Rewriting Systems over Nested Data Words
Invariance checking for systems with dynamic control and data structures

A. Bouajjani C. Drăgoi Y. Jurski M. Sighireanu
{abou,cezarad,jurski,sighirea}@liafa.jussieu.fr

LIAFA, University of Paris Diderot and CNRS, 75205 Paris 13, France

Abstract. We propose a generic framework for reasoning about infinite state systems handling data like integers, booleans etc. and having complex control structures. We consider that configurations of such systems are represented by nested data words, i.e., words of ... words over a potentially infinite data domain. We define a logic called NDWL allowing to reason about nested data words, and we define rewriting systems called NDW-RS over these nested structures. The rewriting systems are constrained by formulas in the logic specifying the rewriting positions as well as structure/data transformations. We define a fragment Σ_2^* of NDWL with a decidable satisfiability problem. Moreover, we show that the transition relation defined by rewriting systems with Σ_2^* constraints can be effectively defined in the same fragment. These results can be used in the automatization of verification problems such as inductive invariance checking and bounded reachability analysis. Our framework allows to reason about a wide range of concurrent systems including multithreaded programs (with procedure calls, thread creation, global/local variables over infinite data domains, locks, monitors, etc.), dynamic networks of timed systems, cache coherence/mutex/communication protocols, etc.

1 Introduction

Automated verification of modern software systems require reasoning about several complex features such as dynamic creation of concurrent threads, data manipulation, procedure calls, timing constraints, etc. For that, infinite-state models must be considered allowing to capture these features, and algorithmic techniques must be designed allowing to cope with these multiple sources of infinity in the state space.

We introduce in this paper a logic-based framework for reasoning about systems with composite (or nested) data structures such as multi-sets of integers, multi-dimensional arrays of integers, arrays of stacks or queues of integers. Nested data structures are also relevant when reasoning about systems with a complex control structure. For instance, the configuration of a program with dynamic thread creation and procedure calls can be naturally modeled as multi-sets of stacks over some potentially infinite domain of data (which can be themselves composite data structures).

Mathematical and Engineering Methods in Computer Science (MEMICS), Znojmo, Czech Republic, 2009.
Petr Hliněný, Vashek Matyáš, Tomáš Vojnar (Eds.)
Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/DROPS.MEMICS.2009.2356

We consider nested data words (NDW for short[1]) as formal objects for the representation of configurations of such systems with complex control and data. The domain of NDW is parameterized by the domain of (scalar) data used in these objects. We propose a logic, NDWL, for reasoning about such objects. The logic NDWL is parametrized by a data logic (a first order logic over the chosen data domain). NDWL allows to constrain the values of the data located at different nesting levels and at different positions in these nested data words. We consider a fragment of NDWL, called Σ_2^*, which consists of all formulas of the form $\exists^{\leq k}\forall^k\exists^{\leq k-1}\forall^{k-1}\ldots\exists^{\leq 1}\forall^1.\,\varphi$ where the quantifiers are on variables ranging over positions at different nesting levels of the structures, and φ is a formula (in the considered data logic) constraining the data attached to these positions. The satisfiability problem is decidable for Σ_2^* whenever the underlying logic on scalar data is. We show that this result can be used for checking automatically the invariance of assertions w.r.t. relations on NDW which are effectively represented in Σ_2^*.

Then, we introduce a class of rewriting systems over NDW, called NDW-RS. Each rewriting rule in a NDW-RS is constrained by formulas in NDWL specifying the rewriting positions and the structure/data transformation at these positions. We associate with each rewriting rule a NDWL formula characterizing the relation on NDWs induced by this rule. Therefore, we obtain a procedure for checking automatically invariance properties w.r.t. this class of models.

Finally, we show that our framework allows to deal with a large class of systems including distributed mutual exclusion protocols, cache coherence protocols, timed networks. In particular, we provide a systematic modeling for multithreaded programs with procedure calls and synchronization by monitors.

Related work. The verification of dynamic/parametrized networks of infinite-state processes has been addressed in several papers such as [2, 13, 10, 4, 11, 9, 8, 14]. All these works consider only one level nesting of data structures, i.e., collections (multisets, arrays, words) over infinite scalar data. Recently, [1, 15] propose a framework allowing two levels of nesting with a special form: N processes may have local arrays of integers of size exactly N. The verification approach used in these works is upper-approximate backward reachability analysis for a particular class of data constraints (gap order constraints on integers). Our work offers a larger framework for modeling and specification. On the other hand, while our framework allows for automatic inductive invariance checking, [13, 1, 4] allow for more automated verification of safety properties based on abstract analysis.

