# On the Expressiveness of Multiset Rewriting with Gap-order Constraints

Giorgio Delzanno

Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Via Dodecaneso 35, 16146 Genova - Italy
E-mail: giorgio@disi.unige.it

**Abstract.** Gap-order constraints are a subclass of difference constraints introduced in the field of constraint databases. When incorporated within multiset rewriting, they can be used to specify concurrent systems with an arbitrary number of components each one with integer data paths and weak form of update of local states. In this paper we study the expressive power of the specification language MSR($\mathcal{G}$) resulting from this combination. Our main result is that the expressiveness of the monadic fragment of MSR($\mathcal{G}$) lies between Petri Nets and Turing Machines. The undecidability of reachability distinguishes monadic MSR($\mathcal{G}$) from Petri Nets. The decidability of control state reachability distinguishes monadic MSR($\mathcal{G}$) from Turing Machines. Thus our investigations identify a class of infinite state systems more expressive than Petri Nets for which problems related to the automated verification of safety properties are still decidable. In the paper we also discuss limits to extensions of these results with respect to the constraint language and to the data structures used in the algorithms solving control state reachability.

## 1 Introduction

Multiset rewriting represents a natural extension of basic models for concurrency like Petri Nets. Synchronous and asynchronous communication can be represented in a natural way in this model. The integration of constraints and multiset rewriting provides an elegant way to separate the specification of the process structure (specified in multiset rewriting) from the specification of the relation over data paths (specified using constraints). Furthermore, it provides interesting ways to combine theoretical and practical results obtained in the two fields within a unified framework. According to this line of research, in [8] we have introduced the language scheme MSR($\mathcal{C}$) to combine multiset rewriting over first order atoms [6] with a generic constraint system $\mathcal{C}$. Instances of MSR($\mathcal{C}$) have been applied to model consistency and security protocols for systems with an unbounded number of components in [3, 4, 11].

In this paper we investigate the theoretical properties of a new instance, called MSR($\mathcal{G}$), in which the constraint system $\mathcal{G}$ is that of Revesz's gap-order constraints [22], a subclass of linear integer arithmetics and difference constraints. Gap-order constraints have been introduced in the field of constraint databases

[21]. However, they also found an interesting application for the verification of concurrent systems with finitely many components and integer data paths [16, 15, 14, 21] and to program analysis [23]. When embedding gap-order constraints into MSR, we obtain a language in which it is possible to specify systems with an arbitrary number of components each one with integer data paths. Furthermore, actions can manipulate process data by using a weak form of update of a local variable $x$ expressed by the gap-order constraint $x' > x + c$ for $c \geq 0$.

The resulting specifications represent powerful abstract models for concurrent programs working on integers. To strengthen this claim, in this paper we study the expressive power of fragments of MSR($\mathcal{G}$). Specifically, we will prove that (1) the monadic fragment of MSR($\mathcal{G}$) is strictly more expressive than Petri Nets and strictly less expressive than Turing machines, whereas (3) the dyadic fragment of MSR($\mathcal{G}$) is Turing equivalent. The first result is obtained by showing that reachability is undecidable for monadic MSR($\mathcal{G}$). The proof is based on a weak simulation of two counter machines. The second result follows from the decidability of control state reachability, a natural extension of coverability, for monadic MSR($\mathcal{G}$). To obtain the last result we exhibit an encoding of two counter machines in dyadic MSR($\mathcal{G}$). Furthermore, in order to show that slight extensions of the constraint language can dramatically change the expressiveness of the combined language, we prove that monadic MSR with difference constraints is Turing equivalent.

The proof of decidability of control state reachability requires several efforts. The methodology we follow is that proposed in [1, 13] and extended in [2, 20]. Specifically, we define a symbolic backward reachability algorithm that, starting from a symbolic representation of all configurations in which the control location occurs, computes a finite representation of the set of all of its predecessors. Some general properties of MSR($\mathcal{C}$) help us to define such an algorithm. Specifically, we can instantiate to gap-order constraints the generic symbolic data structure called constrained configuration and the corresponding predecessor operator described in [9]. The difficult part, however, is the definition of an entailment relation to ensure the termination of the resulting symbolic fixpoint computation. For this purpose, we define an alternative symbolic representation of constrained configurations in terms of sets of strings defined over multisets of predicate symbols. The gaps induced by the constraint part are represented using special substrings. This encoding works only for constrained configurations in which the gap-order constraint induces a total order on the corresponding set of free variables. To overcome this obstacle, we have developed a linearization algorithm for gap-order constraints. Our algorithm allows us to represent a constrained configuration using a finite set of linearized constrained configurations. Based on this property, we define an entailment relation over constrained configurations by extending the embedding relation over the string representation of a linearized constrained configuration to a finite sets of strings using the pointwise ordering. The closure properties of better-quasi orderings (see [2, 19]) can be applied then to show that the resulting entailment is a well-quasi ordering. From a general result of [20], this property ensures the termination of symbolic backward reachability. The resulting algorithm can naturally be applied to solve control state reachability for monadic MSR($\mathcal{G}$).

Concerning the expressiveness of the data structures we use in the symbolic backward reachability approach, we will show that it is not possible to extend it to an assertional language closed under difference of denotations. This would immediately gives us a way to decide reachability of two counter machines.

*Plan of the paper* In Section 2 we introduce the MSR($\mathcal{G}$) language (an example is shown in Section A). In Section 3 we study its expressiveness, postponing the proof of decidability of control state reachability of monadic MSR($\mathcal{G}$) to Section 4 In Section 5 we discuss related work. To make this paper self-contained, Section C and D in appendix present some preliminary notions taken from [9, 20, 19].

## 2 Combining MSR with Gap-order Constraints

Gap-order constraints (see [21]) consists of conjunctions $\varphi_1, \ldots, \varphi_n$ of atoms defined over a denumerable set of variables $\mathcal{V}$ where $\varphi_i$ has one of the following forms: $x = y$ or $x + c < y$ (abbreviated as $x <_c y$), where $x, y$ are integer valued variables and $c \geq 0$ is a **non-negative integer** number. In the rest of the paper we will write $x < y$ instead of $x <_0 y$. We will use $Sol(\varphi)$ to denote the set of solutions of $\varphi$, i.e., the set of mappings $\sigma$ from variables in $\mathcal{V}$ to **integers** such that $\sigma(\varphi) \equiv true$. As an example, $\sigma = [x \mapsto 1, y \mapsto 3, z \mapsto 7]$ is a solution for $x <_1 y \wedge y <_2 z$. Gap-order constraints are a strict subclass of difference constraints (i.e. conjunctions of atoms of the form $x - y \leq c$ for some integer $c$): the difference constraint $x = y + c$ (equivalent to $c \leq x - y \leq c$) **is not** gap-order. Furthermore, they are closed under variable elimination [21]: for any gap-order constraint $\varphi$, it is possible to build a gap-order constraint $\psi$ such that $\psi \equiv \exists x.\varphi$ (i.e. we eliminate $x$ from $\varphi$). Following the integration scheme presented in [9], in this section we will define the language MSR($\mathcal{G}$) that combines multiset rewriting over first order atoms [6] and the gap-order constraints. As an example, let $\mid$ be a multiset constructor, then the MSR($\mathcal{G}$) rule

$$p(x) \mid q(y) \rightarrow p(u) \mid t(v) \ : \ x <_1 y, u = y, y <_2 v$$

represents the ground multiset rewriting rules taken by applying solutions of $x <_1 y, u = y, y <_2 v$ to $x, y, u, v$. Thus, starting from a configuration like $p(0) \mid q(5) \mid q(2)$ we can apply the instance $p(0) \mid q(2) \rightarrow p(2) \mid t(6)$ to obtain the next configuration $p(2) \mid t(6) \mid q(5)$. A more extended example is shown in Section A in appendix.

Formally, the language MSR($\mathcal{G}$) is defined as follows. Given a finite set $\mathcal{P}$ of predicate symbols and a denumerable set $\mathcal{V}$ of variables, an atom is defined as $p(x_1, \ldots, x_n)$ (with $n \geq 0$), where $p \in \mathcal{P}$, and $x_1, \ldots, x_n$ are distinct variables in $\mathcal{V}$. A ground atom is defined as $p(d_1, \ldots, d_n)$, where $p \in \mathcal{P}$ and $d_1, \ldots, d_n$ are integer numbers. In the rest of the paper we will use $\mid$ as a constructor for multisets of atoms; multisets will be indicated $\mathcal{M}$, $\mathcal{N}$, etc. Furthermore, we will use $\oplus$ and $\ominus$ to denote *multiset union* and *multiset difference*, respectively, $\epsilon$

to denote the empty multiset, and $\preccurlyeq$ to denote *multiset inclusion*. ($\mathcal{M} \preccurlyeq \mathcal{N}$ if and only if $Occ_A(\mathcal{M}) \leq Occ_A(\mathcal{N})$ for every atomic formula $A$, where $Occ_A(\mathcal{M})$ denotes the number of occurrences of $A$ in $\mathcal{M}$.) Finally, given a mapping $\sigma$ from $\mathcal{V}$ to integers, we will use $\sigma(F)$ to denote the application of its natural extension to formulas.

