HAL
open science

# Schemata of Formulæ in the Theory of Arrays

Nicolas Peltier

## To cite this version:

Nicolas Peltier. Schemata of Formulæ in the Theory of Arrays. TABLEAUX 2013 - 22th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Sep 2013, Nancy, France. pp.234-249, 10.1007/978-3-642-40537-2_20 . hal-00934604

**HAL Id: hal-00934604**

**https://hal.science/hal-00934604v1**

Submitted on 22 Jan 2014

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Schemata of Formulæ in the Theory of Arrays*

Nicolas Peltier

CNRS - Grenoble Informatics Laboratory
Nicolas.Peltier@imag.fr

**Abstract.** We consider schemata of quantifier-free formulæ, defined using indexed symbols and iterated connectives ranging over intervals (such as $\bigvee_{i=1}^{n} \phi$ or $\bigwedge_{i=1}^{n} \phi$ ), and interpreted in the theory of arrays (with the usual functions for storing and selecting elements in an array). We first prove that the satisfiability problem is undecidable (it is clearly semi-decidable). We then consider a natural restriction on the considered structures and we prove that it makes the logic decidable by providing a sound, complete and terminating proof procedure.

## 1 Introduction

In [2] (see also [3]) the logic of *iterated schemata* is defined, which enriches the language of propositional logic with arithmetic parameters, indexed variables and iterated connectives ranging over intervals of natural numbers. This language allows one to formally define families of formulæ depending on arithmetic parameters, such as, e.g., the parameterized formula $(p_0 \wedge \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} p_i \Rightarrow p_{i+1}) \Rightarrow p_{\mathtt{n}+1}$. Decidability and undecidability results have been obtained for the proposed language (according to the form of the arithmetic expressions that occur in the formulæ) and proof procedures have been devised to test the validity of schemata of formulæ. These procedures use the usual decomposition rules of propositional logic, together with rules performing a lazy instantiation of the parameters and loop detection techniques encoding a limited form of mathematical induction. In [5], these results have been extended to some theories beyond propositional logic, including in particular a fragment of Presburger arithmetic. In the present paper, we consider schemata of formulæ interpreted over the theory of arrays, that plays a central role in program verification[1]. The theory of arrays is defined by using two function symbols *store* and *select* encoding respectively the storage and retrieving of an element in an array, and defined by the two following axioms:

$$\forall x, z, v,\ \ select(store(x, z, v), z) \simeq v \tag{1}$$

$$\forall x, z, w, v,\ \ z \simeq w \vee select(store(x, z, v), w) \simeq select(x, w) \tag{2}$$

These axioms state that if an element $v$ is inserted into an array $x$ at some position $z$, then the resulting array contains $v$ at position $z$ (first axiom) and contains the same elements as in $x$ elsewhere (second axiom).

* This work has been partly funded by the project ASAP of the French *Agence Nationale de la Recherche* (ANR-09-BLAN-0407-01).

[1] This theory does not fall in the scope of the results in [5].

This theory allows for instance to model the heap when describing the behavior of a program. It is very well-studied and several procedures have been proposed to test the validity of formulæ modulo this theory and many extensions. For instance, a decision procedure is defined in [9, 8] for dealing with formulæ over the theory of arrays with uninterpreted or interpreted elements and indices, which is able to handle a restricted form of quantification over indices. The work described in [10] focuses on arrays with integer indices and devises a method to combine existing decision procedures which makes it possible to handle some important features of arrays such as sortedness or array dimension. We also mention the logic presented in [11], which is devoted to reasoning with arrays of integers (by reduction to the emptiness problem for counter automata).

In the present paper, we consider schemata of formulæ interpreted on the theory of arrays and defined using indexed variables and iterated connectives. These indexed symbols may be used, for instance, to denote the state of the same variable at different times during the running of a program. Consider for instance the following example. If, in an array $T$, $n$ elements $e_1, \ldots, e_n$ are inserted at *distinct* indices $i_1, \ldots, i_n$, then it is clear that the result does not depend on the order in which the insertion is performed. A particular instance of this problem can be modeled by constructing the following two sequences $T'_n$ and $T_n$:

$$\begin{array}{lll} T_0 \simeq T & T'_0 \simeq \mathit{store}(T, i', a) & \bigwedge_{j=0}^{n} i_j \neq i' \\ \bigwedge_{j=0}^{n} T_{j+1} \simeq \mathit{store}(T_j, i_j, e_j) & \bigwedge_{j=0}^{n} T'_{j+1} \simeq \mathit{store}(T'_j, i_j, e_j) & \end{array}$$

Intuitively, $T_{n+1}$ is obtained from $T$ by inserting the elements $e_1, \ldots, e_n$ in the cells $i_1, \ldots, i_n$ respectively, and $T'_{n+1}$ is obtained by inserting $a, e_1, \ldots, e_n$ at $i', i_1, \ldots, i_n$. We want to prove that the array obtained by storing $a$ into $T_{n+1}$ in cell $i'$ is identical to $T'_{n+1}$ modulo extensionality, i.e. that $\forall n\ \forall i\ \mathit{select}(\mathit{store}(T_{n+1}, i', a), i) \simeq \mathit{select}(T'_{n+1}, i)$ is a logical consequence of the previous set of axioms. Note that the index variables $j$ and $n$ must be interpreted as natural numbers hence this problem cannot be encoded in first-order logic[2]. By negating the conclusion, we obtain a clause containing a constant interpreted as a natural number. Existing SMT-solvers (see, e.g., `www.SMT-LIB.org`) cannot decide the validity of such formulæ, unless of course $n$ is instantiated to some fixed number[3]. For proving that the formula holds for all values of $n$, the use of mathematical induction is essential.

The paper is structured as follows. After defining in Section 2 the syntax and the semantics of the considered logic, we prove in Section 3 that the satisfiability problem is undecidable (it is obviously semi-decidable), by reduction to the Post correspondence problem. Remarkably, this result still holds if the syntax of the formula is further restricted by forbidding arrays containing elements of the same sort as their indices (thus discarding terms such as $\mathit{select}(T, \mathit{select}(T, i))$).

[2] It does not fall in the scope of existing procedures for handling combination of first-order logic and Presburger arithmetic (see, e.g., [7]), since it necessarily involves induction.

[3] As far as we are aware, this problem is out of the scope of the known decidable extensions of the theory of arrays.

In Section 4, we impose additional semantic restrictions on the considered interpretations and define a decision procedure for testing the satisfiability of formulæ in this particular class of interpretations. The proof procedure is based on propositional tableaux enriched by specific rules for equality reasoning modulo the theory of arrays, together with new simplification and loop detection rules ensuring termination. The conditions on the interpretations that make possible the definition of the decision procedure are formally defined in Section 4.2. Informally, these conditions can be summarized as follows: two indexed constants $a_i$ and $b_j$, with $i < j$ encoding array indices cannot be equal, unless there exists a non-indexed constant $c$ such that $a_i = c = b_j$ or a sequence of constants $c^1, \ldots, c^{j-1-i}$ such that $a_i = c^1_{i+1} = \ldots = c^{j-i-1}_{j-1} = b_j$ holds in the considered interpretation. This restriction does not apply in a systematic way on all the symbols occurring in the formula, but only on those encoding the indices of certain arrays, more precisely those on which a *store* operation is performed, and those containing elements of the same sort as their indices. We call the interpretations satisfying this requirement *contiguous*. Going back to the previous example, the requirement holds if we assume for instance (in addition to the previous properties) that the $i_j$'s are pairwise distinct (more generally it is clear that the requirement always holds if constants with distinct indices are interpreted by distinct terms). Some simple examples of application of our work are presented in Section 5, in which we show that our procedure can be employed to check properties of simple programs with loops, where indexed constants $a_i$ are used to denote the value of some variables $a$ at iteration $i$. Section 6 briefly concludes the paper and provides some lines of future work. Due to space restrictions, some of the proofs are omitted.