In comparison with [9, 8], this paper presents several significant and nontrivial extensions. First, we consider a more general framework where composite (nested) data structures can be handled. This allows to deal with classes of systems (such as multithreaded programs with infinite data and unbounded number of monitors/locks, etc.) which cannot be handled in the previous frameworks. Second, we consider here a more general class of rewriting systems (with mixed existential and universal rewriting strategies) allowing to model a larger class of communication and synchronization primitives. For instance, broadcast commu-

[1] NDW are not related with "nested words" in [3]

nication cannot be considered in the frameworks defined in our previous work, and the same holds for timing constraints (which require a global synchronization of the clocks).

Let us finally mention that logics on data words/trees have been proposed for reasoning about XML documents [6, 5, 12]. The considered logics and the obtained results in these works are not comparable with ours.

2 Motivation

We are interested in verifying automatically concurrent recursive programs with dynamic process creation, where the processes use data from infinite domains. The processes synchronize by monitors. The control is changed using sequential composition, conditionals, "while" loops, and procedures calls[2]. In the following we give an example of such a program.

Example The program, given in Fig. 1, is written in a Java-like syntax. An array M of monitors is accessed concurrently by threads created during the execution of the program. (The size of M changes by creating threads therefore monitors; the code for thread creation is omitted). Each thread has a unique identifier id$\geq$ 0 and has the task of creating a monitor and putting it in M at index id. All monitors have the same type, Moni, which has one procedure p, i.e., p shall be executed in mutual exclusion. The procedure chooses a number j strictly smaller than the identifier of its owning monitor and, if the monitor M[j] has been created, it calls its procedure. The property to check on such programs is the absence of deadlock due to the mutual waiting on the monitors. The inter-blocking of threads may appear if a thread i_1 has locked the monitor M[j_1] and it is waiting now to lock the monitor M[j_2], while, in parallel, a thread i_2 has locked the monitor M[j_2] and it is waiting now to lock the monitor M[j_1]. The absence of deadlock can be established by checking the invariance property that the call stacks of all threads are always sorted w.r.t. their integer values.

```
Vector<Moni> M = new Vector<Moni>();
monitor Moni {//Monitors definition
  int id;
  procedure p() {
    int j = value in [0,id);
    if (M.get(j)!=null)
       M.get(j).p();
}
thread T {//Threads definition
  int id;
  procedure run() {
    M.set(id, new Moni(id));
    M.get(id).p();
  }
}
```

Fig. 1. Example of program.

Representing program configurations by nested data words The configurations of the program are given by the configuration of the vector M and the configuration of threads, where each of these threads has an attached unbounded call stack, and each of these stacks contains values over the infinite domain of integers (corresponding to the values of variables id and j). We represent the threads configuration as words where each position denotes a thread. Therefore, each

[2] We do not allow pointer manipulation.

position has attached an integer data and a subword over integers, denoting the identity of the process and its call stack. The vector elements are distinguished positions in this nested data word, that have attached only the identity of the monitor and an empty subword. This structure of word of words of ... words we called it *nested data words* (NDW) over a potentially infinite data domain. Let Σ be a finite alphabet, and let $\mathbb{D}$ be a (infinite) *data domain*. The nested data words domain NDW is the union of the family $\{\mathsf{NDW}_k\}_{k\geq 0}$ where (i) for $k \geq 1$, NDW_k contains all sequences indexed by subsets of $\mathbb{N}$ with values in $\Sigma \times \mathbb{D} \times \mathsf{NDW}_{k-1}$, i.e., $\mathsf{NDW}_k = \{w \mid w : \mathbb{N} \rightharpoonup \Sigma \times \mathbb{D} \times \mathsf{NDW}_{k-1}\}$, (ii) NDW_0 contains only the empty word, denoted ϵ, $\epsilon(i)$ is undefined for all $i \in \mathbb{N}$.

The elements of NDW_k are called *nested data words of level k*. Since any $w \in \mathsf{NDW}$ is a partial function we denote by $dom(w)$ the subset of $\mathbb{N}$ where w is defined. We call *indexes* the natural numbers in the domain of $w \in \mathsf{NDW}$; their level is given by the level of the word they index.

