x_i \approx y_i^1 ++ \cdots ++ y_i^{n_i}$ for some $y_i^1, \ldots, y_i^{n_i}$ such that $y_i^j \equiv_{\mathsf{S}} x_i^j$ for every $j \in [1, n_i]$. Considering the case where $i = 1$, by Lemma 30 we get $\mathsf{S} \models_{++} x \approx ((y_1^1 ++ \cdots ++ y_1^{n_1}) ++ x_2 ++ \cdots ++ x_k)\!\downarrow$. Continuing this way until $i = k$, and by the properties of $\downarrow$, we obtain $\mathsf{S} \models_{++} x \approx (y_1^1 ++ \cdots ++ y_k^{n_k})\!\downarrow$. Since $y_1^1, \ldots, y_k^{n_k}$ are variables, we actually have $\mathsf{S} \models_{++} x \approx y_1^1 ++ \cdots ++ y_k^{n_k}$. And hence, $\mathsf{S} \models^{*}_{++} x \approx x_1^1 ++ \cdots ++ x_k^{n_k}$. □

Finally, we can show that $\mathcal{M} \models \mathsf{S}$ by considering the various shapes of literals in S.

Proof of Lemma 23 Let $\varphi \in \mathsf{S}$. We prove that $\mathcal{M} \models \varphi$. By Assumption 1, φ is a flat sequence constraint. We consider the possible shapes of φ.

1. φ is $x \approx y$, for x, y of sort Int: by rule S-Prop, $x \approx y \in \mathsf{A}$. By Lemma 22, $\mathcal{M} \models \varphi$.
2. φ is $x \approx y$, for x, y of sort Elem: we have $x \equiv_{\mathsf{S}} y$, so by Definition 12, $\mathcal{M} \models \varphi$.
3. φ is $x \approx y$ where x, y have sort Seq: Definitions 13, 14, 17 and 18 are defined for equivalence classes. Since x and y are in the same equivalence class, $\mathcal{M} \models \varphi$.
4. φ is $x \not\approx y$ where x, y have sort Int: by saturation of S-Prop, $x \approx x, y \approx y \in \mathsf{A}$. Hence, $x, y \in \mathcal{T}(\mathsf{S}) \cap \mathcal{T}(\mathsf{A})$. By saturation of S-A, either $x \approx y \in \mathsf{A}$ or $x \not\approx y \in \mathsf{A}$. In the first case, by saturation of A-Prop, we also have $x \approx y \in \mathsf{S}$, which is impossible by saturation of S-Conf. Hence, we have $x \not\approx y \in \mathsf{A}$. By Lemma 22, $\mathcal{M} \models \varphi$.
5. φ is $x \not\approx y$ where x, y have sort Elem: x is not equivalent to y w.r.t. $\equiv_{\mathsf{S}}$, so Definition 12 assigns them different values; thus, $\mathcal{M}(x) \neq \mathcal{M}(y)$ and $\mathcal{M} \models \varphi$.
6. φ is $x \not\approx y$ where x, y have sort Seq: $\mathcal{M} \models \varphi$ by Lemma 20.
7. φ is $x \approx y + z$, $x \approx -y$, or $x \approx n$ for some $n \in \mathbb{N}$: by saturation of S-Prop, $\varphi \in \mathsf{A}$, so $\mathcal{M} \models \varphi$ by Lemma 22.