## 2 Schemata of Formulæ

### 2.1 Syntax

The set of *sort terms* $\mathtt{T}$ is constructed inductively over a finite set of *base sorts* $\mathtt{B}$ using the operator $\rightarrow$: if $\mathtt{s}$ and $\mathtt{s}'$ are two sort terms in $\mathtt{T}$ then the sort term $\mathtt{s} \rightarrow \mathtt{s}'$ denotes the sort of the arrays (or functions) mapping elements of sort $\mathtt{s}$ to elements of sort $\mathtt{s}'$. Let $\mathtt{nat}$ be a special sort symbol, not occurring in $\mathtt{B}$ or $\mathtt{T}$. Let $\Sigma$ be a set of symbols (denoting either constants of a base sort or arrays). Each symbol is $\Sigma$ is associated with a unique *profile* that is either a sort term or of the form $\mathtt{nat} \rightarrow \mathtt{s}$ where $\mathtt{s}$ is a sort term. A symbol is called *indexed* iff its profile is of the later form. The set of indexed and non-indexed symbols are denoted by $\Sigma_{\mathtt{nat}}$ and $\Sigma_{\perp}$, respectively.

Let $\mathcal{V}$ be a set of *arithmetic constants*. An *arithmetic expression* of parameter $\mathtt{n} \in \mathcal{V}$ is an expression of the form $k$ or $\mathtt{n} + k$, where $\mathtt{n} \in \mathcal{V}$ and $k \in \mathbb{N}$. As usual, $\mathtt{n} + 0$ is simply written $\mathtt{n}$. The set of arithmetic expressions of parameter $\mathtt{n}$ is denoted by $\mathcal{T}_{\mathbb{N}}(\mathtt{n})$. If $\alpha, \beta \in \mathcal{T}_{\mathbb{N}}(\mathtt{n})$ then we write $\alpha < \beta$ iff the previous relation holds regardless of the value of the parameter $\mathtt{n}$, i.e., iff one of the following conditions hold: $\alpha$ and $\beta$ are natural numbers and $\alpha < \beta$, $\alpha$ and $\beta$ are of the form $\mathtt{n} + k$ and $\mathtt{n} + l$ respectively and $k < l$, or $\alpha$ is a natural number, $\beta$ is of the form $\mathtt{n} + k$ and $\alpha < k$.

The set of terms $\mathcal{T}(\mathtt{s},\mathtt{n})$ of sort $\mathtt{s} \in \mathtt{T}$ and of parameter $\mathtt{n} \in \mathcal{V}$ is built inductively as follows.

- All non-indexed symbols $u \in \Sigma_\perp$ of profile $\mathtt{s}$ are terms in $\mathcal{T}(\mathtt{s},\mathtt{n})$.
- For all $u \in \Sigma_{\mathtt{nat}}$ of profile $\mathtt{nat} \to \mathtt{s}$ and for all $\alpha \in \mathcal{T}_{\mathbb{N}}(\mathtt{n})$, we have $u_\alpha \in \mathcal{T}(\mathtt{s},\mathtt{n})$.
- For all sort terms $\mathtt{s}'$, for all $(t,u) \in \mathcal{T}(\mathtt{s}' \to \mathtt{s},\mathtt{n}) \times \mathcal{T}(\mathtt{s}',\mathtt{n})$, $\mathit{select}(t,u) \in \mathcal{T}(\mathtt{s},\mathtt{n})$.
- If $\mathtt{s}$ is of the form $\mathtt{s}' \to \mathtt{s}''$, then for all $(t,u,v) \in \mathcal{T}(\mathtt{s},\mathtt{n}) \times \mathcal{T}(\mathtt{s}',\mathtt{n}) \times \mathcal{T}(\mathtt{s}'',\mathtt{n})$, $\mathit{store}(t,u,v) \in \mathcal{T}(\mathtt{s},\mathtt{n})$.

The set $\mathcal{T}(\mathtt{n})$ denotes the entire set of terms $\mathcal{T}(\mathtt{n}) \stackrel{\text{def}}{=} \bigcup_{\mathtt{s}\in\mathtt{T}} \mathcal{T}(\mathtt{s},\mathtt{n})$. If $u$ is a term of sort $\mathtt{s} \to \mathtt{s}'$, then $\mathtt{s}$ is the *domain* of $u$ and $\mathtt{s}'$ is its *range*. For every arithmetic expression $\alpha$, we denote by $\Sigma[\alpha]$ the set of terms that are either element of $\Sigma_\perp$ or of the form $u_\alpha$, with $u \in \Sigma_{\mathtt{nat}}$.

The *term depth* $\delta(u)$ and the *index depth* $\iota(u)$ of a term $u$ are inductively defined as follows: $\delta(u_\alpha) \stackrel{\text{def}}{=} \delta(u) \stackrel{\text{def}}{=} 1$ if $u \in \Sigma_{\mathtt{nat}} \cup \Sigma_\perp$, $\delta(\mathit{select}(t,u)) \stackrel{\text{def}}{=} \max(\delta(t),\delta(u)) + 1$, $\delta(\mathit{store}(t,u,v) \stackrel{\text{def}}{=} \max(\delta(t),\delta(u),\delta(v)) + 1$, $\iota(u) \stackrel{\text{def}}{=} 0$ if $u \in \Sigma_\perp$, $\iota(u_{\mathtt{n}+k}) \stackrel{\text{def}}{=} \iota(u_k) \stackrel{\text{def}}{=} k+1$ if $k \in \mathbb{N}$, $\iota(\mathit{select}(t,u)) \stackrel{\text{def}}{=} \max(\iota(t),\delta(u))$, $\iota(\mathit{store}(t,u,v) \stackrel{\text{def}}{=} \max(\iota(t),\delta(u),\delta(v))$, For instance, if $s \in \Sigma_\perp$ then $\mathit{select}(t_{\mathtt{n}+1}, \mathit{select}(s,u_{\mathtt{n}}))$ is of term depth 3 and of index depth 2.

The set $\mathcal{F}(\mathtt{n})$ of *A-formulæ* of parameter $\mathtt{n}$ is inductively constructed as follows (note that we assume that all formulæ are in negation normal form).

- $\perp, \top \in \mathcal{F}(\mathtt{n})$.
- If $u,v \in \mathcal{T}(\mathtt{s},\mathtt{n})$ for some sort $\mathtt{s} \in \mathtt{T}$, then $u \simeq v$ and $u \not\simeq v$ are in $\mathcal{F}(\mathtt{n})$.
- If $\phi,\psi \in \mathcal{F}(\mathtt{n})$ then $\phi \wedge \psi, \phi \vee \psi \in \mathcal{F}(\mathtt{n})$.
- If $i \in \mathcal{V}$, $\phi \in \mathcal{F}(\mathtt{i})$, and $k,l \in \mathbb{N}$ then $\bigvee_{\mathtt{i}=k}^{\mathtt{n}+l} \phi$ and $\bigwedge_{\mathtt{i}=k}^{\mathtt{n}+l} \phi$ are in $\mathcal{F}(\mathtt{n})$.

An expression $\alpha$ is an *index* of a formulæ $\phi$ if it occurs in a term of the form $u_\alpha$ of $\phi$. The term depth and index depth of an $A$-formula $\phi$ is the maximal term (resp. index) depth of the terms occurring in $\phi$. The *propositional depth* $\pi(\phi)$ is defined as follows. $\pi(\perp) \stackrel{\text{def}}{=} \pi(\top) \stackrel{\text{def}}{=} \pi(u \simeq v) \stackrel{\text{def}}{=} \pi(u \not\simeq v) \stackrel{\text{def}}{=} 1$, $\pi(\phi \star \psi) \stackrel{\text{def}}{=} 1 + \max(\pi(\phi),\pi(\psi))$ and $\pi(\Pi_{\mathtt{i}=k}^{\mathtt{n}+l}\phi) = 1 + \pi(\phi)$ (with $\star \in \{\vee,\wedge\}$, $\Pi \in \{\bigvee,\bigwedge\}$).

For every expression $e$, we denote by $e\{\alpha \mapsto \beta\}$ the expression obtained from $e$ by replacing every occurrence of $\alpha$ by $\beta$.