A **configuration** is a **multiset of ground atoms**, whereas a **rule** is defined as $\mathcal{M} \longrightarrow \mathcal{M}' : \varphi$, where $\mathcal{M}$ and $\mathcal{M}'$ are two multisets of atoms over $\mathcal{P}$ and $V \subseteq \mathcal{V}$, and $\varphi$ is a **gap-order constraint** with variables in $V$. Either $\mathcal{M}$ or $\mathcal{M}'$ can be empty. An MSR($\mathcal{G}$) **specification** is a tuple $\langle \mathcal{P}, \mathcal{V}, \mathcal{M}_0, \mathcal{R} \rangle$, where $\mathcal{M}_0$ is the initial configuration, and $\mathcal{R}$ is a set of rules over $\mathcal{P}$ and $\mathcal{V}$.

The operational semantics describes the update from a configuration $\mathcal{M}$ to one of its possible successor configurations $\mathcal{M}'$. A rule $\mathcal{H} \longrightarrow \mathcal{B} : \varphi$ from $\mathcal{R}$ is enabled at $\mathcal{M}$ via $\sigma \in Sol(\varphi)$ if and only if $\sigma(\mathcal{H}) \preccurlyeq \mathcal{M}$. The firing of $R$ yields the new configuration (written $\mathcal{M} \Rightarrow_R \mathcal{M}'$) defined as $\mathcal{M}' = \sigma(\mathcal{B}) \oplus (\mathcal{M} \ominus \sigma(\mathcal{H}))$. A run is a sequence of configurations $\mathcal{M}_0\mathcal{M}_1 \ldots \mathcal{M}_k$ with $\mathcal{M}_0 \in \mathcal{I}$ and for which there exists a sequences of rules $R_0 R_1 \ldots R_{k-1} \in \mathcal{R}$ such that $\mathcal{M}_i \Rightarrow_{R_i} \mathcal{M}_{i+1}$ for $i \geq 0$. Given a set $S$ of configurations the successor operator *Post* is defined as $Post(S) = \{\mathcal{M}' \mid \mathcal{M} \Rightarrow_R \mathcal{M}',\ \mathcal{M} \in S,\ R \in \mathcal{R}\}$, whereas, the predecessor operator *Pre* is defined as $Pre(S) = \{\mathcal{M} \mid \mathcal{M} \Rightarrow_R \mathcal{M}',\ \mathcal{M}' \in S,\ R \in \mathcal{R}\}$. The reflexive and transitive closure of the predecessor (successor) operator is denoted $Pre^*$ ($Post^*$). Finally, the reachability set is the set $Post^*(\{\mathcal{M}_0\})$.

## 3 Expressive Power

The **monadic** (resp. **dyadic**) fragment of MSR($\mathcal{G}$) is such that the predicates in $\mathcal{P}$ have arity less or equal than one (resp. two). In this section we discuss the expressive power of these fragments. We start from the following observation.

**Proposition 1.** Petri Nets are a subclass of Monadic MSR($\mathcal{G}$).

*Proof.* Petri Nets with $n$ places can be modeled using the zeroary predicates $a_1, \ldots, a_n$. A marking $\boldsymbol{m} = \langle m_1, \ldots, m_n \rangle$ corresponds to a configuration in which we have $m_i$ occurrences of $a_i$ for $i : 1, \ldots, n$. A transition $t$ can be defined then via a rule in which $a_i$ occurs in the left-hand (resp. right-hand) side only if $p_i$ is an input (resp. output) place for $t$. The operational semantics of a Petri Net coincides with that of its propositional MSR($\mathcal{G}$) encoding. □

Already in the monadic case, the introduction of constraints augments the expressive power of the language. To prove this claim, given $S = \langle \mathcal{P}, \mathcal{V}, \mathcal{M}_0, \mathcal{R} \rangle$ we introduce the following problems:

**Definition 1.** Given a subset of predicate symbols $B = \{q_1, \ldots, q_m\} \subseteq \mathcal{P}$, and natural numbers $n_1, \ldots, n_m$, the **sub-predicate reachability problem** consists in deciding if there exists $\mathcal{M}$ such that $\mathcal{M}_0 \rightarrow^* \mathcal{M}$ and $Occ_{q_i}(\mathcal{M}) = n_i$ for $i : 1, \ldots, m$. The **predicate reachability problem** is obtained from the previous definition by taking $S = \mathcal{P}$ for some $n_1, \ldots, n_m$.

The following property holds.

**Proposition 2.** Predicate and sub-predicate reachability are equivalent.

*Proof.* We show that sub-predicate reachability can be represented as predicate reachability. Given $B = \{q_1, \ldots, q_m\} \subset \mathcal{P}$, we define $S' = \langle \mathcal{P}', \mathcal{V}, \mathcal{M}_0', \mathcal{R}' \rangle$ as the MSR($\mathcal{G}$) specification in which: $\mathcal{P}' = \mathcal{P} \cup \{ok, rm\}$, $\mathcal{M}_0' = \mathcal{M}_0 \oplus \{ok\}$ and such that $\mathcal{M} \oplus \{ok\} \rightarrow \mathcal{M}' \oplus \{ok\}$ is in $\mathcal{R}'$ for every rule $\mathcal{M} \rightarrow \mathcal{M}'$ in $\mathcal{R}$. Furthermore, we add the rule (r1) defined as follows

$$\mathcal{D}_1 \oplus \ldots \oplus \mathcal{D}_m \oplus \{ok\} \rightarrow rm \ : \ true$$

where $\mathcal{D}_i = q_i(x_{i,1}) \mid \ \ldots \ \mid \ q_i(x_{i,n_i})$ with $x_{i,j} \neq x_{l,k}$ for $i \neq l$ or $j \neq k$. Rule (r1) detects the desired subconfiguration and blocks the normal operations of $S$ by removing the flag $ok$; $rm$ is in charge of eliminating all atoms via the rule (r2) defined as follows $rm \mid p(x) \rightarrow rm \ : \ true$ for each $p \in \mathcal{P}$. Thus we have that if $\mathcal{M}_0' \rightarrow^* \{rm\}$, then sub-predicate reachability holds for $S$ and $B$. $\square$

For the remaining results, we need to introduce the model of two counter machines (2CM). A 2CM consists of a list of instructions defined over the counters $c_1$ and $c_2$ and the control locations $l_0, \ldots, l_k$. The initial state is $\langle l_0, c_1 = 0, c_2 = 0 \rangle$ and there are two kind of instructions: $Inc(i, p, q) \doteq (l_p\text{: } c_i := c_i + 1\text{; goto } l_q)$, and $Dec(i, p, q, r) \doteq l_p$: (if $c_i > 0$ then $c_i := c_i - 1$; goto $l_q$ else goto $l_r$) for $i \in \{1, 2\}$, $p, q, r \in \{0, \ldots, k\}$. Executions are defined using a standard semantics for the imperative statements. Problems like reachability of a given state or reachability of a given location are undecidable for this computational model.

We will show later that the use of monadic predicates only does not allow us to fully encode the *zero test* needed for a full simulation of two counter machines (2CM). However it is possible to *weakly simulate* them as we show next. A 2CM execution is split into different stages (represented via integer values). The current value of a counter is represented via a multiset of atoms parametric on the current stage. For instance, when the stage is 3 the 2CM state $\langle l_0, c_1 = 1, c_2 = 2 \rangle$ is represented as $l_0(3) \mid c_1(3) \mid c_2(3) \mid c_2(3)$. Operations like increment and decrement for non-zero counters can be executed only within the current stage. As soon as we need to apply the zero test, we first upgrade the stage by choosing a value strictly greater than the current one, then we update, atom by atom, the parameter in the representation of the "other" counter. Finally, we restart the normal operating status in the new stage. As we will show later, some of these steps (e.g. the update of the atoms of the "other counter") can only be approximated using non-deterministic rules. The formal definition of the encoding and its properties are given below.