8. φ is $x \approx \epsilon$: we know that $\mathsf{S} \models |\epsilon| = 0$ by saturation of L-Intro. Using Assumption 1, we get $\mathsf{S} \models \ell_x \approx 0$, and so by saturation of S-Prop we have $\mathsf{A} \models \ell_x = 0$. It follows by Lemma 17 that $\mathcal{M}(x)$ has length 0 and is thus the empty sequence.
9. φ is $x \approx \mathsf{unit}(y)$: By Lemma 28, $[x]$ is atomic. Also, $[x]$ is a unit equivalence class. Hence $\mathcal{M}([x])$ was defined in Definition 14 and was set to a sequence of size 1 whose only element is $\mathcal{M}(y)$ by Lemma 12. Therefore, $\mathcal{M} \models \varphi$.
10. φ is $x \approx |y|$: we know that $\ell_y \approx |y| \in \mathsf{S}$ by Assumption 1, so $\ell_y \approx x \in \mathsf{A}$ by saturation of S-Prop. From Definition 12, it follows that $\mathcal{M}(\ell_y) = \mathcal{M}(x)$. But by Lemma 17, we also have $\mathcal{M}(|y|) = \mathcal{M}(\ell_y)$, so $\mathcal{M}(x) = \mathcal{M}(|y|)$.
11. φ is $x \approx x_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} x_n$. Suppose that $x_1, \ldots, x_n$ have the unique (by Lemma 6) normal forms $\mathsf{S} \models^*_{+\!+} x_1 \approx u_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} u_{m_1}$, $\mathsf{S} \models^*_{+\!+} x_2 \approx u_{m_1+1} \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} u_{m_2}$, $\ldots$, $\mathsf{S} \models^*_{+\!+} x_n \approx u_{m_{n-1}+1} \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} u_{m_n}$. By Item 2 of Definition 6, we know $\mathsf{S} \models_{+\!+} x \approx x_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} x_n$, so by Lemma 31, we have $\mathsf{S} \models^*_{+\!+} x \approx u_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} u_{m_n}$. By Lemma 25, then $\mathcal{M}(x) = \mathcal{M}(u_1) \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} \mathcal{M}(u_{m_n})$. Also, by Lemma 25, for each $i \in [1, n]$, $\mathcal{M}(x_i) = \mathcal{M}(u_{m_{i-1}+1}) \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} \mathcal{M}(u_{m_i})$. So, $\mathcal{M}(x_1 \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} x_n) = \mathcal{M}(x_1) \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} \mathcal{M}(x_n) = \mathcal{M}(u_1) \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} \mathcal{M}(u_{m_n}) = \mathcal{M}(x)$.
12. φ is $x \approx \mathsf{nth}(y, i)$: We consider two cases:
    (a) $\mathcal{M}(i)$ is negative or is not smaller than the length of $\mathcal{M}(y)$: applying Definition 19 with $\mathcal{M}(y)$ for a, $\mathcal{M}(i)$ for i and x for itself, we get that $\mathcal{M} \models \varphi$.
    (b) $\mathcal{M}(i)$ is non-negative and smaller than the length of $\mathcal{M}(y)$: By Lemma 19 with x for k, y for x and i for y, we have that the $\mathcal{M}(i)$th element of $\mathcal{M}(y)$ is $\mathcal{M}(x)$. Therefore, $\mathcal{M} \models \varphi$.