### 2.2 Semantics

The semantics of $A$-formulæ is defined in a straightforward way. An *interpretation* $\mathcal{I}$ is a function mapping every expression $e$ in $\mathtt{T} \cup \mathcal{V} \cup \{\mathit{store}, \mathit{select}\} \cup \bigcup_{\mathtt{n}\in\mathcal{V}}(\mathcal{T}_{\mathbb{N}}(\mathtt{n}) \cup \mathcal{T}(\mathtt{n}) \cup \mathcal{F}(\mathtt{n}))$ to an object $[e]^{\mathcal{I}}$ such that:

- For every sort $\mathtt{s} \in \mathtt{T}$, $[\mathtt{s}]^{\mathcal{I}}$ is a non-empty set of elements (the *domain* of $\mathtt{s}$).
- For every $\mathtt{n} \in \mathcal{V}$, $[\mathtt{n}]^{\mathcal{I}}$ is a natural number.
- For every symbol $u$ of profile $\mathtt{s} \in \mathtt{T}$, $[u]^{\mathcal{I}}$ is an element of $[\mathtt{s}]^{\mathcal{I}}$.
- For every constant $u$ of profile $\mathtt{nat} \to \mathtt{s}$, $[u]^{\mathcal{I}}$ is a function from $\mathbb{N}$ to $[\mathtt{s}]^{\mathcal{I}}$.
- $[k]^{\mathcal{I}} \stackrel{\text{def}}{=} k$ if $k \in \mathbb{N}$, and $[\mathtt{n}+k]^{\mathcal{I}} \stackrel{\text{def}}{=} [\mathtt{n}]^{\mathcal{I}} + k$ if $\mathtt{n} \in \mathcal{V}, k \in \mathbb{N}$.
- $[\mathit{store}]^{\mathcal{I}}$ is a function that maps every triple $(e,e',f) \in ([\mathtt{s} \to \mathtt{s}']^{\mathcal{I}}, [\mathtt{s}]^{\mathcal{I}}, [\mathtt{s}']^{\mathcal{I}})$ to an element of $[\mathtt{s} \to \mathtt{s}']^{\mathcal{I}}$.
- $[\mathit{select}]^{\mathcal{I}}$ is a function that maps every pair $(e,e') \in ([\mathtt{s} \to \mathtt{s}']^{\mathcal{I}}, [\mathtt{s}]^{\mathcal{I}})$ to an element of $[\mathtt{s}']^{\mathcal{I}}$.

- $\mathcal{I}$ satisfies the axioms of the theory of arrays, i.e., for every $t, u, v, w \in ([\mathtt{s} \to \mathtt{s}']^{\mathcal{I}}, [\mathtt{s}]^{\mathcal{I}}, [\mathtt{s}']^{\mathcal{I}}, [\mathtt{s}]^{\mathcal{I}})$, we have: $[\mathit{select}]^{\mathcal{I}}([\mathit{store}]^{\mathcal{I}}(t, u, v), u) = v$ and $[\mathit{select}]^{\mathcal{I}}([\mathit{store}]^{\mathcal{I}}(t, u, v), w) = [\mathit{select}]^{\mathcal{I}}(t, w)$ if $u \neq w$.
- $[u_\alpha]^{\mathcal{I}} \stackrel{\text{def}}{=} [u]^{\mathcal{I}}([\alpha]^{\mathcal{I}})$, and for every term $f(t_1, \ldots, t_n)$, where $f \in \{\mathit{select}, \mathit{store}\}$, $n = 2, 3$, we have $[f(t_1, \ldots, t_n)]^{\mathcal{I}} \stackrel{\text{def}}{=} [f]^{\mathcal{I}}([t_1]^{\mathcal{I}}, \ldots, [t_n]^{\mathcal{I}})$.
- For every $A$-formula $\phi$, $[\phi]^{\mathcal{I}}$ is a truth value in $\{\mathtt{true}, \mathtt{false}\}$, inductively defined as follows.
  - $[\top]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$, $[\bot]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{false}$, $[u \simeq v]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff $[u]^{\mathcal{I}} = [v]^{\mathcal{I}}$, $[u \not\simeq v]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff $[u]^{\mathcal{I}} \neq [v]^{\mathcal{I}}$, $[\phi \vee \psi]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff $[\phi]^{\mathcal{I}} = \mathtt{true}$ or $[\psi]^{\mathcal{I}} = \mathtt{true}$, and $[\phi \wedge \psi]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff $[\phi]^{\mathcal{I}} = \mathtt{true}$ and $[\psi]^{\mathcal{I}} = \mathtt{false}$.
  - $[\bigvee_{\mathtt{i}=k}^{\mathtt{n}+l} \phi]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff there exists a natural number $m \in [k, [\mathtt{n}]^{\mathcal{I}} + l]$ such that $[\phi]^{\mathcal{I}[\mathtt{i} \leftarrow m]} = \mathtt{true}$, where $\mathcal{I}[\mathtt{i} \leftarrow m]$ denotes the interpretation coinciding with $\mathcal{I}$ except on $\mathtt{i}$, for which we have $[\mathtt{i}]^{\mathcal{I}[\mathtt{i} \leftarrow m]} \stackrel{\text{def}}{=} m$.
  - $[\bigwedge_{\mathtt{i}=k}^{\mathtt{n}+l} \phi]^{\mathcal{I}} \stackrel{\text{def}}{=} \mathtt{true}$ iff for all natural numbers $m \in [k, [\mathtt{n}]^{\mathcal{I}} + l]$ we have $[\phi]^{\mathcal{I}[\mathtt{i} \leftarrow m]} = \mathtt{true}$.

An interpretation $\mathcal{I}$ is a *model* of an $A$-formula $\phi$ (resp. set of $A$-formulæ $S$) if $[\phi]^{\mathcal{I}} = \mathtt{true}$ (resp. if $\forall \phi \in S, [\phi]^{\mathcal{I}} = \mathtt{true}$). This is written $\mathcal{I} \models \phi$ (resp. $\mathcal{I} \models S$). An $A$-formula or a set of $A$-formulæ having a model is *satisfiable*.

## 3 Undecidability Results

The following proposition is an immediate consequence of the previous definition.

**Proposition 1.** *The satisfiability problem is semi-decidable for A-formulæ.*

*Proof.* If the value of the parameter is fixed then all iterated formulæ can be replaced by standard disjunctions and conjunctions. The obtained $A$-formula is then equivalent to a ground formula interpreted on the theory of arrays, for which satisfiability can be tested by usual procedures. Thus it suffices to enumerate all the possible values of the parameter until we get one for which the considered formula is satisfiable.

The next theorem shows that the problem is not decidable.

**Theorem 2.** *The satisfiability problem is undecidable for A-formulæ, even if the sets of range and domain sorts are disjoint*[4].

*Proof.* We provide a brief informal overview of the proof. It is based on an encoding of the Post correspondence problem. We thus consider two sequences of words $w^{1,1}, \ldots, w^{1,n}$ and $w^{2,1}, \ldots, w^{2,n}$ and we construct an $A$-formula $\phi$ that is satisfiable iff there exists a sequence $I_1, \ldots, I_k$ such that $w^{1,I_1}. \cdots .w^{1,I_k} = w^{2,I_1}. \cdots .w^{2,I_k}$ (where "." denotes word concatenation). To this purpose, we use two indexed arrays $W^1$ and $W^2$ denoting the words

[4] The case in which the domain and range of the arrays are allowed to be identical follows immediately from the results in [5] about the undecidability of schemata of equational formulæ in the empty theory.