**Definition 2.** Let $M$ be a 2CM with initial state. The specification $S_M = \langle \mathcal{P}_M, \mathcal{V}, \mathcal{M}_0, \mathcal{R}_M \rangle$ MSR($\mathcal{G}$) specification is such that $\mathcal{M}_0 \doteq \{l_0(0)\}$, and for $i \in \{1, 2\}$ the instruction $Inc(i, p, q)$ is simulated by rule

$$(r1)^i \quad l_p(x) \ \rightarrow l_q(x) \mid c_i(x) \ : \ true$$

and the instruction $Dec(i,p,q,r)$ is weakly simulated by the rules

$$\begin{array}{lll} (r2)^i & l_p(x) \mid c_i(x) \rightarrow l_q(x) & : \; true \\ (r3)^i & l_p(x) \rightarrow update_{r,3-i}(y) \mid last(x) & : \; x < y \end{array}$$

where $update_{r,j}$ is defined for $j \in \{1,2\}$ as

$$\begin{array}{lll} (r4)^j & update_{r,j}(x) \mid last(y) \mid c_j(y) \rightarrow update_{r,j}(x) \mid last(y) \mid c_j(x) & : \; true \\ (r5)^j & update_{r,j}(x) \mid last(y) \rightarrow l_r(x) & : \; true \end{array}$$

Rule $(r2)$ simulates the "then" branch of the decrement instruction, whereas rule $(r3)$ starts the weakly simulation of the "else" branch by selecting a new stage and requesting the update of the stage for the other counter. Rule $(r4)$ is in charge of updating the stage of a counter, whereas $(r5)$ restarts the normal operating status in the new stage.

The following property then holds.

**Lemma 1.** The 2CM $M$ stops with state $\langle l_p, c_1 = 0, c_2 = 0\rangle$ if and only if there exists a run $\mathcal{M}_0 \rightarrow^* \mathcal{M}$ in $S_M$ of Def. 2 such that $Occ_{c_i}(\mathcal{M}) = 0$ for $i : 1, 2$.

*Proof.* Firstly, it is easy to check that all reachable configurations of $S_M$ have one of the following forms $\{l_p(m)\} \oplus \mathcal{M}$ or $\{update_i(m)\} \oplus \mathcal{M}$ where $\mathcal{M}$ is a multiset of atoms of the form $c_i(m)$ with $m \leq n$ $i \in \{1,2\}$.
$(\Rightarrow)$ A single step from $\langle l_p, c_1 = n_1, c_2 = n_2\rangle$ to $\langle l_r, c_1 = m_1, c_2 = m_2\rangle$ can be simulated by a run $\mathcal{M}_1\mathcal{M}_2 \ldots \mathcal{M}_k$ with $k \geq 2$ where for some integers $v, w$ such that $v < w$

- $\mathcal{M}_1 \doteq \{l_p(v)\} \oplus \mathcal{N}_1 \oplus \mathcal{N}_2$, where $Occ_{c_i(v)}(\mathcal{N}_i) = n_i$ for $i : 1, 2$;
- $\mathcal{M}_k \doteq \{l_r(w)\} \oplus \mathcal{N}_1' \oplus \mathcal{N}_2'$, where $Occ_{c_i(w)}(\mathcal{N}_i') = m_i$ for $i : 1, 2$;
- $\mathcal{M}_2, \ldots, \mathcal{M}_{k-1}$ have no occurrences of $l_q$ for $q \in \mathcal{P}$.

This is always true for an application of rules $(r1)$ and $(r2)$. Furthermore, it remains true under the following conditions: $(c1)$: we restrict the application of rule $(r3)^i$ to 2CM states in which $c_i = 0$; $(c2)$: we always apply $(r4)^j$ with $j = 3-i$ as many times as possible so as to replace *all* atoms of the form $c_{3-i}(v)$ with the same number of occurrences of atoms of the form $c_{3-i}(w)$.
Thus, if $M$ stops with state $\langle l_p, c_1 = 0, c_2 = 0\rangle$, then there exists a run from $\mathcal{M}_0$ to a configuration $\mathcal{M}' = \{l_p(w)\}$ for some integer $w \geq 0$.

$(\Rightarrow)$ Now suppose that $\mathcal{M}_0 = l_0(0) \rightarrow^* \mathcal{M}$ for some $\mathcal{M}$ of the form $l_p(w) \oplus \mathcal{M}'$. From $(c1)$ and $(c2)$, if there exists a ground atom $c_i(v)$ in $\mathcal{M}'$ such that $v < w$ for $i \in \{1,2\}$, then the corresponding run does not simulate a computation of the 2CM $M$. Since the rules $r1 - r5$ always operate on atoms defined on the current stage, for every configuration $\mathcal{M}'$ such that $\mathcal{M} \rightarrow^* \mathcal{M}'$ we have that $Occ_{c_i(v)}(\mathcal{M}') \geq 1$, i.e., $Occ_{c_i}(\mathcal{M}')$ will never become zero. Thus, we can distinguish good simulations from bad ones by simply checking for the presence of an atom like $c_i(v)$ such that $v$ is less or equal than the current stage. ◻

Now let the **reachability problem** for an MSR($\mathcal{G}$) specification $S$ and a configuration $\mathcal{M}$ consists in deciding if $\mathcal{M}_0 \rightarrow^* \mathcal{M}$. Then, we have that:

**Theorem 1.** The reachability problem is undecidable for monadic MSR($\mathcal{G}$).

*Proof.* It is well known that 2CM reachability is equivalent to reachability of the special state $\langle l_p, c_1 = 0, c_2 = 0\rangle$. From Lemma 1 and Prop. 2 and since 2CM reachability is undecidable we have that predicate reachability is undecidable, too. We conclude by observing that predicate reachability is a special case of reachability. Indeed, if we take $S = \mathcal{P}$ and remove rule $(r2)$ from the encoding used in the proof of Prop. 2, then $\mathcal{M}'_0 \rightarrow^* \{rm\}$ if and only if $\mathcal{M}_0 \rightarrow^* \mathcal{D}_1 \oplus \ldots \oplus \mathcal{D}_k$ where $\mathcal{D}_i$ is a multiset used to detect $n_i$ occurrences of predicate $p_i$ as in the proof of Prop. 2. $\square$

Since reachability is decidable for Petri Nets we conclude that:

**Corollary 1.** Petri Nets are strictly less expressive than monadic MSR($\mathcal{G}$).

In the following section (see Theorem 4) will show that the following weaker form of reachability problem is decidable for monadic MSR($\mathcal{G}$):

**Definition 3.** Given $\mathcal{P} = \{p_1, \ldots, p_k\}$ and natural numbers $n_1, \ldots, n_k$, the **control state reachability** problem consists in deciding if there exists $\mathcal{M}$ such that $\mathcal{M}_0 \rightarrow^* \mathcal{M}$ and $Occ_{p_i}(\mathcal{M}) \geq n_i$ for $i : 1, \ldots, k$.

This problem is the natural generalization of the coverability problem for Petri Nets. In contrast with Theorem 4 we have that:

**Theorem 2.** Control state reachability is undecidable for dyadic MSR($\mathcal{G}$).

*Proof.* We can reuse the encoding of 2CMs in MSR with linear constraints described in [9] (the constraints used in the encoding are indeed gap-order). For completeness, we reformulate it in Section B in appendix. $\square$

**Corollary 2.** Monadic MSR($\mathcal{G}$) is strictly less expressive than Dyadic MSR($\mathcal{G}$).

Small extensions of the constraint language can drastically change the expressive power of the resulting MSR language as shown below.

**Theorem 3.** Monadic MSR defined over difference constraints (in which atoms have the form $x - y \leq c$) is Turing equivalent.

*Proof.* The 2Cm encoding is defined as follows: $l_0 \mid c_1(0) \mid c_2(0)$ models the initial state, and $l_p \mid c_i(x) \rightarrow l_q \mid c_i(x') : \varphi$ models an instruction on counter $c_i$ where $\varphi \equiv x' = x + 1$ models increment, $\varphi \equiv x > 0, x' = x - 1$ models decrement when $c_i > 0$, and $\varphi \equiv x = 0, x' = x$ models the test $c_i = 0$. $\square$

## 4 Control State Reachability for Monadic MSR($\mathcal{G}$)

Given an initial configuration $\mathcal{M}_0$ and given natural numbers $n_1, \ldots, n_k$, the control state reachability problem of Def. 3 consists in deciding if one of the configurations in the infinite set $\mathcal{T} = \{\mathcal{M} \mid Occ_{p_i}(\mathcal{M}) \geq n_i,\ i : 1, \ldots, k\}$ is reachable from $\mathcal{M}_0$. Notice that $\mathcal{T}$ is upward closed with respect to multiset inclusion, i.e., if $\mathcal{M} \in \mathcal{T}$ and $\mathcal{M} \preccurlyeq \mathcal{M}'$, then $\mathcal{M}' \in \mathcal{T}$. Furthermore, for MSR($\mathcal{G}$) specifications if $\mathcal{T}$ is upward closed, so is $Pre(\mathcal{T})$. On the basis of this property, we can try to apply the methodology proposed in [2] to develop a procedure to compute a finite representation $R$ of $Pre^*(\mathcal{T})$. For this purpose, we need the following ingredients: (1) a symbolic representation of upward closed sets of configurations (e.g. a formula $M$ whose denotation $[\![M]\!]$ is $\mathcal{T}$); (2) a computable symbolic predecessor operator $SPre$ working on sets of formulas such that $[\![SPre(S)]\!] = Pre([\![S]\!])$. (3) a (decidable) entailment relation $Ent$ to compare the denotations of symbolic representations, i.e., such that $Ent(N, M)$ implies $[\![N]\!] \subseteq [\![M]\!]$. $Ent$ can be naturally extended to sets of formulas as follows: $Ent^S(S, S')$ iff forall $N \in S$ there exists $M \in S'$ such that $Ent(N, M)$ holds (clearly, if $Ent$ is an entailment, then $Ent^S(S, S')$ implies $[\![S]\!] \subseteq [\![S']\!]$). Now suppose that $M$ is the formula representing $\mathcal{T}$ (the "target" set of configurations). Then, the procedure computes a chain $\{I_i\}_{i \geq 0}$ of sets of formulas such that $I_0 = \{M\}$ and $I_{i+1} = I_i \cup SPre(I_i)$ The procedure stops when $SPre$ produces only redundant information, i.e., $Ent^S(I_{i+1}, I_i)$. Notice that $Ent^S(I_i, I_{i+1})$ always holds since $I_i \subseteq I_{i+1}$. To prove termination, we can resort to the following lemma (taken from [20]).