13. φ is $x \approx \mathsf{update}(y, i, z)$: First, assume $\mathcal{M}(i)$ is negative or not smaller than the length of $\mathcal{M}(y)$. In this case, the interpretation in $\mathcal{M}$ of $\mathsf{update}(y, i, z)$ is $\mathcal{M}(y)$. Hence, we prove that $\mathcal{M}(x) = \mathcal{M}(y)$. By saturation of Update-Bound, we have that either $\mathsf{A} \models 0 \leq i < \ell_y$ or $x \approx y \in \mathsf{S}$. The first case is impossible by Definitions 11 and 12 and Lemma 17, and hence, the second case holds. We therefore have $x \equiv_{\mathsf{S}} y$, and since the definitions of sequence variables are done by equivalence classes (see Definitions 14, 17 and 18), we have $\mathcal{M}(x) = \mathcal{M}(y)$.
    Next, assume $\mathcal{M}(i)$ is non-negative and smaller than the length of $\mathcal{M}(y)$ (this also implies that $\mathcal{M}(y)$ is not an empty sequence). By Lemma 13, we have $\mathcal{M}(\ell_x) = \mathcal{M}(\ell_y)$, and by Lemma 17, $\mathcal{M}(|x|) = \mathcal{M}(\ell_x)$ and $\mathcal{M}(|y|) = \mathcal{M}(\ell_y)$. We consider the following cases:
    (a) $[x]$ and $[y]$ are atomic in S: We show that for every $j \in [0, \mathcal{M}(\ell_x))$, the jth element of $\mathcal{M}(x)$ is the same as the jth element of $\mathcal{M}(\mathsf{update}(y, i, z))$.
    First, suppose $j = \mathcal{M}(i)$. We show that the jth element of $\mathcal{M}(x)$ is $\mathcal{M}(z)$. By saturation of Nth-Intro, we have $\mathsf{nth}(x, i) \in \mathcal{T}(\mathsf{S})$. Since $x \approx \mathsf{update}(y, i, z) \in \mathsf{S}$, rule Nth-Update applies. Its first and last branches are impossible: the first by our assumption on $\mathcal{M}(i)$ and Definitions 11 and 12 and Lemma 17, and the last because it would require $\mathsf{A} \models i \neq i$, but we know $\mathsf{A} \not\models \perp$ by saturation of A-Conf. Hence, the middle branch applies, which means $\mathsf{nth}(x, i) \approx z \in \mathsf{S}$. By Lemma 18, then, we know that the jth element of $\mathcal{M}(x)$ is $\mathcal{M}(z)$.
    Suppose, on the other hand, that $j \neq \mathcal{M}(i)$. This implies $\mathcal{M}(\ell_x) = \mathcal{M}(\ell_y) \geq 2$, so $[x]$ and $[y]$ cannot be unit by Definition 14. Their values are thus set by Definition 17. Since $x \approx \mathsf{update}(y, i, z) \in \mathsf{S}$, by Definition 15, there is an edge d between $[x]$ and $[y]$ with $\mathcal{M}(i) \in \delta(d)$. It follows, because $j \neq \mathcal{M}(i)$, that $[x] \sim_j [y]$ by Definition 16. But then $[x]$ and $[y]$ are in the same equivalence class of $\sim_j$, so in either case of Definition 17, their jth value is set to the same value. Finally, because $j \neq \mathcal{M}(i)$, the jth element of $\mathcal{M}(\mathsf{update}(y, i, z))$ is (according to the semantics of update) the jth element of $\mathcal{M}(y)$.