$w^{1,I_1}.\cdots.w^{1,I_k}$ and $w^{2,I_1}.\cdots.w^{2,I_k}$ respectively. More precisely $W^i$ is defined on the domain $\{a_1,\ldots,a_{\mathtt{n}}\}$ and $\mathit{select}(W^i,a_j)$ contains the character $j$ of the word $w^{i,I_1}.\cdots.w^{i,I_k}$, defined as a pair $(k,l)\in[1,n]\times[1,|w^{i,k}|]$, meaning that this character is the $l$-th element of the word $w^{i,k}$. It is straightforward to state as an $A$-formula that $W^i$ indeed represents a word of the form $w^{i,I^i_1}.\cdots.w^{i,I^i_k}$, for some sequence $I^i_1,\ldots,I^i_k$: it suffices to check that the successor of any element $(k,l)$ is either of the form $(k,l+1)$ (if $l$ is not the last character in $w^{i,k}$) or of the form $(k',1)$, for some $k'\in[1,n]$ (if $l$ is the last element of $w^{i,k}$). Similarly, it is easy to check that the words denoted by $W^1$ and $W^2$ are identical. The difficult point is to check that the two sequences $I^i_1,\ldots,I^i_k$ $(i=1,2)$ are identical. For expressing this property, we take advantage of the expressive power of the theory of arrays: We introduce two indexed constants $b^i$ such that $(b^i_1,\ldots,b^i_{\mathtt{n}})$ is *exactly* the sequence of cells corresponding to the beginning of a new word in $W^i$. To define these sequences, we construct two arrays $B^i$ $(i=1,2)$ containing `true` exactly at the cells corresponding to the beginning of a new word in $W^i$ (which can be expressed by stating that for every $j=1,\ldots,\mathtt{n}$, $\mathit{select}(B^i,aj)$ is `true` iff $\mathit{select}(W^i,a_j)$ is of the form $(k,1)$). Then we check that this array $B^i$ is identical to the array obtained by inserting `true` at every cell $b^i_j$ $(1\le j\le\mathtt{n})$, inside an array containing initially `false` at each cell $a_j$ $(1\le j\le\mathtt{n})$. This guarantees that $\{b^i_1,\ldots,b^i_{\mathtt{n}}\}$ is indeed the set of cells corresponding to the start of a new word. The most subtle point is to ensure that the order of these cells is the intended one, i.e. that the first new word after $b_j$ indeed starts at $b_{j+1}$ and not, say, at $b_{j+2}$. This is done by adding axioms "copying" the value of the next element corresponding to the start of a word all along the array $W^i$ in order to check that two contiguous words really start at some cells $b^i_j$ and $b^i_{j+1}$. Using these sequences $b^i_j$, it is straightforward to check that the sequences $I^i_j$ are identical, by verifying that $\mathit{select}(W^1,b_j)$ is equal to $\mathit{select}(W^2,b_j)$, for every $j=1,\ldots,\mathtt{n}$.

## 4 Decidability Results

The results in Section 3 show that the satisfiability problem is undecidable for $A$-formulæ. In such a situation, the standard approach consists in focusing on particular syntactic fragments so that decidability and/or complexity results can be obtained. In the present paper, instead of restricting the class of formulæ, we prefer to restrict the class of *interpretations* by imposing additional conditions on the considered structures. These conditions are formalized in Section 4.2.

### 4.1 Simplifying the Syntax

For technical convenience, we shall assume from now that all the formulæ satisfy some additional syntactic restrictions:

1. The term depth is bounded by 2; and every literal is either an equation or a disequation between constants or indexed constants or of the form $f(\boldsymbol{u})\simeq v$, with $f\in\{\mathit{select},\mathit{store}\}$ and $\boldsymbol{u},v$ are constants or indexed constants.
2. The only arithmetic expressions occurring in the considered formula are 0, $\mathtt{n}$ or $\mathtt{n}+1$, where $\mathtt{n}\in\mathcal{V}$.
3. The formula contains no nested iterations, i.e., for every subformula of the form $\bigvee_{\mathtt{i}=0}^{\mathtt{n}+l}\phi$ or $\bigwedge_{\mathtt{i}=k}^{\mathtt{n}+l}\phi$, $\phi$ contains no iterated formula.

It is easy to check that these conditions do not reduce the expressive power of the language. First, any arithmetic expression distinct from the parameter $\mathtt{n}$ occurring as the upper bound of an iteration can be removed by unfolding the considered iteration, e.g., $\bigvee_{\mathtt{i}=0}^{\mathtt{n}+2} \phi$ is equivalent to $\bigvee_{\mathtt{i}=0}^{\mathtt{n}} \phi \vee \phi\{\mathtt{i} \mapsto \mathtt{n}+1\} \vee \phi\{\mathtt{i} \mapsto \mathtt{n}+2\}$. It is also easy to get rid of an iteration whose lower bound is a number $k \neq 0$: this can be done by introducing a new atom $p_{\mathtt{i}}^k$ stating that $\mathtt{i}$ is strictly greater than $k$, and defined by the axioms: $\neg p_0^k \wedge \ldots \wedge \neg p_{k-1}^k \wedge p_k^k \wedge \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \neg p_{\mathtt{i}}^k \vee p_{\mathtt{i}+1}^k$. Then an iteration $\bigvee_{\mathtt{i}=k}^{\mathtt{n}} \phi$ (resp. $\bigwedge_{\mathtt{i}=k}^{\mathtt{n}} \phi$) can be written $\bigvee_{\mathtt{i}=0}^{\mathtt{n}} p_{\mathtt{i}}^k \wedge \phi$ (resp. $\bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \neg p_{\mathtt{i}}^k \vee \phi$). Condition 3 is also easy to enforce, because any iteration $\Pi_{\mathtt{j}=0}^{\mathtt{i}} \psi$ occurring inside an iteration can be replaced by a new atom $p_{\mathtt{i}}$, while adding the axioms: $\neg p_0 \vee \psi\{\mathtt{i} \mapsto 0\}$ and $\bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \neg p_{\mathtt{i}+1} \vee (\psi\{\mathtt{j} \mapsto \mathtt{i}+1\} \star p_{\mathtt{i}})$ (with $(\Pi, \star) \in \{(\bigvee, \vee), (\bigwedge, \wedge)\}$). Afterward, Condition 1 can be enforced by applying the standard *flattening* operation (see for instance [6]), i.e., every complex subterm $u$ of parameter $\mathtt{n}$ can be (repeatedly) replaced by a new constant $v_{\mathtt{n}}$, while adding the axiom $\bigwedge_{\mathtt{i}=0}^{\mathtt{n}+1} v_{\mathtt{i}} \simeq u\{\mathtt{n} \mapsto \mathtt{i}\}$. This ensures that the term depth is at most 2 and that the symbols *select* and *store* only occur in positive literals (and occur only once in every literal). Condition 2 is ensured in a similar way, by (recursively) replacing any constant $u_{\mathtt{n}+k}$ with $k > 1$ by a constant $u'_{\mathtt{n}+k-1}$, while adding the axiom $\bigwedge_{\mathtt{i}=0}^{\mathtt{n}+1} u'_{\mathtt{i}} \simeq u_{\mathtt{i}+1}$. Due to space restrictions, the formal description of these transformations is omitted, see [3, 4] for more details.

### 4.2 Restricting the Class of Interpretations

We first define a property of the sort terms which depends only on the syntactic form of the formula. Intuitively, a sort $\mathtt{s}$ is called *non-cyclic* if: (i) no storing operation is performed on arrays of domain $\mathtt{s}$; and (ii) no array of sort $\mathtt{s} \to \mathtt{s}$ occurs in the signature, even with array composition (by composing two arrays of sort $\mathtt{s} \to \mathtt{s}'$ and $\mathtt{s}' \to \mathtt{s}$ one gets an array of sort $\mathtt{s} \to \mathtt{s}$). The condition is formalized as follows.

**Definition 3.** *Let $\prec$ be a (fixed) ordering among sort terms, such that $\mathtt{s}, \mathtt{s}' \prec \mathtt{s} \to \mathtt{s}'$, for all sort terms $\mathtt{s}, \mathtt{s}'$. A sort $\mathtt{s}$ is* non-cyclic *in an A-formula $\phi$ if the two following conditions hold.*
- *$\phi$ contains no term of the form $store(u, v, w)$ where $v$ is of sort $\mathtt{s}$.*
- *$\phi$ contains no term of sort $\mathtt{s}' \to \mathtt{s}$ with $\mathtt{s}' \not\prec \mathtt{s}$.*

The second condition of Definition 3 is related to the notion of a *stratified signature* in [1]: if the formula at hand contains no occurrence of *store* and if the signature is stratified then every sort is non-cyclic.

The conditions of Definition 3 are rather restrictive, thus instead of assuming that every sort is non-cyclic, we prefer to allow cyclic sorts and to add further restrictions on their interpretations. These restrictions are formalized in the following definition.