Given a word w in NDW_k ($k > 1$) and $p \in dom(w)$, then $label(w[p])$ (resp. $data(w[p])$, $ndw(w[p])$) denotes the first (resp. second, third) member of the tuple $w[p]$. These notations extend to sequences of indexes, e.g., $label(w[p_1, \dots, p_j])$ $(1 < j \leq k)$ denotes the label attached to index p_j of the inner subword $ndw(w[p_1, \dots, p_{j-1}])$ $(label(w[p_1, \dots, p_j]) = label(ndw(w[p_1, \dots, p_{j-1}])[p_j]))$.

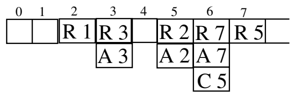

Fig. 2. Element of NDW_2.

Fig. 2 provides an example of a nested data word $w \in \mathsf{NDW}_2$ built on the finite alphabet $\Sigma = \{R, A, B, C, D\}$ and the data domain $\mathbb{D} = \mathbb{N}$. This word is a simplified representation for the configurations of the program in Section 1 (the value of j is omitted and also the elements of the array M are omitted). The domain of w is $\{2, 3, 5, 6, 7\}$, i.e., there are five created threads. The labels A,B,C, and D denote the control points at line 14, 8, 7, 4 respectively. These control points are important because they correspond to calls of /returns from the procedure p. Notice that, $label(w[6, 0]) = A$, $data(w[6]) = 7$ and $ndw(w[6]) = [0 \mapsto (A, 7), 1 \mapsto (C, 5)]$ (the integer numbers in $ndw(w[6])$ are the identities of the monitors locked by the thread with the identity $data(w[6]) = 7$).

Reasoning about programs To prove safety properties (e.g., the absence of deadlock) we use invariant checking. Given a set of initial configurations $Init$, a set of safe configurations $Safe$ and a set of configurations Inv, we have to check that Inv is an inductive invariant and that $Inv \subseteq Safe$. Inv is an inductive invariant if (1) $Init \subseteq Inv$, and (2) for every statement st of the program, $post(st, Inv) \subseteq Inv$, where $post(st, Inv)$ denotes the set of configurations obtained by executing st on Inv.

We give a logical framework to specify properties of program configurations and transformations between configurations. We define a multi-sorted second order logic called *nested data word logic*, NDWL. Sets of configurations, like $Init, Inv$, and $Safe$ are modeled by formulas in NDWL, $\varphi_{Init}, \varphi_{Inv}$, resp. φ_{Safe} and the relation between configurations defined by $post(st, \bullet)$ is a formula $\varphi_{post(st)}(\gamma, \gamma')$ where γ and γ' represent the configuration before resp. after the execution of statement st. Then, a formula φ_{Inv} is an *inductive invariant* if

(1) $\varphi_{Init}(\gamma) \wedge \neg\varphi_{Inv}(\gamma)$ is unsatisfiable and (2) for each program statement st, $\varphi_{Inv}(\gamma) \wedge \varphi_{post(st)}(\gamma, \gamma') \wedge \neg\varphi_{Inv}(\gamma')$ is unsatisfiable.

Formulas in NDWL can specify, properties of the global variables and configurations of processes. For example, using NDWL one can specify properties on the call stack of some process, relations between two call stacks, or relations between global and local variables.

3 Nested data word logic NDWL

The logic NDWL is parameterized by a (first-order) logic $\mathsf{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$ on the considered data domain $\mathbb{D}$, i.e., by the set of operations $\mathbb{O}$ and by the set of basic predicates (relations) $\mathbb{P}$ allowed on elements of $\mathbb{D}$.

Syntax Consider the following pairwise disjoint sets of variables: (1) D (of elements denoted by $b, c, d, \dots$) is the set of *data variables* taking values in $\mathbb{D}$, (2) $\varGamma$ (of elements denoted by $\gamma, \gamma', \gamma_1, \dots$) is the set of *nested data word variables* taking values in NDW, (3) I (of elements denoted by $x, y, \dots$) is the set of *index variables* taking values in $\mathbb{N}$, and (4) $\mathcal{I}$ (of elements denoted by $X, Y, \dots$) is the set of *index-set variables* taking values in $2^{\mathbb{N}}$.