(b) Suppose $\mathsf{S} \models^*_{++} y \approx w_1 ++ \cdots ++ w_n$. By Lemma 27, we have $x \approx z_1 ++ \cdots ++ z_n \in \mathsf{S}$ for some atomic $z_1, \dots, z_n$, $\sum_{j=1}^{k-1} \mathcal{M}(\ell_{w_j}) \leq \mathcal{M}(i) < \sum_{j=1}^{k} \mathcal{M}(\ell_{w_j})$ and $z_k = \mathsf{update}(w_k, \alpha_k, z) \in \mathsf{S}$ for some $k \in [1, n]$, where $\mathsf{A} \models \alpha_k \approx i - \sum_{j=1}^{k-1} \ell_{w_j}$, and for $m \in [1, n], m \neq k$, $z_m \approx w_m \in \mathsf{S}$. Since Definitions 14 and 17 assign equivalence classes, we also have $\mathcal{M}(z_m) = \mathcal{M}(w_m)$.

Since z_k, w_k are atomic, we have $\mathcal{M}(z_k) = \mathcal{M}(\mathsf{update}(w_k, \alpha_k, z))$ by Item 13(a) above. By Lemma 22, we have $\mathcal{M}(\alpha_k) = \mathcal{M}(i) - \sum_{j=1}^{k-1} \mathcal{M}(\ell_{w_j})$. By Item 11, above, $\mathcal{M}(x) \approx \mathcal{M}(z_1) ++ \ldots ++ \mathcal{M}(z_n)$. It follows that $\mathcal{M}(x) = \mathcal{M}(w_1) ++ \ldots ++ \mathcal{M}(\mathsf{update}(w_k, \alpha_k, z)) ++ \ldots ++ \mathcal{M}(w_n)$. By Lemma 24, we also have $\mathcal{M}(y) = \mathcal{M}(w_1) ++ \ldots ++ \mathcal{M}(w_n)$, so $\mathcal{M}(\mathsf{update}(y, i, z)) = \mathcal{M}(w_1) ++ \ldots ++ \mathcal{M}(\mathsf{update})(\mathcal{M}(w_k), \mathcal{M}(i) - \sum_{j=1}^{k-1} \mathcal{M}(\ell_{w_j}), \mathcal{M}(z)) ++ \ldots ++ \mathcal{M}(w_n)$. It follows that $\mathcal{M}(x) = \mathcal{M}(\mathsf{update}(y, i, z))$.

14. φ is $x \approx \mathsf{extract}(y, i, j)$: By saturation w.r.t. R-Extract, we have two options.
In the first, $\mathsf{A} \models i < 0 \vee i \geq \ell_y \vee j \leq 0$ and $x \approx \epsilon \in \mathsf{S}$. By Item 8, above, $\mathcal{M}(x)$ is the empty sequence. By Lemmas 17 and 22, $\mathcal{M}(i) < 0$ or $\mathcal{M}(i)$ has at least the length of $\mathcal{M}(y)$ or $\mathcal{M}(j) \leq 0$. In each of these 3 cases, we get from the semantics of $\mathsf{extract}$ in T_{Seq} that $\mathcal{M}(\mathsf{extract})$ assigns the empty sequence w.r.t the inputs $\mathcal{M}(y)$, $\mathcal{M}(i)$ and $\mathcal{M}(j)$. Hence in this case we get $\mathcal{M} \models \varphi$.
In the second case, $\mathsf{A} \models 0 \leq i < \ell_y \wedge j > 0 \wedge \ell_k \approx i \wedge \ell_x \approx \min(j, \ell_y - i) \wedge \ell_{k'} \approx \ell_y - \ell_x - i$ and $y \approx k ++ x ++ k' \in \mathsf{S}$. We thus have $\mathcal{M}(y) = \mathcal{M}(k) ++ \mathcal{M}(x) ++ \mathcal{M}(k')$ by Item 11, above. Also, $\mathcal{M} \models \mathsf{A}$, by Lemma 22. According to the semantics of $\mathsf{extract}$ in T_{Seq}, since $\mathcal{M}(i), \mathcal{M}(j) \geq 0$, $\mathcal{M}(i)$ is smaller than the length of $\mathcal{M}(y)$, the value assigned in $\mathcal{M}$ to $\mathsf{extract}(y, i, j)$ is the maximal sub-sequence of $\mathcal{M}(y)$ that starts at index $\mathcal{M}(i)$ and has length at most $\mathcal{M}(j)$. Since $\mathcal{M} \models y \approx k ++ x ++ k'$ with the appropriate length constraints (by Lemma 17), we have that this sequence value is exactly $\mathcal{M}(x)$ and hence, $\mathcal{M} \models \varphi$. $\square$

6 Implementation

We implemented our procedure for sequences in the cvc5 SMT solver as an extension of cvc5's theory solver for strings [21, 26]. We have generalized that theory solver to reason about both strings and sequences. In this section, we describe how the rules of the calculus are implemented and the overall strategy for when they are applied.

Like most SMT solvers, cvc5 is based on the CDCL(T) architecture [23] which combines several subsolvers, each specialized on a specific theory, with a solver for propositional satisfiability (SAT). Following that architecture, cvc5 maintains an evolving set of formulas F. If F consists of quantifier-free formulas over the theory T_{Seq}, the case targeted by this work, the SAT solver searches for a satisfying assignment for F at the Boolean level, represented as a set M of literals whose atoms come from F. If none exists, the problem is unsatisfiable at the propositional level and hence also T_{Seq}-unsatisfiable. Otherwise, M is partitioned into the arithmetic constraints A and the sequence constraints S and checked for T_{Seq}-satisfiability using the rules of the EXT calculus. Many of those rules, including all those with multiple conclusions, are implemented by adding new formulas to F according to the splitting-on-demand approach [4].