**Definition 4.** *A sort $\mathtt{s}$ is* contiguous *in an interpretation $\mathcal{I}$ iff for every pair of constant symbols $u, v$ of profile $\mathtt{nat} \to \mathtt{s}$ and for every $k, l \in \mathbb{N}$ such that $k < l$ and $\mathcal{I} \models u_k \simeq v_l$, one of the following conditions holds:*

- *There exists a constant $w$ of sort* $\mathtt{s}$ *such that* $\mathcal{I} \models u_k \simeq w \wedge v_l \simeq w$.
- *There exists* $l-k+1$ *constants* $w^1, \ldots, w^{l-k+1}$ *such that* $w^1 = u$, $w^{l-k+1} = v$ *and* $\forall i \in [1, l-k], \mathcal{I} \models w^i_{k+i-1} \simeq w^{i+1}_{k+i}$.

For instance the condition of Definition 4 holds if $\mathtt{s}$ is finite, because in this case one may associate a non-indexed constant symbol $w$ with each element of the domain of $\mathtt{s}$; or if the implication $k+1 < l \Rightarrow u_k \not\simeq v_l$ holds in $\mathcal{I}$, for all constant symbols $u, v \in \Sigma_{\mathtt{nat}}$ and for all $k, l \in \mathbb{N}$. It also holds if the interpretation of the constants of sort $\mathtt{nat} \to \mathtt{s}$ is monotonic, i.e., if there exists an ordering $\leq$ such that $k < l \Rightarrow a_k \leq b_l$ holds (it suffices to add for each sort $\mathtt{s}$ a new constant $u$ such that $u_k$ is interpreted as the maximal term of sort $\mathtt{s}$ of the form $v_{k-1}$).

**Definition 5.** *An interpretation $\mathcal{I}$ is* contiguous *if every sort is contiguous in $\mathcal{I}$. It is* quasi-contiguous *iff every cyclic sort is contiguous.*

The following lemma shows that quasi-contiguous and contiguous interpretations are equivalent for satisfiability testing:

**Lemma 6.** *An A-formula has a quasi-contiguous model iff it has a contiguous model (up to the addition of a finite set of new constant symbols and axioms).*

In the next section, we shall therefore assume that all the considered interpretations are contiguous.

### 4.3 Proof Procedure

We devise a tableau-based proof procedure deciding the satisfiability of $A$-formulæ in contiguous (or quasi-contiguous) interpretations. We first briefly review some basic terminology. Tableaux are viewed as trees labeled by sets of $A$-formulæ. A *branch* is a path from the root to a leaf. An interpretation $\mathcal{I}$ *validates* a tableau if there exists a leaf labeled by some set $S$ such that $\mathcal{I} \models S$. A branch is *closed* if it contains $\bot$. The procedure is defined by rules of the form $\dfrac{H_1, \ldots, H_n}{\mathcal{C}_1 \mid \ldots \mid \mathcal{C}_m}$, meaning that a leaf labeled by a set $S$ can be expanded by $m$ new children, labeled by $S \cup \mathcal{C}_1, \ldots, S \cup \mathcal{C}_m$ respectively, if $S$ contains $H_1, \ldots, H_n$ (up to an instantiation of the meta-variables). The rule only applies if the branch is open and if there is no $i \in [1, m]$ such that $S$ contains all the formulæ in $\mathcal{C}_i$.

**Overview of the Proof Procedure** The proof procedure can be informally described as follows.

- First, the usual decomposition rules of propositional logic are applied, together with additional transitivity and paramodulation rules handling the properties of the equality predicate. We also consider generalized decomposition rules unfolding iterated formulæ (e.g., to infer $\phi\{\mathtt{i} \mapsto \mathtt{n}+1\}$ from $\bigwedge_{\mathtt{i}=0}^{\mathtt{n}+1} \phi$). In order to handle store operations, we introduce new atoms of the form $t \simeq_E s$, meaning that $t$ and $s$ coincide on every element, except on those occurring in the set $E$.
- Then, we apply an inductive rule performing a case analysis on the parameter $\mathtt{n}$, considering separately the two cases $\mathtt{n} = 0$ and $\mathtt{n} > 0$. The rule recursively replaces $\mathtt{n}$ by either $0$ or $\mathtt{n}+1$, and thus lazily instantiates the parameter with natural numbers, which enables further applications of the decomposition

rules. Of course, the addition of the induction rule makes the calculus non-terminating, since $\mathtt{n}$ can be instantiated indefinitely.

- To avoid non-termination, a loop detection mechanism is added to prune infinite branches. To get rid of such branches, we have to ensure that the depth of the formulæ occurring in the tableau is bounded, so that every infinite branch contains a cycle, i.e., two nodes labeled by the same set of formulæ. It is easy to see that the term depth and propositional depth of the formula cannot increase, thus we only have to ensure that its index depth is bounded. To this aim, we introduce new (satisfiability-preserving) rules allowing to get rid of all formulæ containing an expression $\mathtt{n}+2$. This ensures that only terms indexed by 0, $\mathtt{n}$ or $\mathtt{n}+1$ will remain in the formula before the inductive rule is applied, hence the index depth will never be greater than 3. When trying to eliminate formulæ containing an occurrence of $\mathtt{n}+2$, it turns out that it is sometimes necessary to infer additional properties of the remaining symbols. For instance, if the considered branch contains a formula $\mathit{select}(t, u_{\mathtt{n}+2}) \not\simeq \mathit{select}(s, u_{\mathtt{n}+2})$ then we have to express the fact that $t$ and $s$ disagree at some element $u_{\mathtt{n}+2}$. But we have to express this property without explicitly referring to the term $u_{\mathtt{n}+2}$ (since our goal is to get rid of all such terms), and of course without introducing new symbols (which would prevent termination). This cannot be done in the initial language, thus we need to enrich the syntax by new predicates allowing to express such properties in a convenient way. Of course, we also need to add expansion rules encoding the axioms defining these predicates.

**Enriching the Syntax** We therefore enrich the syntax of the language by new constructions. We first consider *set expressions*, denoting sets of individuals, and built using the constructor $\cup$ over a set of basic sets $\emptyset$, $\{u\}$ and $\theta(\alpha)$, where $u$ is a constant or indexed constant and $\alpha$ an arithmetic expression. The sets $\emptyset$ and $\{u\}$ and the constructor $\cup$ are interpreted in a natural way, and $\theta(\alpha)$ is interpreted in any interpretation $\mathcal{I}$ as the set of terms that are distinct from all expressions in $\mathcal{T}^{\mathcal{I}}_{|<\alpha}$, with $\mathcal{T}^{\mathcal{I}}_{|<\alpha} \stackrel{\text{def}}{=} [\Sigma_\perp]^{\mathcal{I}} \cup \{[u_k]^{\mathcal{I}} \mid u \in \Sigma_{\mathtt{nat}}, k < [\alpha]^{\mathcal{I}}\}$. Intuitively, $\mathcal{T}^{\mathcal{I}}_{|<\alpha}$ denotes the set of named elements whose index is strictly lower than $\alpha$. We also consider a predicate symbol $\in$ interpreted as usual as set membership, and a symbol $\simeq_E$ stating that two arrays agree on all elements not occurring in the set denoted by $E$. For instance, $t \simeq_{\{u,v\}} s$ states that $t$ and $s$ coincide at every element distinct from $u$ and $v$ and $t \simeq_{\theta(\alpha)} s$ states that $t$ and $s$ coincide on every element in $\mathcal{T}^{\mathcal{I}}_{|<\alpha}$. Furthermore, we introduce a predicate $\nsim_\alpha$ (with $\alpha \in \mathcal{T}_{\mathbb{N}}(\mathtt{n}) \cup \{-1\}$), such that $\mathcal{I} \models t \nsim_\alpha s$ iff there exists an element $e \in \theta(\alpha+1)$ (in the domain of $t, s$) such that the interpretations of $t$ and $s$ disagree at $e$. The advantage of the predicates $\simeq_E$ and $\nsim_\alpha$ is that they allow us to express properties of arrays without having to refer explicitly to the symbols denoting elements of the domain of these arrays. In particular, the symbol $\simeq_E$ can be used to encode store operations. More precisely, every atom of the form $\mathit{store}(t, u, v) \simeq s$ can be replaced by the conjunction $\mathit{select}(s, u) \simeq v \wedge t \simeq_{\{u\}} s$, stating the fact that $s$ contains $v$ at $u$ and coincides with $t$ at all other elements. Thanks to this transformation (which obviously preserves satisfiability) we can assume that the considered $A$-formula contains no instance of *store*. Finally, we add new constants

denoting all terms $\mathit{select}(t,u) \in \mathcal{T}(\mathtt{n})$, with appropriate axioms, so that the index of the constant denoting a term $\mathit{select}(t,u)$ is the maximal index in $t,u$ (note that these special constants do not themselves occur in $\mathcal{T}(\mathtt{n})$).