**Lemma 2.** If $Ent$ is a well quasi ordering, then for each chain $I_0 \subseteq I_1 \subseteq \ldots$ there exists $i$ such that $Ent^S(I_{j+1}, I_j)$ for all $j \geq i$.

This implies that if $Ent$ is a well quasi ordering than the procedure always terminates. $Ent$ is a *well quasi ordering* if for each infinite sequence of constrained configurations $\mathcal{M}_0\ \mathcal{M}_1\ \mathcal{M}_2\ \ldots$ there exist indexes $i < j$ such that $Ent(\mathcal{M}_j, \mathcal{M}_i)$. In the rest of this section we will investigate the different ingredients needed to obtain an algorithm solving control state reachability.

### 4.1 Symbolic Representation and Predecessor Operator

As a first attempt, we could try to represent the set $T$ by a special configuration $\mathcal{M}_T$ containing $n_i$ occurrences of predicate $p_i$ for $i : 1, \ldots, k$ and to interpret it using an"upward closed semantics". To illustrate, suppose $\mathcal{P} = \{p_1, p_2\}$ and $n_1 = 2, n_2 = 1$. Then, we could use the formula $M = p_1(x) \mid p_1(y) \mid p_2(z)$ with $x, y, z$ as free variables to represent all configurations $\mathcal{M}'$ that contain a ground instance of $M$ (e.g. $p_1(1) \mid p_1(2) \mid p_2(0) \mid p_2(3)$ etc.). However, this representation is too rough to symbolically compute predecessor configurations. As an example, given the rule $p_1(x) \mid p_2(y) \ \rightarrow a \ : \ x < y$, the formula $p_1(x) \mid p_2(y)$ (equipped with the aforementioned upward closed semantics) over-approximates the set of predecessors of $a$ ($p_1(1) \mid p_2(0)$ is not a predecessor of $a$). To solve this problem, we can resort to the notion of constrained configuration [9]. A (monadic) **constrained configuration** over $\mathcal{P}$ is a formula $p_1(x_1) \mid \ldots \mid p_n(x_n) : \varphi$, where $p_1, \ldots, p_n \in \mathcal{P}$ (not necessarily distinct), $\varphi$ is a *satisfiable* gap-order constraint, and $x_1, \ldots, x_n$ are *distinct* variables in $\mathcal{V}$. The denotation a constrained configuration $M \doteq (\mathcal{M} : \varphi)$ is defined by taking the upward closure with respect to multiset inclusion of the set of ground instances, namely $[\![M]\!] = \{\mathcal{M}' \mid \sigma(\mathcal{M}) \preccurlyeq \mathcal{M}',\ \sigma \in Sol(\varphi)\}$. To manipulate constrained configurations, we can instantiate to gap-order constraints the **symbolic predecessor operator** $SPre$ defined for a generic constraint system in [9]. (see also Appendix C). From the general properties proved in [9] and from the closure of gap-order constraints under variable elimination (see [21] and Appendix C), we have that when applied to a finite set of constrained configurations $S$, $SPre$ returns a finite set of constrained configuration such that $[\![SPre(S)]\!] = Pre([\![S]\!])$, i.e., $SPre(S)$ is a symbolic representation of the immediate predecessors of the configurations in the denotation (an upward closed set) of $S$.

The definition of an adequate entailment relation is the key point for obtaining a terminating procedure. We will deal with this problem in the next section.

### 4.2 Entailment

The methodology of [2] can help us to find an entailment relation for monadic constrained configurations that is also a well-quasi ordering. This methodology is based on closure properties of well- and better-quasi orderings under operations like strings, sets and multisets construction (see Prop. 5 in appendix). To exploit these properties, we will try to find an alternative representation of constrained configurations using data structures like strings and multisets. To illustrate, let us consider the following (monadic) constrained configuration

$$M = (p(x) \mid q(y) \mid q(z) \mid r(t) \ : \ x <_2 y, y <_1 z, z = t)$$

By reasoning modulo the equivalence induced by $=$ on the set of variables $\{x, y, z, t\}$, we obtain three classes of equivalence $C_1 = \{x\}$, $C_2 = \{y\}$, and $C_3 = \{z, t\}$ ordered as follows $C_1 <_2 C_2 <_1 C_3$. To each one of these classes we associate the multisets of predicates in which the corresponding variables occur:

$$\mathcal{M}_1 = \{p\} <_2 \mathcal{M}_2 = \{q\} <_1 \mathcal{M}_3 = \{q, r\}$$

To transform this structure into a string of multisets, we have to represent the gaps 2 and 1 using special substrings of multisets. For this purpose, we use a substring consisting of repeated occurrences of a singleton multiset containing a fresh predicate symbol $\Box \notin \mathcal{P}$:

$$Str(M) = \{p\} \cdot \{\Box\} \cdot \{\Box\} \cdot \{q\} \cdot \{\Box\} \cdot \{q, r\}$$

The string representation can only be applied to a constrained configuration $\mathcal{M} : \varphi$ such that its variables $V$ are totally ordered by $\varphi$. In general, however, $\varphi$ induces a partial order on $V$. As an example, in the constraint $x <_2 z, u <_1 v$ there is no explicit relation between $x$ and $u$. Thus as a preliminary step we need to linearize gap-order constraints.

**Linearization.** A gap-order constraint can be viewed as a graph in which nodes are variables and edges are labeled by $=$ or $<_c$ (gap-graph [21]). Given a gap-order constraint $\varphi$ with variables in $V$, we would like to compute an equivalent **finite set** of **gap-order** constraints $S = \{\psi_1, \ldots, \psi_n\}$ in which for $i : 1, \ldots, n$ $\psi_i$ has the following property: the gap-graph associated to $\psi_i$ is a single path involving all variables in $V$. This special linearization problem has not been investigated in previous work on gap-order constraints (see [21]). In [10] we present an algorithm, that, for brevity, we only illustrate with the help of the following example.

Let us consider the gap-order constraint $\varphi = x <_2 z, u <_1 v$ over the set of variables $V = \{x, z, u, v, t\}$. In order to compute all linearizations induced by $\varphi$ on the set $V$, we first consider all total orderings of $V$ on $(= \cup <)$ that are consistent with $\varphi$. Since $V$ is a finite set, we have finitely many possible choices. As an example, the total order (gap-order constraint) $\xi = x < u, u < z, z < v, v = t$ is consistent with $\varphi$. Note however that $\xi$ does not entail $\varphi$. Indeed, we still need to determine the "gaps" $\langle a, b, c\rangle$ such that $x <_a u <_b z <_c v = t$ entails $\varphi$. Although there can be infinitely many such gaps. it is possible to calculate a finite set of "generators" of all gaps. We first notice that, since all gaps must be non negative, then $a, b, c \geq 0$ must hold. Furthermore, since the sum of the gaps $a$ and $b$ must be at least as large as the gap between $x$ and $z$ in $\varphi$, then $a + b \geq 2$ must hold. Similarly, we have that $b + c \geq 1$ must hold. This observation leads us to the following system of inequalities: $a + b \geq 2, b + c \geq 1$ where $a, b, c \geq 0$. Being interpreted over natural numbers, this kind of formulas always admit a finite set of minimal solutions. In our example any solution $s = \langle a, b, c\rangle$ must be greater or equal (componentwise) than one tuple among $\langle 2, 0, 1\rangle$, $\langle 1, 1, 0\rangle$, and $\langle 0, 2, 0\rangle$. Thus, for the given ordering $\xi$ we get the three (incomparable) gap-order constraints $x <_2 u <_0 z <_1 v = t$, $x <_1 u <_1 z <_0 v = t$, and $x <_0 u <_2 z <_0 v = t$ as a representation of all linearizations induced by $\xi$ on $\varphi$. To compute all linearizations of $\varphi$, we should perform the aforementioned steps for every $(= \cup <)$-total ordering of variables in $V$ consistent with $\varphi$. As a final result we get a finite set $Lin(\varphi, V)$ of gap-order constraints. In [10] we formally prove that $Sol(\varphi) = Sol(Lin(\varphi, V))$.