This causes the SAT solver to try to extend its assignment to those formulas, which results in the addition of new literals to M (and thereby also to A and S).

In this setting, the rules of the two calculi are implemented as follows. The effect of rule A-Conf is achieved by invoking cvc5's theory solver for linear integer arithmetic. Rule S-Conf is implemented by the congruence closure submodule. Rules A-Prop and S-Prop are implemented by a standard mechanism for theory combination, namely sharing equalities, as in the Nelson-Oppen method.

Note that each of these four rules may be applied *eagerly*, i.e., before constructing a complete satisfying assignment M for F.

The remaining rules are implemented in the theory solver for sequences. Each time M is checked for satisfiability, cvc5 follows a strategy to determine which rule to apply next. If none of the rules apply and the configuration is different from unsat, then it is saturated, and the solver returns sat. The strategy for EXT prioritizes rules as follows. Only the first applicable rule is applied, and then control goes back to the SAT solver.

1. (Add length constraints) For each sequence term in S, apply L-Intro for non-variables and L-Valid for variables, if not already done.
2. (Mark congruent terms) For each set of update (resp. nth) terms that are congruent to one another in the current configuration, mark all but one term and ignore the marked terms in the subsequent steps.
3. (Reduce extract) For extract(y, i, j) in S, apply R-Extract if not already done.
4. (Construct normal forms) Apply U-Eq or C-Split. We choose how to apply the latter rule based on constructing normal forms for equivalence classes in a bottom-up fashion, where the equivalence classes of x and y are considered before the equivalence class of $x \mathbin{+\!\!+} y$. We do this until we find an equivalence class such that $\mathsf{S} \models^*_{++} z \approx u_1$ and $\mathsf{S} \models^*_{++} z \approx u_2$ for distinct u_1, u_2.
5. (Normal forms) Apply C-Eq to two variables if they have the same normal form but are in different equivalence classes.
6. (Extensionality) For each disequality in S, apply Deq-Ext, if not already done.
7. (Distribute update and nth) For each term update(x, i, t) (resp. nth(x, j)) such that the normal form of x is a concatenation term, apply Update-Concat and Update-Concat-Inv (resp. Nth-Concat) if not already done. Alternatively, if the normal form of the equivalence class of x is a unit term, apply Update-Unit (resp. Nth-Unit).
8. (Array reasoning on atomic sequences) Apply Nth-Intro and Update-Bound to update terms. For each update term, find the matching nth terms and apply Nth-Update. Apply Nth-Split to pairs of nth terms with equivalent indices.
9. (Integer arrangement) Apply S-A to two arithmetic terms occurring in S and A.

Whenever a rule is applied, the strategy will restart from the beginning in the next iteration. The strategy is designed to apply with higher priority steps that are easy to compute and are likely to lead to conflicts. Some steps are ordered based on dependencies from other steps. For instance, Steps 5 and 7 use normal forms, which are computed in Step 4. The strategy for the BASE calculus is the same, except that Steps 7 and 8 are replaced by one that applies R-Update and R-Nth to all update and nth terms in S.

We point out that the C-Split rule may cause non-termination of the proof strategy described above in the presence of *cyclic* sequence constraints, for instance, constraints where sequence variables appear on both sides of an equality. The solver uses methods for detecting some of these cycles, to restrict when C-Split is applied. In particular, when $\mathsf{S} \models^*_{++} x \approx (\overline{\boldsymbol{u}} +\!\!+ s +\!\!+ \overline{\boldsymbol{w}})\!\downarrow$, $\mathsf{S} \models^*_{++} x \approx (\overline{\boldsymbol{u}} +\!\!+ t +\!\!+ \overline{\boldsymbol{v}})\!\downarrow$, and s occurs in $\overline{\boldsymbol{v}}$, then C-Split is not applied. Instead, other heuristics are used, and in some cases the solver terminates with a response of "unknown" (see e.g., [21] for details). In addition to the version shown here, we also use another variation of the C-Split rule where the normal forms are matched in reverse (starting from the last terms in the concatenations). The implementation also uses fast entailment tests for length inequalities. These tests may allow us to conclude which branch of C-Split, if any, is feasible, without having to branch on cases explicitly.