We are now in position to formally describe the inference rules defining the proof procedure.

**Propositional Decomposition Rules** The rules ($\wedge$-D), ($\vee$-D) and ($\bot$) are the usual decomposition rules of propositional tableaux. Other decomposition rules (similar to those in [2]) are added to handle iterated connectives.

$$(\wedge\text{-D})\quad \frac{\phi \wedge \psi}{\phi, \psi} \qquad (\vee\text{-D})\quad \frac{\phi \vee \psi}{\phi \mid \psi} \qquad (\bot)\quad \frac{\phi, \neg\phi}{\bot}$$

$$(\bigwedge\text{-D})\quad \frac{\bigvee_{\mathtt{i}=0}^{\mathtt{n}+1} \phi}{\bigvee_{\mathtt{i}=0}^{\mathtt{n}} \phi \mid \phi\{\mathtt{i} \mapsto \mathtt{n}+1\}} \qquad (\bigvee\text{-D})\quad \frac{\bigwedge_{\mathtt{i}=0}^{\mathtt{n}+1} \phi}{\phi\{\mathtt{i} \mapsto \mathtt{n}+1\}, \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \phi}$$

**Equality Rules** The next rules encode the usual properties of the equality predicate. The rule (S) encodes the substitutivity property, ($\simeq_E$-R$_1$) and ($\simeq_E$-R$_2$) encode the properties of the predicate $\simeq_E$, whereas (P) allows one to replace a term by an equal constant or indexed constant. Finally, (Ref) is a closure rule encoding reflexivity and (C) encodes the fact that interpretations are contiguous. To this purpose, it suffices to check that all constants of the form $u_{\beta+2}$ occur either in $\Sigma[\beta+1]$ or in $\theta(\beta+2)$.

$$(\text{S})\quad \frac{\mathit{select}(t,u) \simeq w, \mathit{select}(s,v) \simeq w'}{t \not\simeq s \mid u \not\simeq v \mid w \simeq w'} \qquad (\text{P})\quad \frac{u \simeq u', \phi[u]}{\phi[u']} \quad \text{if } \delta(u') = 1$$

and either $u' \in T(\mathtt{n})$ or $u$ does not occur in the scope of *select* or $\simeq_E$ in $\phi$.

$$(\simeq_E\text{-R}_2)\quad \frac{u \simeq_E v, u' \simeq_{E'} w}{u \not\simeq u' \mid v \simeq_{E \cup E'} w} \qquad (\simeq_E\text{-R}_1)\quad \frac{t \simeq_E s, \mathit{select}(t',u) \simeq v}{u \in E \mid t \not\simeq t' \mid \mathit{select}(s,u) \simeq v}$$

$$(\text{Ref})\quad \frac{u \not\simeq u}{\bot} \qquad (\text{C})\quad \frac{}{u_{\beta+2} \in E \mid u_{\beta+2} \in \theta(\beta+2)}$$

where $E$ is the set of terms in $\Sigma[\beta+1]$ that are of the same sort as $u_{\beta+2}$.

**$\sim$-Rules** The two following rules handle the predicate $\nsim_\alpha$. The first one encodes a form of transitivity on $\nsim_\alpha$ and $\simeq_E$. Indeed, if $u \nsim_\alpha v$ and $w \simeq_E u$ hold, for some set of elements $E$ whose indices are lower or equal to $\alpha$, then we necessarily have $w \nsim_\alpha v$. Indeed, by definition, $u \nsim_\alpha v$ holds if $\mathit{select}(u,e) \neq \mathit{select}(v,e)$ for some element $e$ distinct from every non-indexed term and from every term indexed by a natural number that is lower or equal to $\alpha$. But such an element $e$ cannot occur in $E$, thus $w$ coincides with $u$ at $e$, and $w \nsim_\alpha v$ necessarily holds.

$$(\sim\text{-E})\quad \frac{u \nsim_\alpha v, w \simeq_E u'}{u \not\simeq u' \mid w \nsim_\alpha v} \quad \text{if } \alpha \text{ is maximal in the node, } E \text{ is a finite set.}$$

The next rule allows one to derive expressions of the form $t \nsim_{\alpha-1} s$. Such an expression is derivable if there exists an element $e$ such that $select(t, e) \neq select(s, e)$ and if this element is distinct from every non-indexed term and from every term indexed by a natural number that is strictly lower than $\alpha$. Thanks to the fact that the interpretations are assumed to be contiguous, we only have to test that $e$ is distinct from all constants of the same sort of $e$ that are non-indexed or indexed by $\alpha - 1$.

$$(\sim\text{-I}) \quad \frac{select(t, u_\alpha) \simeq v, select(s, w) \simeq w'}{u_\alpha \in E \quad | \quad u_\alpha \not\simeq w \quad | \quad v \simeq w' \quad | \quad t \nsim_{\alpha-1} s, u_\alpha \simeq w, v \not\simeq w'}$$

where $E$ is the set of terms in $\Sigma[\alpha - 1]$ that are of the same sort as $u_\alpha$.

The next rule in this section allows one to introduce expressions of the form $\theta(\alpha)$. This is done by replacing a constant $u$ occurring in a set $E$ by $\theta(\alpha)$, if $u$ occurs in $\theta(\alpha)$. This last condition is tested in the same way as in the previous rule, by verifying that $u_\alpha$ is distinct from all constants that are non-indexed or indexed by $\alpha - 1$.

$$\theta\text{-}I \quad \frac{t \simeq_{E \cup \{u_\alpha\}} s}{u_\alpha \in E' \quad | \quad t \simeq_{E \cup \theta(\alpha)} s}$$

where $E'$ is the set of terms in $\Sigma[\alpha - 1]$ that are of the same sort as $u_\alpha$.

The last rules in this subsection express straightforward properties of $\nsim_\alpha$:

$$(\nsim\text{-I}) \quad \frac{t \nsim_{\alpha+1} s}{t \nsim_\alpha s} \qquad (\sim\text{-}\bot) \quad \frac{t \nsim_\alpha t}{\bot}$$

$$(\sim\text{-D}) \quad \frac{}{t \nsim_\beta s \quad | \quad t \sim_\beta s} \qquad (\sim\text{-}T) \quad \frac{t \nsim_\beta s}{s' \nsim_\beta t \quad | \quad s' \nsim_\beta s}$$

if $\beta$ is the maximal index in the node, $t, s, s'$ are non-indexed constants or constants indexed by an expression lower or equal to $\beta$.

**$\in$-Rules** The following rules encode the definition of $\in$ and of the sets $\theta(\alpha)$.

$$\in\text{-}E_1 \quad \frac{u \in E \cup \{v\}}{u \in E \quad | \quad u \simeq v} \qquad \in\text{-}E_2 \quad \frac{u \in \emptyset}{\bot}$$

$$\in\text{-}E_3 \quad \frac{u \in \theta(0)}{\bigwedge_{v \in E'} u \not\simeq v} \qquad \in\text{-}E_4 \quad \frac{u \in \theta(\beta + 1)}{\bigwedge_{v \in E''} u \not\simeq v, u \in \theta(\beta)}$$

if $E'$ and $E''$ denote respectively the terms in $\Sigma_\bot$ and $\Sigma[\beta]$ that are of the same sort as $u$.