**Encoding into Sets of Strings.** We are ready now to formally define the string representation of constrained configurations. First of all, given $M = (\mathcal{M} : \varphi)$ with variables in $V$, we define

$$Lin(M) = \{(\mathcal{M} : \psi) \mid \psi \in Lin(\varphi, V)\}$$

The elements of $Lin(M)$ are called linearized constrained configurations. This definition is extended to a finite set $S = \{M_1, \ldots, M_n\}$ as $Lin(S) = Lin(M_1) \cup \ldots \cup Lin(M_n)$. From the properties of the linearization algorithm, we have that $Lin(S)$ is a finite set and $[\![Lin(S)]\!] = [\![S]\!]$. Now let $M = (\mathcal{M} : \varphi)$ be a linearized constrained configuration with variables in $V$ and $\langle C_1, \ldots, C_r\rangle$ be the equivalence classes induced by the equalities in $\varphi$ on $V$. Since $\varphi$ is linearized, we have that (for an adequate choice of indexes) $\varphi$ induces the following order on the equivalence classes: $C_1 <_{c_1} \ldots <_{c_{r-1}} C_r$.

Now, for $i : 1, \ldots, r$ let $\mathcal{M}_i$ be the maximal *multiset* of predicate symbols $\{p_1, \ldots, p_m\}$ such that $p_j(x)$ occurs in $\mathcal{M}$ and $x \in C_i$ for $j = 1, \ldots, m$.

Furthermore, given the string constructor $\cdot$, let $\square_0$ be the empty string, and let $\square_{c+1}$ be the string $\{\square\} \cdot \square_c$. The string $Str(M)$ associated to $M$ is defined as

$$Str(M) \doteq \mathcal{M}_1 \cdot \square_{c_1} \cdot \mathcal{M}_2 \cdot \square_{c_2} \cdot \ldots \cdot \square_{c_{k-2}} \cdot \mathcal{M}_{k-1} \cdot \square_{c_{k-1}} \cdot \mathcal{M}_k$$

Since the denotation of constrained configurations do not change under variable renaming, $Str(M)$ retains all information we need from $M$. Indeed, if $q, r$ are in $\mathcal{M}_i$, then $q(x), r(y)$ occurs in $\mathcal{M}$ and $x = y$ follows from $\varphi$, if $q \in \mathcal{M}_i$ and $r \in \mathcal{M}_j$ and $\mathcal{M}_i$ and $\mathcal{M}_j$ are separated by $k$-occurrences of $\{\square\}$, then $q(x), r(y)$ occurs in $\mathcal{M}$ and $x <_k y$ follows from $\varphi$ for two variables $x, y \in \mathcal{V}$.

**Definition of Entailment.** In the previous section we have shown that a monadic constrained configuration can be represent as a **finite set** of **strings** over **multisets** defined over the **finite alphabet** $\mathcal{P} \cup \{\square\}$. The entailment relation for monadic constrained configurations is defined on such an alternative representation by taking the **pointwise** extension of a **string embedding** relation compatible with **multiset inclusion**.

Specifically, let us first introduce the string embedding relation. Given two *strings* $S = N_1 \cdot N_2 \cdot \ldots \cdot N_r$ and $T = M_1 \cdot M_2 \ldots \cdot M_k$, $StrEnt(S, T)$ holds if and only if there exists an injective and strictly monotone mapping $h$ from $1, \ldots, k$ (the indexes of $T$) to $1, \ldots, r$ (the indexes of $S$) such that $M_i \preccurlyeq N_{h(i)}$ for $i : 1, \ldots, k$. A mapping $h$ is strictly monotone if $i < j$ implies $h(i) < h(j)$. As an example, given $S = \{p,p,r\} \cdot \{\square\} \cdot \{t,t\} \cdot \{\square\} \cdot \{\square\} \cdot \{q,q\}$ and $T = \{p,p\} \cdot \{\square\} \cdot \{\square\} \cdot \{\square\} \cdot \{\square\} \cdot \{q\}$. we have that $StrEnt(S, T)$ holds since we can define $h$ so as to map $\{p,p\}$ to $\{p,p,r\}$ ($\{p,p\} \preccurlyeq \{p,p,r\}$), the first, second and third occurrence of $\{\square\}$ in $T$ to the first, second and third occurrence in $S$, and $\{q\}$ to $\{q,q\}$.

The string embedding can be used to define the entailment relation $CEnt$. However, since the string representation only works for linearized constrained configurations, we have to extend $StrEnt$ to sets as follows.

**Definition 4.** Given constrained configuration $N$ and $M$,

$$CEnt(N, M) \equiv \forall N' \in Lin(N). \exists M' \in Lin(M). StrEnt(Str(N), Str(M))$$

Fig. 1 illustrates the structure of the ordering. The following property holds.

**Lemma 3.** $CEnt$ is an entailment.

*Proof.* We show that $StrEnt(Str(N), Str(M))$ implies $[\![N]\!] \subseteq [\![M]\!]$. We first notice that, since $\square \notin \mathcal{P}$, then $\{\square\}$ can only be mapped to another singleton $\{\square\}$. Now, given $Str(M) = \mathcal{M}_1 \cdot \square_{c_1} \cdot \ldots \square_{c_{k-1}} \cdot \mathcal{M}_k$ and $Str(N) = \mathcal{N}_1 \cdot \square_{d_1} \cdot \ldots \square_{c_{n-1}} \cdot \mathcal{N}_n$, if $StrEnt(Str(N), Str(M))$ holds then for every $\mathcal{M}_i$ there exists a distinct $\mathcal{N}_{h(i)}$ such that $\mathcal{M}_i \preccurlyeq \mathcal{N}_{h(i)}$. Furthermore, let $m > 0$, then $\square_{c_i}$ must be mapped to $c_i$ different occurrences of $\{\square\}$ between $\mathcal{N}_{h(i)}$ and $\mathcal{N}_{h(i)+m}$. This

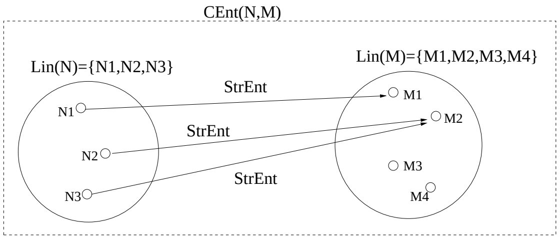


**Fig. 1.** Structure of the *CEnt* ordering.

implies that (1) $d_{h(i)} + d_{h(i)+1} + \ldots + d_{h(i)+m} \geq c_i$.
Thus, let $\mathcal{C} \in [\![N]\!]$ and $N = (\mathcal{N} : \varphi)$. Then, $\mathcal{C} = \mathcal{C}_1 \oplus \ldots \oplus \mathcal{C}_n \oplus \mathcal{D}$ where $\mathcal{C}_i = \sigma(\mathcal{A}_i)$ where $\mathcal{A}_i$ is the multiset of atoms in $\mathcal{N}$ associated to the predicates in $\mathcal{N}_i$ for $i : 1, \ldots, n$ and $\sigma \in Sol(\varphi)$. Then, we have that if $p(v), p(w) \in \mathcal{C}_i$ then $v = w$, and if $p(v) \in \mathcal{C}_i$ and $p(w) \in \mathcal{C}_{i+1}$ then $v <_{c_i} w$.
Now, let $\mathcal{B}_i$ be the multiset of atoms in $\mathcal{M}$ associated to the predicates in $\mathcal{M}_i$ for $i : 1, \ldots, k$. We define the evaluation $\gamma$ for the variables in $\mathcal{M}$ as follows: $\sigma$ maps all variables in $\mathcal{B}_i$ to the same value $v$ such that $p(v) \in \mathcal{C}_{h(i)}$. Now, if $x = y$ is in $\psi$ and $p(x), q(y)$ occur in $\mathcal{M}$ for $x \neq y$, then they must be in the same multiset $\mathcal{B}_i$, and thus $\gamma(x) = \gamma(y)$. Furthermore, if $x <_{d_i} y$ is in $\psi$, then $p(x) \in \mathcal{B}_i$ and $q(y) \in \mathcal{B}_{i+1}$ (we work on linearized const. conf.). Thus, $\gamma(p(x)) = p(v) \in \mathcal{A}_{h(i)}$ and $\gamma(p(x)) = p(w) \in \mathcal{A}_{h(i+1)}$. Furthermore, since $\sigma$ is a solution for $\varphi$, we have that $v <_{d_{h(i)} + d_{h(i)+1} + \ldots + d_{h(i)+m}} w$ must hold. From (1), it follows that $v <_{c_i} w$ holds, too. Thus, $\gamma$ satisfies all atoms in $\psi$, i.e., $\gamma \in Sol(\psi)$.
Now let $\mathcal{E} = \mathcal{E}_1 \oplus \ldots \oplus \mathcal{E}_k$ where $\mathcal{E}_i = \gamma(\mathcal{B}_i)$ for $i : 1, \ldots, k$. Then, $\mathcal{E}_i \subseteq \mathcal{C}_i$ for $i : 1, \ldots, k$. Thus, $\mathcal{E} \preccurlyeq \mathcal{C}$, i.e., $\mathcal{C} \in [\![M]\!]$. □

### 4.3 Termination

To prove the termination of our procedure we need the following lemma.