Although not shown here, the calculus can also accommodate certain *extended* sequence constraints, that is, constraints using a signature with additional functions. For example, our implementation supports sequence containment, replacement, and reverse. It also supports an extended variant of the update operator, in which the third argument is a sequence that overrides the sequence being updated starting from the index given in the second argument. Constraints involving these functions are handled by reduction rules similar to those in Figure 5. The implementation is further optimized by using context-dependent simplifications, which may eagerly infer when certain sequence terms can be simplified to constants based on the current set of assertions [26].

7 Evaluation

We evaluate the performance of our approach, as implemented in cvc5. The evaluation investigates:

(i) whether the use of sequences is a viable option for reasoning about vectors in programs,
(ii) how our approach compares with other sequence solvers, and
(iii) the performance impact of our array-style extended rules.

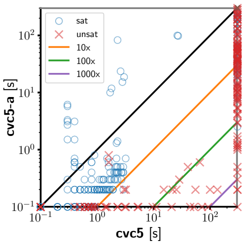
As a baseline, we use version 4.8.14 of the Z3 SMT solver, which supports a theory of sequences without updates. For cvc5, we evaluate implementations of both the basic calculus (denoted **cvc5**) and the extended array-based calculus (denoted **cvc5-a**). The benchmarks, solver configurations, and logs from our runs are available for download.[7] We ran all experiments on a cluster equipped with Intel Xeon E5-2620 v4 CPUs. We allocated one physical CPU core and 8GB of RAM for each solver-benchmark pair and used a time limit of 300 seconds. We use the following two sets of benchmarks:

Array Benchmarks (Arrays) The first set of benchmarks is derived from the QF_AX benchmarks in SMT-LIB [3]. To generate these benchmarks, we

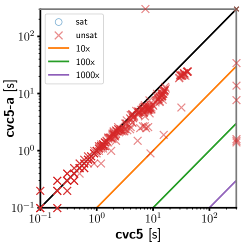
[7] http://dx.doi.org/10.5281/zenodo.6146565

Set		w/ update cvc5	cvc5-a	z3
Arrays (551)	Slvd	242	**390**	170
	Time	162	303	4329
Diem (558)	Slvd	542	**547**	443
	Time	518	440	639

(a)



(b) Arrays



(c) Diem

Fig. 7: Figure 7a lists the number of solved benchmarks and total time on commonly solved benchmarks. The scatter plots compare the base solver (**cvc5**) and the extended solver (**cvc5-a**) on Arrays and Diem benchmarks.

(i) replace declarations of arrays with declarations of sequences of uninterpreted sorts, (ii) change the sort of index terms to integers, and (iii) replace store with update and select with nth. The resulting benchmarks are quantifier-free and do not contain concatenations. Note that the original and the derived benchmarks are not T_{Seq}-equisatisfiable, because sequences take into account out-of-bounds cases that do not occur in arrays. For the Z3 runs, we add to the benchmarks a definition of update in terms of extraction and concatenation.

Smart Contract Verification (Diem) The second set of benchmarks consists of verification conditions generated by running the Move Prover [28] on smart contracts written for the Diem framework. By default, the encoding does not use the sequence update operation, so Z3 can be used directly. To generate benchmarks that use the update operator, we modified the Move Prover encoding. In addition to using the sequence theory, the benchmarks make heavy use of quantifiers and the SMT-LIB theory of datatypes.

