# Consistency of Constraint Networks Induced by Automaton-Based Constraint Specifications

María Andreína Francisco Rodríguez, Pierre Flener, and Justin Pearson

Uppsala University, Department of Information Technology, Uppsala, Sweden
MariaAndreina.Francisco_Rodriguez.3450@student.uu.se,
Pierre.Flener@it.uu.se, Justin.Pearson@it.uu.se

**Abstract.** We discuss the consistency of constraints for which the set of solutions can be recognised by an automaton. Such an automaton induces a decomposition of the constraint into a conjunction of constraints. The so far most general result is that if the constraint hypergraph of such a decomposition is Berge-acyclic, then the decomposition provides hyper-arc consistency. We focus on constraint networks that have $\alpha$-acyclic or centred-cyclic hypergraph representations and show the necessary conditions to achieve hyper-arc consistency in these cases.

**Keywords:** Acyclicity, hypergraph, automaton.

## 1 Introduction

Global constraints are an important component in many modern constraint solvers. A global constraint does two things: from the modelling perspective, it allows a modeller to express commonly occurring combinatorial structures; from the solving perspective, it comes with a filtering algorithm that removes impossible domain values during search. There are global constraints for many combinatorial structures, such as scheduling [1, 3], packing [14], staff scheduling [13], and so on [8].

Although modern constraint solvers have many global constraints, often a constraint that one is looking for is not there. In the past, the choices where either to reformulate the problem or to write one's own filtering algorithm.

In [7], a framework is given where a global constraint can be specified in a relatively simple and high-level way by a (deterministic or non-deterministic) finite automaton. The idea is to describe *what* it means for the constraint to be satisfied in terms of the accepting paths of the automaton. Based on the automaton, the framework decomposes the specified new global constraint into a conjunction of already implemented (global) constraints. These constraints collectively give the semantics of the specified global constraint and provide the filtering.

It is so far known [6, 7] that if the constraint graph of a decomposition (induced by an automaton) is Berge-acyclic [9], then the decomposition automatically provides hyper-arc consistency, that is the decomposition achieves all the

filtering that is possible. Beside Berge-acyclicity, another ten patterns of constraint hypergraph structure are identified in the current on-line version of the *Global Constraint Catalogue* [8],[1] but little is known about the filtering strength of the (automaton-induced) decompositions that satisfy these structures.

In this paper, we show how an $\alpha$-acyclic constraint hypergraph (see Section 3) can be modified to provide hyper-arc consistency. Moreover, we show that by adding implied constraints the so-called centred-cyclic networks of [8] can also be modified to provide hyper-arc consistency (see Section 4). This covers five of the ten open hypergraph patterns (namely $\alpha$-acyclic(2), $\alpha$-acyclic(3), centred-cyclic(1), centred-cyclic(2), and centred-cyclic(3)) and almost doubles the number of automaton-induced decompositions in the current on-line version of the *Global Constraint Catalogue* [8] that are now known to provide hyper-arc consistency. The *Global Constraint Catalogue* contains at the moment 15 $\alpha$-acyclic and 19 centred-cyclic constraints.

It was already observed in [6] that an $\alpha$-acyclic constraint hypergraph can be made hyper-arc consistent by making all the constraints pairwise consistent (see Definition 6), but no algorithm was given. Here we show the connection (see Theorem 1) between achieving pairwise consistency and doing a reachability analysis on an automaton.

There is also a large body of related work (e.g., [10–12, 20]) on decomposing global constraints to achieve hyper-arc consistency. The work in this paper can be seen as a more systematic approach to providing hyper-arc consistency via decompositions.

## 2 Background: The *automaton* Constraint

The $automaton(A, V)$ constraint [7] holds if the constraint described by the automaton $A$ holds for the sequence of decision variables $V$, that is if $A$ accepts the sequence of values of $V$. We define the *automaton* constraint in three stages: first its particular case that is also known as the *regular* constraint [19], and then two orthogonal extensions, namely predicate automata and counter automata.

### 2.1 Modelling Constraints with Automata

**Definition 1.** *A* deterministic finite-state automaton (DFA) *is a tuple* $\langle Q, \Sigma, S, F, \delta \rangle$ *where* $Q$ *is the set of states;* $\Sigma$ *is the alphabet;* $S$ *is a subset of* $Q$ *denoting the start states;* $F$ *is a subset of* $Q$ *denoting the accepting states; and* $\delta$ *is a function from* $Q \times \Sigma$ *to* $Q$ *denoting the transition function. If* $\delta(\rho, \sigma) = \rho'$*, then we say that there is a transition from state* $\rho$ *to state* $\rho'$ *that* consumes *alphabet symbol* $\sigma$*; this is often written as:*