Additionally, each variable, which is not a data variable, is indexed by a number from 1 to N called *the level* of the variable. These levels define a partition on $\varGamma, I, \mathcal{I}$: $\varGamma = \bigcup_{1 \le k \le N} \varGamma_k$, resp. $I = \bigcup_{1 \le k \le N} I_k$, $\mathcal{I} = \bigcup_{1 \le k \le N} \mathcal{I}_k$ where $\varGamma_k$ (resp. I_k, $\mathcal{I}_k$) is the set of nested data word (resp. index, index-set) variables of level k. The syntax of *terms* and *formulas* in NDWL is given by the following grammars:

$$
\begin{aligned}
t &::= d \mid o(t_1, \dots, t_m) \mid \upsilon(\mathit{tt}[x_0, \dots, x_p]) \qquad \mathit{tt} ::= \gamma \mid \delta(\mathit{tt}[x_0, \dots, x_p]) \\
\varphi &::= \mathtt{true} \mid r(t_1, \dots, t_m) \mid A(\mathit{tt}[x_0, \dots, x_p]) \mid 0 < x \mid x < x' \mid x \in X \mid \mathsf{idx}(x, \mathit{tt}) \\
&\quad \mid \exists x.\ \varphi \mid \exists d.\ \varphi \mid \neg\varphi \mid \varphi \vee \varphi
\end{aligned}
$$

where o is an operation in $\mathbb{O}$ of arity $m \ge 0$, t is a data term and tt is a nested data word term (ndw-term for short), $x, x', x_0, \dots, x_p$,$(p \ge 0)$ are in I, r is a predicate in $\mathbb{P}$ of arity m, $A \in \varSigma$, 0 is a constant index, $X \in \mathcal{I}$, $d \in D$, and $\gamma \in \varGamma$. The elements generated by this grammar respect the following level constraints: (1) x and x' have the same level in $x < x'$ (2) x and tt have the same level in $\mathsf{idx}(x, \mathit{tt})$ (3) x and X have the same level in $x \in X$ (4) if tt is a ndw-term of level k then $\delta(\mathit{tt}[x_0, \dots, x_p])$ (resp. $\upsilon(\mathit{tt}[x_0, \dots, x_p])$) is a ndw-term of level $k - p - 1$ (resp. a data term) and then x_0, x_1, … , x_p have levels k, $k - 1$, … , $k - p$ and $k - p \ge 1$; (5) $t_1, \dots, t_m$ are data terms, i.e., they have level 0, in $r(t_1, \dots, t_m)$ and in $o(t_1, \dots, t_m)$; (6) if tt is a ndw-term of level k in $A(\mathit{tt}[x_0, \dots, x_p])$ then $k - p \ge 1$ and x_0, x_1, … , x_p have levels k, $k - 1$, … , $k - p$.

As usual, conjunction ($\wedge$), implication ($\Rightarrow$), and universal quantification ($\forall$) can be defined in terms of $\neg$, $\vee$, and $\exists$. We also define equality ($=$), disequality ($\neq$) and inequality ($\le$) in terms of $<$ and boolean connectives. To emphasize the level of some quantified variable, we use notations $\exists^k$ (resp. $\forall^k$) instead of $\exists$ (resp. $\forall$).

Notice that the variables in $\varGamma$ and $\mathcal{I}$ are free in any NDWL formula. We assume w.l.o.g. that in every formula, each variable is quantified at most once.

Semantics Formally, a model of a NDWL formula is a mapping $\mathcal{M} : \mathbb{N}^* \to 2^{\mathbb{N}}$ which gives for each level k the set of positions defined within the model (notation $\mathcal{M}_k = \mathcal{M}(k)$) and valuations of free variables. A valuation of index variables is a partial mapping $\rho \in [I \rightharpoonup \mathbb{N}]$ s.t. variables in I_k take values in $\mathcal{M}_k$. We extend ρ by $\rho(0) = 0$. A valuation of index-set variables is a partial mapping $\nu \in [\mathcal{I} \rightharpoonup 2^{\mathbb{N}}]$ s.t. for any variable $X \in \mathcal{I}_k$, $\nu(X) \subseteq \mathcal{M}_k$. A valuation of data variables is a partial mapping $\beta \in [D \rightharpoonup \mathbb{D}]$. A valuation of NDW variables is a partial mapping $\theta \in [\Gamma \rightharpoonup \mathsf{NDW}]$ s.t. variables in Γ_k take values in NDW_k. Moreover, for all variables $\gamma \in \Gamma_k$, $dom(\theta(\gamma)) \subseteq \mathcal{M}_k$, and so on recursively, i.e., for any subword of $\theta(\gamma)$, $w \in \mathsf{NDW}_\ell$ with $1 \leq \ell < k$, $dom(w) \subseteq \mathcal{M}_\ell$.