Figure 7a summarizes the results in terms of number of solved benchmarks and total time in seconds on commonly solved benchmarks. The configuration that solves the largest number of benchmarks is the implementation of the extended calculus (**cvc5-a**). This approach also successfully solves most of the Diem benchmarks, which suggests that sequences are a promising option for encoding vectors in programs. The results further show that the sequences solver of cvc5 significantly outperforms Z3 on both the number of solved benchmarks and the solving time on commonly solved benchmarks.

Figures 7b and 7c show scatter plots comparing **cvc5** and **cvc5-a** on the two benchmark sets. We can see a clear trend towards better performance when using the extended solver. In particular, the table shows that in addition to solving the most benchmarks, **cvc5-a** is also fastest on the commonly solved instances from the Diem benchmark set.

For the Arrays set, we can see that some benchmarks are slower with the extended solver. This is also reflected in the table, where **cvc5-a** is slower on the commonly solved instances. This is not too surprising, as the extra machinery of the extended solver can sometimes slow down easy problems. As problems get harder, however, the benefit of the extended solver becomes clear. For example, if we drop Z3 and consider just the commonly solved instances between **cvc5** and **cvc5-a** (of which there are 242), **cvc5-a** is about 2.47× faster (426 vs 1053 seconds). Of course, further improving the performance of **cvc5-a** is something we plan to explore in future work.

8 Conclusion

We introduced calculi for checking satisfiability in the theory of sequences, which can be used to model the vector data type. We described our implementation in cvc5 and provided an evaluation, showing that the proposed theory is rich enough to naturally express verification conditions without introducing quantifiers, and that our implementation is efficient. We believe that verification tools can benefit by changing their encoding of verification conditions that involve vectors to use the proposed theory and implementation.

We plan to propose the incorporation of this theory in the SMT-LIB standard and contribute our benchmarks to SMT-LIB. As future research, we plan to integrate other approaches for array solving into our basic solver. We also plan to study the politeness [20, 24] and decidability of various fragments of the theory of sequences.

Acknowledgments

This work was funded in part by the Stanford Center for Blockchain Research, NSF-BSF grant numbers 2110397 (NSF) and 2020704 (BSF), ISF grant number 619/21, and Meta Novi. Part of the work was done when the first author was an intern at Meta Novi.

References

[1] Francesco Alberti, Silvio Ghilardi, and Elena Pagani. Cardinality constraints for arrays (decidability results and applications). *Formal Methods Syst. Des.*, 51(3):545–574, 2017.

[2] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *TACAS (1)*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.

[3] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[4] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in SAT modulo theories. In M. Hermann and A. Voronkov, editors, *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'06), Phnom Penh, Cambodia*, volume 4246 of *Lecture Notes in Computer Science*, pages 512–526. Springer, 2006.

[5] Clark W. Barrett, Igor Shikanian, and Cesare Tinelli. An abstract decision procedure for a theory of inductive data types. *J. Satisf. Boolean Model. Comput.*, 3(1-2):21–46, 2007.

[6] Clark W. Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[7] Murphy Berzish, Vijay Ganesh, and Yunhui Zheng. Z3str3: A string solver with theory-aware heuristics. In Daryl Stewart and Georg Weissenbacher, editors, *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, pages 55–59. IEEE, 2017.

[8] Nikolaj Bjørner, Leonardo de Moura, Lev Nachmanson, and Christoph Wintersteiger. Programming Z3. https://theory.stanford.edu/~nikolaj/programmingz3.html#sec-sequences-and-strings, 2018.

[9] Nikolaj Bjørner, Vijay Ganesh, Raphaël Michel, and Margus Veanes. An SMT-LIB format for sequences and regular expressions. *SMT*, 12:76–86, 2012.

[10] Nikolaj Bjørner, Nikolai Tillmann, and Andrei Voronkov. Path feasibility analysis for string-manipulating programs. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 2009.