**Proposition 3.** *CEnt* is a well quasi ordering.

*Proof.* From Prop 5 (p. 2-3), the multiset inclusion relation $\preccurlyeq$ is a better quasi ordering. Since *StrEnt* is built over elements ordered with the better quasi ordering $\preccurlyeq$ and from Prop 5 (p. 4), *StrEnt* is still a better quasi ordering. Now, from Prop 5 (p. 5), the pointwise ordering of finite sets built on a better quasi ordering is still a better quasi ordering. Thus, *CEnt* is a better quasi ordering, and thus, from Prop 5 (p. 1), a well quasi ordering. □

We are ready now to prove the following result.

**Theorem 4.** Control state reachability is decidable for monadic MSR($\mathcal{G}$).

*Proof.* Given $\mathcal{P} = \{p_1, \ldots, p_k\}$ and natural numbers $n_1, \ldots, n_k$, in order to decide if there exists an execution run that starting from $\mathcal{M}_0$ leads to a configuration $\mathcal{M}$ such that $Occ_{p_i} \geq n_i$ for $i : 1, \ldots, k$, we apply the symbolic backward reachability algorithm with entailment $CEnt$ on the input

$$\mathcal{M} \doteq p_1(x_{11}) \mid \ldots \mid p_1(x_{1n_1}) \mid \ldots \mid p_k(x_{n1}) \mid \ldots \mid p_k(x_{nn_k}) \; : \; true$$

where $x_{ij} \neq x_{lm}$ for $i \neq l$ or $j \neq m$; $[\![\mathcal{M}]\!]$ coincides with the target set of configurations $\mathcal{T}$ of Section 3.
From Prop. 3 and Lemma 2, the algorithm always terminates and returns a finite set of constrained configurations $R$. Notice that the string representation is only needed for the entailment test; the predecessor computation is performed at the level of constrained configurations. Furthermore, from the general property of $SPre$ proved in [9], namely $[\![SPre(S)]\!] = Pre([\![S]\!])$, we have that $[\![R]\!] = Pre^*(\mathcal{T})$. To check if $\mathcal{M}_0$ belongs to $[\![R]\!]$, we simply have to find $(\mathcal{M} : \varphi) \in R$ such that $\mathcal{M}_0$ unifies (modulo AC) with $\mathcal{M}$ with unifier $\gamma$ and $\varphi \wedge \gamma$ is a satisfiable constraint. We conclude by noting that both problems are decidable. □

As a final result, we study the expressiveness of the symbolic data structure:

**Proposition 4.** It is not possible to extend symbolic backward reachability for monadic MSR($\mathcal{G}$) to an assertional language $\mathcal{L}$ in which we have asssertions that represents the (set) difference of denotations of formulas in $\mathcal{L}$.

*Proof.* We prove that if symbolic backward reachability based on language $\mathcal{L}$ always terminates then we can solve control state reachability for 2CMs. We consider again the weak encoding of 2CMs of Def. 2. Suppose that we can perform a complete backward search starting from $\{l_p(x)\}$, i.e., the symbolic representation of all configurations in which the control location $l_p$ occurs. Then, we can restrict the search to good simulations, by pruning bad simulations using the formulas in $\mathcal{L}$ that represent the difference between the denotation of a generated formula $F$ and of the formulas $l_p(x) \mid c_i(y) \; : \; x > y$ for all $p$ and $i$. This contradicts the fact that control state reachability is undecidable for 2CMs. □

## 5 Conclusions and Related Work

Our results identify a class of infinite state systems (Monadic MSR($\mathcal{G}$)) whose expressiveness lies between Petri Nets and Turing machines. We are currently working on an implementation of symbolic backward reachability combining the constraint-based and the string-based representations for monadic MSR($\mathcal{G}$). Such a procedure could find interesting applications to verify safety properties of abstract models of concurrent programs and protocols with integer data path and arithmetic operations (as parametric versions of the mutual exclusion protocols and of the Circular Queue example of [5]). The application of constraint deductive databases techniques to verification of concurrent systems dates back to the work of Fribourg [16, 15, 14] in which the bottom-up evaluation of Datalog programs with gap-order constraints is applied to compute the set of reachable states of systems with integer counters. In this setting CLP programs express the update of the global state of a system with a finite number of components. For instance, the CLP clause $p(x', y') \subset p(x, y) \wedge x + 1 < x' \wedge y + 1 < y'$ (to be read from right to left) is used to represent the update of two counters. In our computational model we refine this idea as well as the CLP-based approach to model checking of [12] by using multiset rewriting and constraints to locally specify updates to the global state. For instance, in MSR($\mathcal{G}$) the previous rule can be written as $p(x) \rightarrow p(x') : x + 1 < x'$ (to be read from left to right) and will work for any number of counters put in parallel in the initial configuration $p(0) \mid p(0) \mid \ldots$). This way, we can reason about systems with an arbitrary number of components each one with integer data paths. In [23] Toman combined these techniques with abstract interpretation for program analysis. The MSR($\mathcal{C}$) language together with the notion of constrained configuration and the corresponding symbolic operations have been introduced in [8]. The resulting framework has been applied to different types of concurrent systems. In [3, 4] we used an instance of the MSR($\mathcal{C}$) scheme with linear arithmetic constraints to verify properties of parameterized versions of mutual exclusion and coherence protocols. In [11] we used a constraint system combining arithmetics and equalities over first order terms to verify secrecy and authentication properties of time-sensitive cryptographic protocols with an unbounded number of sessions. In the extended paper [9] (a generalization of the specific applications [3, 4, 11]) we present the proof of termination of a symbolic backward reachability algorithm for a monadic fragment with arithmetic constraints in which atoms have the form $x < y$ and $x = y$ interpreted over **rational or reals** (over integers they are not closed under variable elimination). In this paper we study a monadic fragment with a larger fragment of linear constraints (gap-order), and we interpret them in the (usually) more difficult domain of **integers**. Furthermore, in our previous work we only focused on symbolic verification algorithms, whereas in the present paper we also compare the expressiveness of monadic MSR($\mathcal{G}$) with that of Petri Nets and show that they are not equivalent, thus obtaining a hierarchy of Petri Nets extensions. In [17, 18] Lazic et al. study the properties of unbounded arrays with $x < y$ and $x = y$ comparisons of integer-value index variables using a string encoding similar to that presented in [9]. Specifically, they use a special "invisible" predicate symbol $\epsilon$ so as to represent array formulas with existential quantified constraints like $\exists y. x < y < z$ (in which $x$ and $y$ must have a gap of one unit) as the multiset $a(x) \mid \epsilon(y) \mid b(z)$. These formulas are needed to symbolically compute preconditions. In this setting existentially quantified variables are never eliminated. This might lead to huge formulas during a fixpoint analysis. In our work we directly work on gap-order constraints, thus enlarging the class of integer systems we can deal with, so as to use Revesz's variable elimination procedure inside the symbolic predecessor operator of [9]. This way, we can work on a compact symbolic data structure, e.g., useful to apply entailment relations defined over constraint operations.Our proof method is based on the framework proposed in [1, 13] and extended to deal with better-quasi orderings in [2, 20]. This method has been applied to Petri Nets, Lossy FIFO systems and

Timed Petri Nets (see [20]). Timed systems enjoy special properties that can be exploited for obtaining symbolic representations of sets of states (see e.g. [20]), and (un)decidability results (see e.g. the weak encoding of 2CMs based on time transitions of [7]). As opposed in the present paper we study decision problems for untimed extensions of Petri Nets.

*Acknowledgments.* We thank R. Lazic and P. Abdulla for useful discussions/links to related work.