**Induction Rule** The following rule instantiates the parameter $\mathtt{n}$ by considering the two cases: $\mathtt{n} = 0$ and $\mathtt{n} > 0$. The later case is handled by replacing $\mathtt{n}$ by $\mathtt{n}+1$.

$$(\text{Ind}) \quad \frac{\Phi}{\Phi[0/\mathtt{n}] \mid \Phi\{\mathtt{n} \mapsto \mathtt{n} + 1\}}$$

where $\Phi[0/\mathtt{n}]$ denotes the set of $A$-formulæ obtained from $\Phi$ by replacing $\mathtt{n}$ by 0, and by replacing all iterations $\Pi_{\mathtt{i}=0}^{0} \phi$ (with $\Pi \in \{\bigwedge, \bigvee\}$) by $\phi\{\mathtt{i} \mapsto 0\}$.

**Loop Detection** The two following rules aim at preventing divergence, as explained in Section 4.3. The first one simply deletes from a given node all the $A$-formulæ containing a maximal index $\alpha$ (the rule applies with $\alpha = \mathtt{n}{+}2$, but also with $\alpha = 0, 1$ after $\mathtt{n}$ has been instantiated). Of course, the rule does not preserve satisfiability in general, but it preserves satisfiability if the considered node is irreducible by all the previous rules, except (Ind). The intuitive justification is that these rules extract all the relevant information from the formulæ and express it using only symbols indexed by expressions that are strictly smaller than $\alpha$.

For every set of $A$-formulæ and for every arithmetic expression $\alpha$, we denote by $\langle S \rangle_\alpha$ the set of $A$-formulæ obtained from $S$ by removing all formulæ containing an index greater or equal to $\alpha$.

$$\text{(W)} \quad \frac{S}{\langle S \rangle_\alpha}$$

if $\alpha$ is the maximal index expression in $S$ and either $\alpha = \mathtt{n} + 2$ or $\alpha \in \mathbb{N}$ and $S$ contains no occurrence of $\mathtt{n}$.

Note that in contrast to the other rules, $S$ and $\langle S \rangle_\alpha$ denote the *labels* of the parent and child nodes (not subsets of these labels). Finally, the rule (L) closes a node that is subsumed by another node in the proof tree:

$$\text{(L)} \quad \frac{S}{\bot}$$

if there exists a node $N'$ in the tableau, distinct from the current one labeled by a set of formulæ $S'$ such that $S' \subseteq S$ and either $N'$ is a leaf or $N'$ occurs in the same branch as the current node

**The Properties of the Calculus** We denote by StabArray the procedure defined by the previous inference rules. We assume that the rules are applied with the following strategy. The rules (L), (W) and (Ind) are applied with a strictly lower priority than the other rules. The rule (L) is applied only if (W) does not apply, and (Ind) applies when no other rule applies.

**Theorem 7.** *The calculus* StabArray *is:*

- *terminating: every tableau is finite;*
- *complete (w.r.t. the class of contiguous interpretations): every irreducible open node has a contiguous model;*
- *sound (w.r.t. the class of contiguous interpretations): an $A$-formula admitting a closed tableau has no contiguous model.*

The number of $A$-formulæ occurring in the tableau is at most exponential[5] w.r.t. the size of the initial formula. Therefore, it is easy to check that the complexity of StabArray is at most doubly exponential (since the number of nodes in the tableau is bounded by the number of sets of formulæ).

[5] The exponential blow-up stems from the atoms of the form $\simeq_E$, where $E$ is a set of terms of arbitrary cardinality.

*Example 8.* We consider the following set of $A$-formulæ: $\{select(t_{\mathtt{n}+1}, u_{\mathtt{n}+1}) \simeq v, select(s, u_{\mathtt{n}+1}) \simeq w, v \not\simeq w, u_{\mathtt{n}+1} \not\simeq u_{\mathtt{n}}, t_0 \simeq s, \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} t_{\mathtt{i}} \simeq_{\{u_{\mathtt{i}}\}} t_{\mathtt{i}+1}\}$. Note that this set has no contiguous model: since $u_{\mathtt{n}+1} \not\simeq u_{\mathtt{n}}$, $u_{\mathtt{n}+1}$ must be distinct from $u_{\mathtt{n}}, u_{\mathtt{n}-1}, \ldots, u_0$, and thus $t_{\mathtt{n}+1}$ necessarily coincides with $t_0$ (hence with $s$) on $u_{\mathtt{n}+1}$. We first apply the rule (S) on the first two formulæ, yielding the branches $t_{\mathtt{n}+1} \not\simeq s$, $u_{\mathtt{n}+1} \not\simeq u_{\mathtt{n}+1}$ and $v \simeq w$. The last two branches can be closed immediately by applying the rules (Ref) and ($\bot$), respectively. Then the rule ($\sim$-I) applies, yielding the two branches: $u_{\mathtt{n}+1} \in E$ (where $E$ is the set of terms in $\Sigma[\mathtt{n}]$ of the same sort as $u_{\mathtt{n}+1}$) and $t_{\mathtt{n}+1} \nsim_{\mathtt{n}} s$ (the other branches can be closed immediately). We have $E = \{u_{\mathtt{n}}\}$, thus the rule $\in$-$E_1$ applies on the first branch, yielding $u_{\mathtt{n}+1} \simeq u_{\mathtt{n}}$, and the branch can be closed due to the formula $u_{\mathtt{n}+1} \not\simeq u_{\mathtt{n}}$. In the second branch, we can apply the rule (Ind). In the base case, we get $t_1 \nsim_0 s$ and $t_0 \simeq_{\{u_0\}} t_1$, hence by (P), $s \simeq_{\{u_0\}} t_1$. Then the rule ($\sim$-E) derives $s \nsim_0 s$ and the branch is closed by ($\sim$-$\bot$). In the inductive case, $\mathtt{n}$ is replaced by $\mathtt{n}+1$, and the formula $t_{\mathtt{n}+1} \simeq_{\{u_{\mathtt{n}+1}\}} t_{\mathtt{n}+2}$ is derived by ($\bigvee$-D). By ($\simeq_E$-R$_1$), we derive $select(t_{\mathtt{n}+1}, u_{\mathtt{n}+2}) \simeq v$ and thus $t_{\mathtt{n}+1} \nsim_{\mathtt{n}+1} s$ can be obtained as in the previous node. Then the rule (W) applies and removes all formulæ containing $\mathtt{n}+2$. We get the set of formulæ: $\{t_{\mathtt{n}+1} \nsim_{\mathtt{n}+1} s, v \not\simeq w, t_0 \simeq s, \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} t_{\mathtt{i}} \simeq_{\{u_{\mathtt{i}}\}} t_{\mathtt{i}+1}\}(\star)$. We apply again the rule (Ind). The base case can be closed immediately as before. In the inductive case, we derive the formulæ $t_{\mathtt{n}+2} \nsim_{\mathtt{n}+2} s$, $t_{\mathtt{n}+1} \simeq_{\{u_{\mathtt{n}+1}\}} t_{\mathtt{n}+2}$, hence by ($\sim$-E) and ($\nsim$-I) we get $t_{\mathtt{n}+1} \nsim_{\mathtt{n}+2} s$ and $t_{\mathtt{n}+1} \nsim_{\mathtt{n}+1} s$. Then the rule (W) applies again, yielding a set of formulæ that is identical to ($\star$). Thus the rule (L) applies, closing the whole tableau (note that some irrelevant rule applications have been omitted for readability).