The data terms t are interpreted into values in $\mathbb{D}$; they denote the values stored at different positions of some (inner) word. The ndw-terms tt are interpreted into nested data words of the corresponding level. Formulas in NDWL can express ordering relations between indexes ($x < x'$) and the membership relation between an index and an index-set ($x \in X$) or an index and the definition domain of a nested data word ($\mathsf{idx}(x, tt)$). Intuitively, $A(tt[x_0, \ldots, x_p])$ says that tt is interpreted into some word w and the indexes $x_0, \ldots, x_p$ form a valid path to an inner word of w s.t. $label(w[x_0, \ldots, x_p]) = A$. The concept of valid path refers to the fact the x_0 must be defined in w, $x_1 \in dom(ndw(w[x_0]))$ and so forth x_p must be defined in $ndw(w[x_0, \ldots, x_{p-1}])$.

In general, a term $\delta(tt[x_0, \ldots, x_p])$ or $\upsilon(tt[x_0, \ldots, x_p])$ is *well defined* if $x_0, \ldots, x_p$ forms a valid path to an inner word of the word tt interprets into. In the following we consider that γ interprets into w the nested data word pictured in Fig. 2. Then, when $x = 6$ the term $\delta(\gamma[x])$ denotes $ndw(w[6])$ and $\upsilon(\gamma[x])$ interprets into 7. The atomic formulas built over non well defined terms are false. This fact might induce some difficulties when reasoning about nested structures. E.g., the formula $\forall^2 y \exists^1 x.\ \neg A(\gamma[y, x])$ saying that any subword of γ shall have an index not labeled by A, has a model in which γ is interpreted to w. This happens even if all defined positions in $ndw(w[2])$ are labeled by A; in this case, a value for x that satisfies the property is an index of $\mathcal{M}_1$ not defined in $ndw(w[2])$, e.g., $1 \in \mathcal{M}_1$ since $1 \in dom(ndw(w[6]))$ but $1 \notin ndw(w[2])$. To obtain the intuition and reject this model, we must specify that only indexes y in the domain of w are considered: $\forall^2 y \exists^1 x.\ \mathsf{idx}(y, \gamma) \Longrightarrow \mathsf{idx}(x, \gamma[y]) \wedge \neg A(\gamma[y, x])$.

Examples In the following, we consider that γ is interpreted to w, the nested data word w in Fig 2. Then, the following formula states that all threads in the configuration γ are running and their identity is smaller than 10: $\forall^2 y.\ \mathsf{idx}(y, \gamma) \Longrightarrow R(\gamma[y]) \wedge \upsilon(\gamma[y]) \leq 10$. The formula $\forall^2 y \exists^1 x.\ \mathsf{idx}(y, \gamma) \Longrightarrow \mathsf{idx}(x, \gamma[y]) \wedge \upsilon(\gamma[y, x]) \geq 2$ says that all the inner words of w have an index whose data is at least 2. Finally, the next formula says that all the threads in the configuration denoted by γ have their call stack (represented by the inner word) sorted w.r.t. the identity of the owned monitors: $\forall^2 z \forall^1 x, y.\ (x < y \wedge \mathsf{idx}(z, \gamma) \wedge \mathsf{idx}(x, \gamma[z]) \wedge \mathsf{idx}(y, \gamma[z])) \Longrightarrow \upsilon(\gamma[z, x]) > \upsilon(\gamma[z, y])$.

Syntactical forms and fragments A formula is in *prenex normal form* (PNF) if it is of the form $Q_1 z_1 Q_2 z_2 \ldots Q_m z_m.\ \varphi$ where (1) $Q_1, \ldots, Q_m \in \{\exists, \forall\}$, (2)

$z_1, \ldots, z_m \in I \cup D$, and (3) φ is a quantifier-free formula. It can be proved that for every formula φ in NDWL, there exists an equivalent formula in PNF.