[11] Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. CDSAT for nondisjoint theories with shared predicates: Arrays with abstract length. In David Déharbe and Antti E. J. Hyvärinen, editors, *Proceedings of the 20th Internal Workshop on Satisfiability Modulo Theories co-located with the 11th International Joint Conference on Automated Reasoning (IJCAR 2022) part of the 8th Federated Logic Conference (FLoC 2022), Haifa, Israel, August 11-12, 2022*, volume 3185 of *CEUR Workshop Proceedings*, pages 18–37. CEUR-WS.org, 2022.

[12] Jürgen Christ and Jochen Hoenicke. Weakly equivalent arrays. In *FroCos*, volume 9322 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 2015.

[13] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[14] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Commun. ACM*, 22(8):465–476, 1979.

[15] Neta Elad, Sophie Rain, Neil Immerman, Laura Kovács, and Mooly Sagiv. Summing up smart transitions. In *CAV (1)*, volume 12759 of *Lecture Notes in Computer Science*, pages 317–340. Springer, 2021.

[16] Herbert B. Enderton. *A mathematical introduction to logic.* Academic Press, 2 edition, 2001.

[17] Stephan Falke, Florian Merz, and Carsten Sinz. Extending the theory of arrays: memset, memcpy, and beyond. In *VSTTE*, volume 8164 of *Lecture Notes in Computer Science*, pages 108–128. Springer, 2013.

[18] Vijay Ganesh, Mia Minnes, Armando Solar-Lezama, and Martin C. Rinard. Word equations with length constraints: What's decidable? In *Haifa Verification Conference*, volume 7857 of *Lecture Notes in Computer Science*, pages 209–226. Springer, 2012.

[19] Silvio Ghilardi, Alessandro Gianola, and Deepak Kapur. Interpolation and amalgamation for arrays with maxdiff. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 268–288. Springer, 2021.

[20] Dejan Jovanovic and Clark W. Barrett. Polite theories revisited. In Christian G. Fermüller and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings*, volume 6397 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2010.

[21] Tianyi Liang, Andrew Reynolds, Cesare Tinelli, Clark W. Barrett, and Morgan Deters. A DPLL(T) theory solver for a theory of strings and regular expressions. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 646–662. Springer, 2014.

[22] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.

[23] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: from an abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.

[24] Silvio Ranise, Christophe Ringeissen, and Calogero G. Zarba. Combining data structures with nonstably infinite theories using many-sorted logic. In Bernhard Gramlich, editor, *Frontiers of Combining Systems, 5th International Workshop, FroCoS 2005, Vienna, Austria, September 19-21, 2005, Proceedings*, volume 3717 of *Lecture Notes in Computer Science*, pages 48–64. Springer, 2005. Extended technical report is available at https://hal.inria.fr/inria-00070335/.

[25] Andrew Reynolds, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. Reductions for strings and regular expressions revisited. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 225–235. IEEE, 2020.

[26] Andrew Reynolds, Maverick Woo, Clark W. Barrett, David Brumley, Tianyi Liang, and Cesare Tinelli. Scaling up DPLL(T) string solvers using context-dependent simplification. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2017.

[27] Ying Sheng, Andres Nötzli, Andrew Reynolds, Yoni Zohar, David L. Dill, Wolfgang Grieskamp, Junkil Park, Shaz Qadeer, Clark W. Barrett, and Cesare Tinelli. Reasoning about vectors using an SMT theory of sequences. In *IJCAR*, volume 13385 of *Lecture Notes in Computer Science*, pages 125–143. Springer, 2022.

[28] Jingyi Emma Zhong, Kevin Cheang, Shaz Qadeer, Wolfgang Grieskamp, Sam Blackshear, Junkil Park, Yoni Zohar, Clark Barrett, and David L. Dill. The Move prover. In Shuvendu K. Lahiri and Chao Wang, editors, *Proceedings of the 32nd International Conference on Computer Aided Verification (CAV '20)*, volume 12224 of *Lecture Notes in Computer Science*, pages 137–150. Springer International Publishing, July 2020.