## References

1. P. A. Abdulla, K. Cerāns, B. Jonsson, and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. LICS'96: 313–321, 1996.
2. P. A. Abdulla and A. Nylén. Better is Better than Well. LICS'00: 132–140, 2000.
3. M. Bozzano and G. Delzanno. Beyond Parameterized Verification. TACAS '02: 221-235, 2002.
4. M. Bozzano and G. Delzanno. Automatic Verification of Invalidation-based Protocols. CAV '02: 295-308, 2002.
5. T. Bultan, R. Gerber, W. Pugh. Model-checking concurrent systems with unbounded integer variables. TOPLAS 21(4): 747-789 (1999)
6. I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. A Meta-notation for Protocol Analysis. CSFW '99: 55–69, 1999
7. D. de Frutos Escrig, F. Cuartero Gomez, and V.V. Ruiz. On non-decidability of reachability for timed-arc Petri nets. PNPM '99, 1999.
8. G. Delzanno. An Assertional Language for Systems Parametric in Several Dimensions. ENTCS 50, 2001.
9. G. Delzanno. Constraint Multiset Rewriting. Technical report DISI available at the URL [24] (extends [8, 3, 4, 11]). Subm. for journal publication, 2002, rev. 2003.
10. G. Delzanno. On the Linearization of Gap-order Constraints. T.R. DISI, available at the URL [24], March 2005.
11. G. Delzanno and P. Ganty. Automatic Verification of Time Sensitive Cryptographic Protocols. TACAS '04: 342–356, 2004.
12. G. Delzanno and A. Podelski. Model Checking in CLP. TACAS '99: 223-239, 1999
13. A. Finkel and P. Schnoebelen. Well-Structured Transition Systems Everywhere! *TCS*, 256(1-2):63–92, 2001.
14. L. Fribourg and H. Olsen. A Decompositional Approach for Computing Least Fixpoint of Datalog Programs with Z-counters. *Constraints*, 2(3-4):305–336, 1997.
15. L. Fribourg and J. Richardson. Symbolic Verification with Gap-order Constraints. LOPSTR '96, LNCS 1207: 20-37, 1997.
16. L. Fribourg and M. Veloso Peixoto. Bottom-up Evaluation of Datalog Programs with Arithmetical Constraints. CADE '94: 311-325, 1994.
17. R. Lazic Decidability of Reachability for Polymorphic Systems with Arrays: A Complete Classification Prel. version, Infinity '04, 2004
18. R. Lazic, T. Newcomb and A. W. Roscoe Polymorphic Systems with Arrays, 2-Counter Machines and Multiset Rewriting Prel. version, Infinity '04, 2004.
19. A. Marcone. Fine and Axiomatic Analysis of the quasi-orderings on $\mathcal{P}(q)$. T.R. 17/99/RR, DIMI, Università di Udine, 1999.
20. A. Nylén. Compact Constraints for Verification of Well Quasi-Ordered Programs. PhD thesis. Uppsala University, 2003.

$$(init)\ init \to hd(h) \mid tl(t)\ :\ t = h.$$

$$(n_1)\quad \epsilon \to pd_c : true$$

$$(n_2)\quad \epsilon \to cn_c : true$$

$$(r_1)\quad pd_c \to pd_c \mid preq_c : true$$

$$(r_2)\quad cn_c \to pd_c \mid creq_c : true$$

$$(p_1)\quad unlocked \mid hd(h) \mid tl(t) \mid preq_c \to locked \mid hd(h) \mid tl(t) \mid wpreq_c :\ true$$

$$(p_1)\quad locked \mid hd(h) \mid tl(t) \mid wpreq_c \to unlocked \mid hd(h) \mid tl(t')\ :\ t + (c-1) < t'$$

$$(c_1)\quad unlocked \mid hd(h) \mid tl(t) \mid creq_c \to locked \mid hd(h) \mid tl(t') \mid wcreq_c\ :\ h + (c-1) < t$$

$$(c_1)\quad locked \mid hd(h) \mid tl(t) \mid wcreq_c \to unlocked \mid hd(h) \mid tl(t')\ :\ h + (c-1) < h'$$

**Fig. 2.** Producer and consumers with an unbounded buffer.


21. P. Revesz. Introduction to Constraint Databases, Springer, 2002.
22. P. Revesz. A Closed Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints, TCS 116(1):117-149, 1993.
23. D. Toman. Constraint Databases and Program Analysis using Abstract Interpretation. CDB '97: 246–262, 1997.
24. Home page: http://www.disi.unige.it/person/DelzannoG/papers.html


## A Example

Let us consider a system in which producers and consumers communicate via a potentially unbounded buffer. A producer $pd_c$ where $c$ is a fixed integer can put $c$ elements in the buffer, whereas a consumer $cn_c$ can remove $c$ elements from the buffer. Elements are all the same (we only need to count them). The buffer is implement using a Boolean semaphore *un/locked*, and two pointers, namely $hd(h)$ (head) and $tl(t)$ (tail), where $h$ points to the next item to be removed, and $t$ points to the first empty location. Upon reception of $c$-items coming from a consumer $cn_c$, assuming that $h + c \leq t$ we should update the pointer $h$ as $h' = h + c$. The update $h' = h + c$ cannot be expressed using gap-order constraints. Thus we can only approximate it via the gap-order constraint $h' > h + (c - 1)$ (i.e. $h' \geq h + c$). Similarly for a producer request. The buffer is locked during the processing of requests. The rules in Fig. 2 specify the resulting abstract system with any number of producers and consumers (indexed on a finite set of constants). Initially, the buffer is empty (rule $init$). Processes are injected via rules $(n_1), (n_2)$. Rule $r_1$ and $r_2$ generate new requests. Rule $p_1$ approximate the pointer update related to a produced message, whereas rules $c_1$ and $c_2$ approximate the consumer messages. Although in this abstract model $h \leq t$ is no more an invariant, it always holds when the buffer is locked by a consumer. To prove this invariant using symbolic backward reachability we should start from the constrained configuration expressing bad states: $M = (wreq_c \mid hd(h) \mid tl(t)\ :\ h > t)$ and check that $init$ cannot occur in the set of predecessors, i.e., $init \not\in [\![SPre^*(\{M\})]\!]$.

## B 2CMs in Dyadic MSR($\mathcal{G}$)

We first show that 2CMs can be fully simulated by dyadic MSR($\mathcal{G}$). Following an encoding used for MSR with linear constraints in [9], every counter can be represented via a doubly linked list implemented via a multiset of dyadic atoms built on the binary predicate $p$ and via a sentinel cell implemented via the dyadic predicate $s_i$. The sentinel points to the head and tail of the list. The formula $s_i(0,0)$ represents $c_i = 0$. We represent $c_i = k$ with the multiset $s_i(0, v_k) \mid c_i(0, v_1) \mid \ldots \mid c_i(v_{k-1}, v_k)$, where $0 < v_1 < \ldots < v_{k-1} < v_k$ for $v_i \in \mathbb{Z}$. The current control location $l_p$ is stored as an atomic formula $l_p$.
The instruction $l_p$: $INC_i$; *goto* $l_q$ is simulated by the mutually exclusive rules

$$(i_1)\ l_p \mid s_i(x,x) \ \rightarrow\ l_q \mid s_i(x,z) \mid c_i(x,z)\ : x < z$$
$$(i_2)\ l_p \mid s_i(x,y) \mid c_i(z,y)\ \rightarrow\ l_q \mid s_i(x,u) \mid c_i(z,y) \mid c_i(y,u)\ :\ y < u$$

Rule $i_1$ applies when $c_i = 0$; $i_2$ applies when $c_i > 0$. The instruction $l_p$: *if* $c_i > 0$ *then* $DEC_1$; *goto* $l_q$ *else goto* $l_r$ is simulated by the mutually exclusive rules

$$(d_1)\ l_p \mid s_i(x,y) \mid c_i(x,y)\ \rightarrow\ l_q \mid s_i(x,x)\ :\ true$$
$$(d_2)\ l_p \mid s_i(x,y) \mid c_i(z,u) \mid c_i(u,y)\ \rightarrow\ l_q \mid s_i(x,y) \mid c_i(z,y)\ :\ true$$
$$(d_3)\ l_p \mid s_i(x,x)\ \rightarrow\ l_r \mid s_i(x,x) : true$$

Rule $d_1$ applies when $c_i = 1$, $d_2$ applies when $c_i > 1$, and $d_3$ when $c_i = 0$. Every step of a 2CM $M$ is simulated by a step of $S_M$. Reachability of the control location $l_f$ in $M$ is equivalent to reachability of a configuration of $S_M$ with at least one occurrence of $l_f$.