We consider $\{\Sigma_2^\ell\}_{\ell \geq 0}$ and $\{\Theta_1^\ell\}_{\ell \geq 0}$ two fragments of NDWL defined by restricting the quantifier alternation over index variables of the same level in PNF formulas. We define $\Sigma_2^* = \bigcup_{\ell \geq 0} \Sigma_2^\ell$ and $\Theta_1^* = \bigcup_{\ell \geq 0} \Theta_1^\ell$ where

$$\Sigma_2^\ell = \{Q_l \ldots Q_2 Q_1 .\ \varphi \mid Q_i = \exists^{\leq i} \overrightarrow{x}_i \exists \overrightarrow{d}_i \forall^i \overrightarrow{y}_i,\ 1 \leq i \leq l, \text{ and } \varphi \text{ quantifies over } D\}$$
$$\Theta_1^\ell = \{S_l \ldots S_1 .\ \phi \mid S_i = \exists^i \overrightarrow{x} \text{ or } S_i = \forall^i \overrightarrow{x},\ 1 \leq i \leq l, \text{ and } \phi \text{ quantifies over } D\}$$

Notice that, Θ_1^* is a subset of Σ_2^* which is closed under all boolean operations while Σ_2^* is closed only under disjunction and conjunction. All the formulas given as example above are in the Θ_1^* fragment.

4 Application to verification

The satisfiability problem for the full NDWL is not decidable. E.g., [8] proves the undecidability of this problem for a subfragment of NDWL, which allows $\forall^* \exists^*$ quantification over variables of the same level. The next theorem provides a positive result for a fragment of NDWL.

Theorem 1. *Whenever the data logic* $\mathsf{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$ *has a decidable satisfiability problem, the satisfiability problem of the fragment* Σ_2^* *of NDWL is also decidable.*

The proof of this theorem is similar to the decidability proof for CSL logic in [7]. The proof gives a decision procedure whose complexity is NP when the number of universally quantified variables is fixed. Moreover, the structure of the nested data words is simpler than the one of the heap graphs in CSL which leads to a more efficient implementation for the decision procedure. The decidability result in Theorem 1 is used to automate invariant checking.

Theorem 2. *Checking that a formula* $\varphi_{Inv} \in \Theta_1^*$ *is an inductive invariant is decidable for specifications with the transition relation* $\varphi_{post(st)}$ *in* Σ_2^* *for every program statement st, when the underlying logic,* $\mathsf{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$*, has a decidable satisfiability problem.*

In the next section, we introduce a formalism to specify the transition relation of a system, i.e., φ_{post}, with formulas in Σ_2^*. This formalism handles a large class of complex systems with interesting control structures like rendez-vous, broadcast, procedure call, process creation, locks.

5 Rewriting systems over nested data words

A *nested data words rewriting system* (NDW-RS for short) is a pair $RS = (\Sigma, \Delta)$ where Σ is a finite set of labels, and Δ is a finite set of rewriting rules. Each rule may be an *existential rule*, an *universal rule*, or a general (mixed existential and universal) rule. In the following, we give the syntax and the intuitive semantics of the rewriting rules.
The existential rules have the following syntax:

$$\overrightarrow{A} \hookrightarrow_\sharp \overrightarrow{B} : \varphi_g \ / \ \varphi_a \qquad (1)$$

where $\overrightarrow{A}$ and $\overrightarrow{B}$ are labels in Σ, $\sharp$ is the *rewriting policy* that can be one of multiset ($\sharp = m$), factor ($\sharp = f$), or suffix ($\sharp = s$), and φ_g and φ_a are NDWL formulas.

An existential rule selects the nested data word rewritten using the guard φ_g. On this word, the rule rewrites according to the policy the indexes labeled by $\overrightarrow{A}$ and constrained using φ_g into indexes labeled by $\overrightarrow{B}$ whose data are assigned using φ_a. In φ_g and φ_a, the path to the rewritten word, the indexes labeled by $\overrightarrow{A}$, and those labeled by $\overrightarrow{B}$ are denoted using variables $\overrightarrow{\xi}$, $\overrightarrow{x}$, resp. $\overrightarrow{y}$. Also, the initial (resp. resulting) nested data word is denoted by the NDW variable γ (resp. γ'). Fig. 3 (1) gives an example of a thread creation in the configuration given in Fig. 2 modeled by the existential rule $R1$ below:

$$\text{R1}: \ R \hookrightarrow_m R\ R \ : \ v(\gamma[x_1]) \geq 2 \ / \ v(\gamma'[y_1]) = v(\gamma[x_1]) \wedge \delta(\gamma'[y_1]) = \delta(\gamma[x_1]) \wedge v(\gamma'[y_2]) = 2v(\gamma[x_1])$$

Intuitively, a thread with the identity not smaller than 2 spawns a new thread with the identity doubled. Formally, the rule rewrites γ at an index x_1 labeled by R that stores a value $v(\gamma[x_1]) \geq 2$. The rewriting introduces two positions labeled by R: y_1 is a copy of x_1 and y_2 has attached an integer with the data twice the value of $v(\gamma[x_1])$ and an empty sub-word (in Fig. 3 (1) a thread with the identity 4 is spawned).