## 5 Applications

We provide some simple applications of the proposed procedure. The program in pseudo-code below copies in an array $B$ all the elements occurring in an array $A$ and satisfying some property $p$.

```
i ← 0
j ← 0
while i ≤ n do
    if p(A[i]) then
        B[j] ← A[i]
        j ← j + 1
    end if
    i ← i + 1
end while
```

The behavior of this program can be encoded by the following $A$-formula. The parameter $\mathtt{n}$ denotes the number of iterations and $\mathtt{i}$ is the iteration rank. The indexed constants $B_{\mathtt{i}}$ and $\mathtt{j}_{\mathtt{i}}$ denote the value of $B$ and $\mathtt{j}$ at time $\mathtt{i}$. Note that since $A$ is not affected and since it is always indexed by $\mathtt{i}$, it is simpler to encode it as an indexed constant rather than as an array.

$$\mathtt{j}_0 \simeq 0$$
$$\bigwedge_{\mathtt{i}=0}^{\mathtt{n}} (\neg p(A_{\mathtt{i}}) \simeq \mathtt{true} \vee (B_{\mathtt{i}+1} \simeq store(B_{\mathtt{i}}, \mathtt{j}_{\mathtt{i}}, A_{\mathtt{i}}) \wedge \mathtt{j}_{\mathtt{i}+1} \simeq select(succ, \mathtt{j}_{\mathtt{i}})))$$
$$\bigwedge_{\mathtt{i}=0}^{\mathtt{n}} (p(A_{\mathtt{i}}) \simeq \mathtt{true} \vee (B_{\mathtt{i}+1} \simeq B_{\mathtt{i}} \wedge \mathtt{j}_{\mathtt{i}+1} \simeq \mathtt{j}_{\mathtt{i}}))$$

The symbol $succ$ denotes the successor function ($succ(x)$ is written $select(succ, x)$ because all functions are encoded as arrays in our framework). By using a

straightforward typing algorithm, we can infer that $A, B$ and $\mathtt{j}$ are of sorts $\mathtt{nat} \to \mathtt{s}$, $\mathtt{nat} \to \mathtt{s}' \to \mathtt{s}$ and $\mathtt{nat} \to \mathtt{s}'$, respectively. The constants $\mathtt{true}$, $0$, *succ* and $p$ are of sort $\mathtt{bool}$, $\mathtt{s}'$, $\mathtt{s}' \to \mathtt{s}'$ and $\mathtt{s} \to \mathtt{bool}$, respectively. It is clear that the sort $\mathtt{s}$ is non-cyclic. The sort $\mathtt{s}'$ is cyclic, however, it is easy to check that its interpretation is necessarily contiguous, since $\mathtt{j}$ is the only constant of sort $\mathtt{nat} \to \mathtt{s}'$ and since $\mathtt{j}_{\mathtt{i}+1}$ is always obtained from $\mathtt{j}_{\mathtt{i}}$ by applying some increasing function (thus $j_l$ is equal to $j_k$ for $l > k$ only if $j_l = j_{l-1} = \ldots = j_k$).

We can therefore use StabArray to check that all the values stored in the final array (at $\mathtt{j}_1, \ldots, \mathtt{j}_n$) satisfies the property $p$. It suffices to check that the following $A$-formula, together with the previous axioms, is unsatisfiable: $C \simeq B_{\mathtt{n}+1} \wedge \bigvee_{\mathtt{i}=0}^{\mathtt{n}} p(\mathit{select}(C, \mathtt{j}_{\mathtt{i}})) \not\simeq \mathtt{true}$. Note that we *cannot* state in our language the fact that the initial array satisfies the property $p$ at all cells, since the logic does not allow universal quantification.

Conversely, we can also check that all elements occurring in the initial array and satisfying the property $p$ occur in the final array: $C \simeq B_{\mathtt{n}+1} \wedge \bigvee_{\mathtt{i}=0}^{\mathtt{n}} (u \simeq A_{\mathtt{i}} \wedge p(u) \simeq \mathtt{true}) \wedge \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \mathit{select}(C, \mathtt{j}_{\mathtt{i}}) \not\simeq u$

We provide another similar example. The following program interleaves in the same array $C$ the elements occurring in two arrays $A$ and $B$.

```
j ← 0
i ← 0
while i ≤ n do
    C[j] ← A[i]
    C[j + 1] ← B[i]
    j ← j + 2
end while
```

The behavior of the program is modeled as follows.

$$\bigwedge_{\mathtt{i}=0}^{\mathtt{n}} (C_{\mathtt{i}+1} \simeq \mathit{store}(\mathit{store}(C_{\mathtt{i}}, \mathtt{j}_{\mathtt{i}}, A_{\mathtt{i}}), \mathit{select}(\mathit{succ}, \mathtt{j}_{\mathtt{i}}), B_{\mathtt{i}}))$$
$$\mathtt{j}_0 \simeq 0 \wedge \bigwedge_{\mathtt{i}=0}^{\mathtt{n}} \mathtt{j}_{\mathtt{i}+1} \simeq \mathit{select}(\mathit{succ}, \mathit{select}(\mathit{succ}, \mathtt{j}_{\mathtt{i}}))$$

We can check, for instance, that if the array $A$ satisfies some property $p$, then all pairs of consecutive cells in the final array necessarily contain an element satisfying $p$:

$$D \simeq C_{\mathtt{n}+1} \wedge \bigvee_{\mathtt{i}=0}^{\mathtt{n}} (u \simeq \mathtt{j}_{\mathtt{i}} \vee u \simeq \mathit{select}(\mathit{succ}, \mathtt{j}_{\mathtt{i}})) \wedge$$
$$p(\mathit{select}(D, u)) \not\simeq \mathtt{true} \wedge p(\mathit{select}(D, \mathit{select}(\mathit{succ}, u))) \not\simeq \mathtt{true}$$
$$\bigwedge_{\mathtt{i}=0}^{\mathtt{n}} p(A_{\mathtt{i}}) \simeq \mathtt{true}$$

## 6 Conclusion

We have shown that the satisfiability problem is undecidable for schemata of formulæ interpreted in the theory of arrays, and we have defined a restricted class of interpretations (called quasi-contiguous) in which satisfiability can be decided in finite time (with a doubly exponential complexity). Future work includes the implementation of our approach. From a more theoretical point of view, it would be interesting to provide a lower bound for the complexity of satisfiability testing for $A$-formulæ in quasi-contiguous interpretations (only an

upper bound is provided in the present paper). We shall also investigate whether the presented results extend to more expressive theories, including for instance a combination of the theory of arrays and Presburger arithmetic, possibly enriched with additional axioms allowing to express general properties of arrays (e.g., to state that an array is sorted or constant on some interval). As evidenced by the examples in Section 5, such properties can sometimes be expressed in our language, using iterated conjunctions, but this is possible only if the considered interval corresponds to a family of constants. Allowing some restricted form of quantification would therefore enhance the expressive power of the language.

An obvious drawback of our approach is that the conditions on the interpretations are of a semantic nature and thus must be checked by the user. We therefore plan to devise syntactic criteria ensuring that the conditions are satisfied (i.e. ensuring that a satisfiable formula has a contiguous model). The examples in Section 5 suggest that this is feasible, at least in simple cases. It would also be interesting to characterize formally the class of programs and properties that can be modeled in our language.

## References


1. A. Abadi, A. Rabinovich, and M. Sagiv. Decidable fragments of many-sorted logic. *J. Symb. Comput.*, 45(2):153–172, 2010.
2. V. Aravantinos, R. Caferra, and N. Peltier. A schemata calculus for propositional logic. In *TABLEAUX 09 (International Conference on Automated Reasoning with Analytic Tableaux and Related Methods)*, volume 5607 of *LNCS*, pages 32–46. Springer, 2009.
3. V. Aravantinos, R. Caferra, and N. Peltier. Decidability and undecidability results for propositional schemata. *Journal of Artificial Intelligence Research*, 40:599–656, 2011.
4. V. Aravantinos, M. Echenim, and N. Peltier. A resolution calculus for first-order schemata. *Fundamenta Informaticae*, 2013. Accepted for publication, to appear.
5. V. Aravantinos and N. Peltier. Schemata of SMT problems. In *TABLEAUX 11 (International Conference on Automated Reasoning with Analytic Tableaux and Related Methods)*, LNCS. Springer, 2011.
6. A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Information and Computation*, 183(2):140–164, 2003.
7. P. Baumgartner, A. Fuchs, and C. Tinelli. (LIA) - Model Evolution with Linear Integer Arithmetic Constraints. In I. Cervesato, H. Veith, and A. Voronkov, editors, *LPAR*, volume 5330 of *LNCS*, pages 258–273. Springer, 2008.
8. A. R. Bradley and Z. Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
9. A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In E. A. Emerson and K. S. Namjoshi, editors, *Proc. VMCAI-7*, volume 3855 of *LNCS*, pages 427–442. Springer, 2006.
10. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decision procedures for extensions of the theory of arrays. *Annals of Mathematics and Artificial Intelligence*, 50:231–254, 2007.
11. P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about integer arrays? In R. M. Amadio, editor, *FoSSaCS*, volume 4962 of *LNCS*, pages 474–489. Springer, 2008.