## C The Symbolic Predecessor Operator $SPre$

Given two atoms $A = p(x_1, \ldots, x_k)$ and $B = q(y_1, \ldots, y_l)$, we use $A = B$ as a shorthand for the constraint $x_1 = y_1 \wedge \ldots \wedge x_k = y_k$, provided $p = q$ and $k = l$. Given $M = (A_1 \mid \ldots \mid A_n : \varphi)$, and $N = (B_1 \mid \ldots \mid B_m : \psi)$, we say that $\mu$ is a *unifier* for $M$ and $N$, written $M =_\mu N$ if and only if $m = n$ and there exists a permutation $j_1, \ldots, j_n$ of $1, \ldots, n$ such that the constraint $\mu \equiv \varphi \wedge \psi \wedge \bigwedge_{i=1}^{n} A_i = B_{j_i}$ is *satisfiable*. As an example, given $M = (p(x) \mid q(y) \mid r(z) :\ x < y \wedge y = z)$ and $N = (q(u) \mid p(v) \mid r(w) :\ v <_1 u)$. then, $M =_\mu N$ where $\mu \equiv (x = v, u = y, z = w, x < y, y = z, v <_1 u) \equiv (x = v, v <_1 u, u = y, y = z, z = w)$. Now, let $M = (\mathcal{M} : \varphi)$ be a constrained configuration and $R = (\mathcal{A} \longrightarrow \mathcal{B} : \psi)$ be a rule. The symbolic predecessor operator $SPre$ of [9] computes a set of constrained configurations $SPre(M)$ such that $[\![SPre(M)]\!] = Pre([\![M]\!])$. Since denotations are upward closed sets, $SPre$ applies a weak form of (backward) rewriting. As a simple example, let $M = (a \mid b\ :\ true)$ and $R$ be the rule $d \rightarrow b \mid c$. Although the right-hand side of $R$ does not match $M$, the configuration $a \mid b \mid c$ obtained by adding $c$ to $M$ (i.e. it belongs to $[\![M]\!]$) is a successor of $a \mid d$, i.e., $a \mid d \in Pre([\![M]\!])$. The same result can be obtained by removing the subconfiguration $b$ from $M$

and by adding the remaining multiset to $a$. The symbolic predecessor operator extends this idea to configurations and constraints as follows:

$$
\begin{array}{rl}
SPre(\mathcal{M}:\varphi) = \{\ N \mid & \text{there exist } R = (\mathcal{A} \longrightarrow \mathcal{B} : \psi) \in \mathcal{R}, \\
& \text{three multisets } \mathcal{M}',\ \mathcal{B}',\ \mathcal{N}, \text{ and a constraint } \mu \text{ s.t.} \\
& \mathcal{M}' \preccurlyeq \mathcal{M},\ \ \mathcal{B}' \preccurlyeq \mathcal{B},\ \ (\mathcal{M}':\varphi)\ =_\mu\ (\mathcal{B}':\psi), \\
& \mathcal{N} = \mathcal{M} \ominus \mathcal{M}',\ N = (\mathcal{A} \oplus \mathcal{N}\ :\ \mu'),\ \mu' \equiv \exists x_1.\ldots x_k.\mu \\
& \text{and } x_1,\ldots,x_k \text{ are the variables of } \mu \text{ not in } \mathcal{A} \oplus \mathcal{N}\}
\end{array}
$$

The operator $SPre$ is extended to a finite set of const. configurations $S$ as follows $SPre(\{M_1,\ldots,M_n\}) = SPre(M_1) \cup \ldots SPre(M_n)$. As an example, consider the rule $p(x,y) \longrightarrow q(x',y') : y <_1 x, x' = x, y' = y$, and the constrained configuration $M = (q(u,v) : v <_2 u)$. Then, $SPre(M)$ contains $p(x,y) : y <_2 x$ (obtained by unification of $q(u,v)$ and $q(x',y')$ and by eliminating variables $x',y',v,u$ from $v <_2 u, y <_1 x, x' = x, y' = y, x' = u, y' = v$) as well as $p(x,y) \mid q(u,v) : y <_1 x, v <_2 u$, e.g., $p(4,1) \mid q(6,1)$ rewrites into $q(4,1) \mid q(6,1) \in [\![M]\!]$. The latter constrained configuration is obtained by setting $\mathcal{M}' = \mathcal{B}' = \epsilon$ (the empty multiset) when applying $SPre$ to $M$. In [9] we have proved that for any constrained configuration $M$ $[\![SPre(M)]\!] = Pre([\![M]\!])$, and for any set of constrained configurations $S$, $[\![SPre(S)]\!] = Pre([\![S]\!])$. Since gap-order constraints are closed under variable elimination, it is immediate to check that, constrained configurations with gap-order constraints are closed under applications of $SPre$.

## D The Theory of Well and Better Quasi-Orderings

In the following we summarize some basic definitions and results on the theory of *well quasi-orderings* taken from [2, 20]. Following the notation used in the literature the relation $M \sqsubseteq N$ must be read as "$M$ has more information than $N$" or "the denotation of $N$ is contained in that of $M$", i.e., $N$ entails $M$. In the present paper we use the reverse notation $Ent(N,M)$ because seems more natural when read in terms of containment of denotations. A *quasi-order* (preorder) $\sqsubseteq$ on a set $A$ is a binary relation over $A$ which is reflexive and transitive. In the following it will be denoted $(A, \sqsubseteq)$. A quasi order $(A, \sqsubseteq)$ is a *well quasi ordering* (wqo) if each infinite sequence $a_0\ a_1\ a_2\ \ldots$ of elements in $A$ is *good*. A sequence is $a_0\ a_1\ a_2\ \ldots$ is good if there exist indexes $i < j$ such that $a_i \sqsubseteq a_j$, i.e., we reach a point in which $a_j$ adds no new information (in the present paper we use $Ent(a_j, a_i)$ instead of $a_i \sqsubseteq a_j$). Wqo's are closed under several operations (see Prop. 5 below) like **bag/set construction** with **bag/set embedding**, and **strings** with **string embedding**. However, they are not closed under the **pointwise ordering** of **finite sets** $S \sqsubseteq^S T$ (the set $S$ has more information than $T$) defined as $S \sqsubseteq^S T$ iff $\forall N \in T.\exists M \in S.M \sqsubseteq S$. According to the "reverse" notation used for $Ent$, in the present paper we use $Ent^S(T,S)$ to denote the extension of $Ent$ to finite sets. As shown in Section 3, this is a natural operation to extend an entailment on elements to a set of elements. Rado has given a counterexample to this closure property (see [2]). Notice that **set embedding** (under which wqo's are closed), defined as $S \sqsubseteq^E T$ iff $\forall M \in S.\exists N \in T.M \sqsubseteq N$, is not adequate for extending to sets a given entailment relation $\sqsubseteq$. Indeed, $S \sqsubseteq^E T$ does not guarantee that we compare all elements in $T$ with elements in $S$. The notion of *better quasi ordering* (bqo) is stronger than wqo and it has been studied to isolate the subclass of wqo's that do not suffer from Rado's counterexample. To obtain this property, instead of reasoning over sequences, i.e., mappings $f : \mathbb{Z} \to A$ in which indexes are ordered with respect to $<$, we reason over an abstract structure $\langle \beta, \lhd \rangle$ called barrier equipped with the generalization $\lhd$ of $<$. A quasi-order $(A, \sqsubseteq)$ is a *better quasi ordering* if for any barrier $\langle \beta, \lhd \rangle$ and any function $f : \beta \to A$, $f$ is good, i.e., $\exists s \lhd s'.\ f(s) \sqsubseteq f(s')$. The definition of barrier and of the order $\lhd$ are fairly complicate to explain. The interest reader can found them in [19]. For our purposes, it is enough to recall the closure properties of a bqo. In what follows, let $\widehat{n}$ denote the set $\{1, \ldots, n\}$. We have the following results, according to which a hierarchy of wqos and bqos can be built starting from known ones.

**Proposition 5.**

1. *Each bqo is a wqo;*
2. *If $A$ is a finite set, then $(A, =)$ is a bqo.*
3. *Let $(A, \sqsubseteq)$ be a bqo, and let $A^B$ denote the set of finite bags (i.e. multisets) over A. Then, $(A^B, \sqsubseteq^B)$ is a bqo, where $\sqsubseteq^B$ is the quasi-order on $A^B$ defined as follows: $w \sqsubseteq^B w'$ iff there exists an injection $h : \widehat{|w|} \to \widehat{|w'|}$ such that $w(j) \sqsubseteq w'(h(j))$ for $1 \leq j \leq |w|$ (as in the definition of multiset inclusion in the present paper). This property holds for wqo's.*
4. *Let $(A, \sqsubseteq)$ be a bqo, and let $A^*$ denote the set of finite strings over $A$. Then, $(A^*, \sqsubseteq^*)$ is a bqo, where $\sqsubseteq^*$ is the quasi-order on $A^*$ defined in the following way: $w \sqsubseteq^* w'$ iff there exists a strictly monotone (meaning that $j_1 < j_2$ iff $h(j_1) < h(j_2)$) injection $h : \widehat{|w|} \to \widehat{|w'|}$ such that $w(j) \sqsubseteq w'(h(j))$ for $1 \leq j \leq |w|$ (as in the definition of Str in the present paper). This property holds wqo's.*
5. *Let $(A, \sqsubseteq)$ be a bqo. Then, $(\mathcal{P}(A), \sqsubseteq^S)$ is a bqo, where $\mathcal{P}(A)$ is the (finite) power set of $A$ and $\sqsubseteq^S$ is the* pointwise *ordering defined as follows: given $X, Y \subseteq A$, $X \sqsubseteq^S Y$ (in the paper we use the reverse notation $Ent^S(Y, X)$) iff for every $y \in Y$ there exists $x \in X$ s.t. $x \sqsubseteq y$ (in the paper we use the reverse notation $Ent(y, x)$). This property* **does not** *hold for a wqo.*