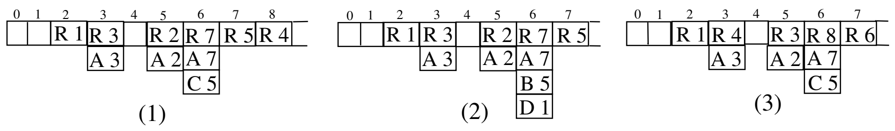

Fig. 3. Applying R1, R2 resp. R3 on the word of Fig. 2.

Note that more than one choice is possible for the indexes that are rewritten, i.e., $\overrightarrow{x}$ and $\overrightarrow{y}$. The rewriting policy refines the selection of these indexes. For example, the multiset policy puts no ordering relation between indexes in $\overrightarrow{x}$ (resp. $\overrightarrow{y}$); it says that rewriting concerns $(min(|\overrightarrow{x}|, |\overrightarrow{y}|))$ positions from γ satisfying φ_g plus some added (resp. removed) positions that were not defined in the initial (resp. resulting) word. For example, in the rule R1 $y_2 \notin dom(\gamma)$ but it is defined in γ', $y_2 \in dom(\gamma')$.

The suffix rewriting policy says that $\overrightarrow{x}$ (and $\overrightarrow{y}$) are the last $|\overrightarrow{x}|$, (resp. $|\overrightarrow{y}|$) consecutive positions of the subword rewritten. The call of the procedure p at line 7 in Fig.1 on a monitor with the identity 1 in a thread with the identity 7 is modeled by an existential rewriting rule with suffix rewriting policy:

$$\text{R2}: \ C \hookrightarrow_s B\ D \ : \ R(\gamma[\xi]) \wedge v(\gamma[\xi]) = 7 / v(\gamma[\xi, x_1]) = v(\gamma'[\xi, y_1]) \wedge v(\gamma'[\xi, y_2]) = 1$$

where C is the control point of the process at the call of p, B is the return point after the call of p, and D is the entry control point of p; the process calling p is selected using φ_g, the local data of the new position (labeled by D) is initialised using φ_a. Existential rewriting rules can model communication by shared, global variables or rendez-vous.

The universal rules have the following syntax:

$$\overrightarrow{C} \mapsto \overrightarrow{D} : \psi_g \ / \ \psi_a \tag{2}$$

where $\overrightarrow{C}$ and $\overrightarrow{D}$ are labels in Σ and ψ_g and ψ_a are NDWL formulas.

A universal rule rewrites *all* indexes labeled by $\overrightarrow{C}$ and satisfying the guard ψ_g by replacing their label with the respective label in $\overrightarrow{D}$ and their data with the data assigned in ψ_a. To refer the positions rewritten we use the set of variables $\overrightarrow{u}$ ($|\overrightarrow{u}| = |\overrightarrow{C}| = |\overrightarrow{D}|$). The formulas ψ_g and ψ_a contain as free variables $\overrightarrow{\xi}$, γ, and γ' with the same semantics as in existential rules. Fig. 3 (3) shows the nested data word resulting by applying the universal rule below on the word of Fig. 2:

$$\text{R3}: \quad R \mapsto R \ : \ \upsilon(\gamma[u]) \geq 2 \ / \ \upsilon(\gamma'[u]) = \upsilon(\gamma[u]) + 1$$

The rule increments the identity of all threads having the identity greater than 2. In this way it will be possible to create later a thread with the identity 2.

The mixed rules combine an existential and an universal rule as follows:

$$\overrightarrow{A} \hookrightarrow_\sharp \overrightarrow{B} \ : \ \varphi_g \ / \ \varphi_a \quad | \quad \overrightarrow{C} \mapsto \overrightarrow{D} \ : \ \psi_g \ / \ \psi_a \tag{3}$$

The subword of γ rewritten (given by $\overrightarrow{\xi}$) is fixed in φ_g and ψ_g. The indexes rewritten by the existential part ($\overrightarrow{x}$ and $\overrightarrow{y}$) may be used in ψ_g and ψ_a to choose the indexes $\overrightarrow{u}$ and their new data, i.e., all the four formulas $\varphi_g, \varphi_a, \psi_g, \psi_a$ share the same index variables in $\overrightarrow{\xi}$, $\overrightarrow{x}$, and $\overrightarrow{y}$. These rules model statements like *notifyAll*(), in synchronization by monitors, or time elapsing in networks where processes manipulate clocks.

Formally, the semantics of rewriting any rule R is given by NDWL formulas denoted $\texttt{reach}_R(\gamma, \gamma')$, where γ denotes the word to be rewritten (it satisfies the guard φ_g/ψ_g) and γ' denotes the word after the rewriting.

We denote by NDW-RS$[\Sigma_2^*]$ the class of NDW-RS where any rewriting rule has the constraints φ_g and φ_a in Σ_2^*, ψ_g and ψ_a in Θ_1^k.

Proposition 1. *For every mixted rule of a rewriting system in NDW-RS$[\Sigma_2^*]$, the associated NDWL formula is in the fragment Σ_2^*.*

Then, the following theorem is a consequence of the results given in Section 4.

Theorem 3. *Checking that a formula $\varphi \in \Theta_1^*$ is an inductive invariant is decidable for any system in NDW-RS$[\Sigma_2^*]$, if the underlying logic* $\mathsf{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$ *has a decidable satisfiability problem.*

Notice that all the examples of rewriting rules given in this section belong to a system in NDW-RS$[\Sigma_2^*]$.

6 Conclusion

We have defined a generic framework for reasoning about unbounded networks of processes with complex data and control structures. Various instances of this framework allow to deal in a uniform way with important classes of system models such as dynamic networks of processes with counters, clocks, unbounded/parametric structures (arrays, stacks, queues) over infinite data domains, etc. This is based on generic decidability and closure results for a (useful fragment of a) logic for specifying configurations of such networks as nested data words. Several classes of actions in such networks can be modeled in this logical framework. For example, the process creation, the procedure calls and the *rendez-vous* are modeled by existential rewriting rules while global synchronization between processes (or *broadcast*) is modeled using universal and mixed rewriting rules.

Future work includes extending our framework by developing techniques for compositional verification of concurrent programs.

References

1. P.A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *Proc. of CAV*, volume 4590 of *LNCS*, pages 145–157, 2007.
2. P.A. Abdulla and B. Jonsson. Verifying networks of timed processes (extended abstract). In *Proc. of TACAS*, volume 1384 of *LNCS*, pages 298–312, 1998.
3. R. Alur and P. Madhusudan. Adding nesting structure to words. *J.ACM*, 56(3), 2009.
4. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L.D. Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. In *Proc. of CAV*, volume 2102 of *LNCS*, pages 221–234, 2001.
5. M. Bojanczyk, C. David, A. Muscholl, Th. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. In *Proc. of PODS*, pages 10–19. ACM, 2006.
6. M. Bojanczyk, A. Muscholl, Th. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proc. of LICS*, pages 7–16. IEEE, 2006.
7. A. Bouajjani, C. Drăgoi, C. Enea, and M. Sighireanu. A logic-based framework for reasoning about composite data structures. In *Proc. of CONCUR*, volume 5710 of *LNCS*, pages 178–195, 2009.
8. A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting Systems with Data. In *Proc. of FCT*, volume 4639 of *LNCS*, pages 1–22, 2007.
9. A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *Proc. of TACAS*, volume 4424 of *LNCS*, pages 690–705, 2007.
10. M. Bozzano and G. Delzanno. Beyond Parameterized Verification. In *Proc. of TACAS*, volume 2280 of *LNCS*, pages 221–235, 2002.
11. A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In *Proc. of VMCAI*, volume 3855 of *LNCS*, pages 427–442, 2006.
12. C. David. Complexity of data tree patterns over xml documents. In *Proc. of MFCS*, volume 5162 of *LNCS*, pages 278–289, 2008.
13. G. Delzanno. An assertional language for the verification of systems parametric in several dimensions. *Electr. Notes Theor. Comput. Sci.*, 50(4), 2001.
14. C. Ihlemann, S. Jacobs, and V. Sofronie-Stokkermans. On local reasoning in verification. In *Proc. of TACAS*, volume 4963 of *LNCS*, pages 265–281, 2008.
15. A. Rezine. *Parameterized Systems: Generalizing and Simplifying Automatic Verification.* PhD thesis, University of Uppsala, 2008.