$$\rho \xrightarrow{\sigma} \rho'$$

[1] See `http://www.emn.fr/z-info/sdemasse/gccat/sec3.6.5.html`

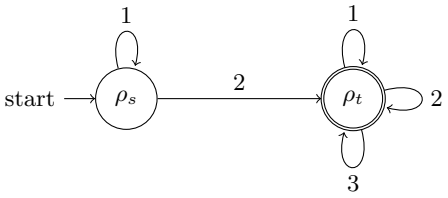


**Fig. 1.** A deterministic finite state automaton for the regular expression $1^*2(1+2+3)^*$

*A sequence* $\sigma_1\sigma_2\cdots\sigma_{n-1}\sigma_n$ *of alphabet symbols is* accepted *by the automaton if there is a chain of transitions*

$$\rho_0 \xrightarrow{\sigma_1} \rho_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_{n-1}} \rho_{n-1} \xrightarrow{\sigma_n} \rho_n$$

*such that* $\rho_0 \in S$ *and* $\rho_n \in F$.

One often uses pictures to define finite-state automata. For example, in Figure 1, we define an automaton with two states, $Q = \{\rho_s, \rho_t\}$, represented by circles, and an alphabet of three symbols, $\Sigma = \{1, 2, 3\}$, on the transitions. A start state is indicated by an arrow from the 'start' symbol, and an accepting state is represented by a double circle. The transition function is represented by the labelled arrows, that is $\delta(\rho, \sigma) = \rho'$ if there is an arrow from $\rho$ to $\rho'$ labelled with $\sigma$. For each state, there is one outgoing arrow per alphabet symbol; any missing arrow is assumed to go to an implicit non-accepting state, on which there is a self-looping arrow for every symbol of the alphabet, so that no accepting state is reachable from that state. For example, in Figure 1, the missing transition from state $\rho_s$ on symbol 3 goes to such an implicit non-accepting state.

The symbol sequences accepted by an automaton form a regular language. Hence any constraint (on a sequence of decision variables) whose extensional definition forms a regular language can be described by an automaton. In fact, any constraint on a finite sequence of decision variables that range over finite domains can be described by an automaton, since every finite language is a regular language. For instance, the automaton in Figure 1 accepts the language of the regular expression $1^*2(1 + 2 + 3)^*$.

The *automaton* constraint discussed so far can be implemented either via a specialised propagator [19], or via decomposition into a conjunction of constraints [7]. For a given automaton, define a new constraint $T(\rho, \rho', \sigma)$ extensionally by the following set:

$$\{\langle \rho, \rho', \sigma \rangle \mid \rho \xrightarrow{\sigma} \rho'\} \tag{1}$$

That is, $T(\rho, \rho', \sigma)$ is satisfied whenever there is a transition from state $\rho$ to state $\rho'$ that consumes symbol $\sigma$. An *automaton* constraint on a sequence of $n$ decision variables, $v_1, \ldots, v_n$, is then decomposed into the following conjunction

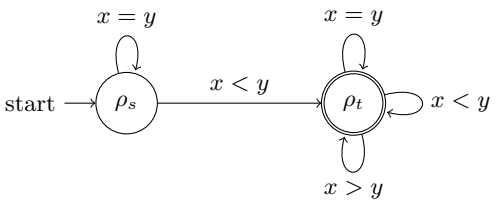


**Fig. 2.** A $k$-ary predicate automaton (with $k = 2$) for the $<_{\text{lex}}$ constraint

of $n + 2$ constraints, called the *transition constraints*:

$$q_0 \in S \wedge T(q_0, q_1, v_1) \wedge \cdots \wedge T(q_{n-1}, q_n, v_n) \wedge q_n \in F \tag{2}$$

where $q_0, q_1, \ldots, q_{n-1}, q_n$ are new decision variables, called the *state variables*, with domain $Q$. For contrast, we call $v_1, \ldots, v_n$ the *problem variables*.

The implementation of [7] actually works unchanged for non-deterministic finite-state automata, but we have elected to restrict our focus to deterministic ones, in order to ease the notation.

### 2.2 Modelling Constraints with Predicate Automata

The automata in [7] are more powerful than those in [19]: when used to describe constraints, the labels can be predicates, and all predicates must be satisfied on an accepting path.

The definition presented here is parametrised by a suitable set of predicates. Let $\mathbf{Pred}_k$ be a set of $k$-ary predicates in some suitable language. That is, a predicate takes a vector, $\mathcal{V}$, of $k$ values and it is either true or false.

**Definition 2.** *A* $k$-ary-predicate DFA *is a tuple* $\langle Q, \Sigma, \phi, S, F, \delta \rangle$*, where* $Q$*,* $\Sigma$*,* $S$*,* $F$*, and* $\delta$ *are exactly as for a deterministic finite-state automaton, and* $\phi$ *is a function from* $\Sigma$ *to* $\mathbf{Pred}_k$*. A sequence of* $k$*-ary vectors of values* $\mathcal{V}_1\mathcal{V}_2 \cdots \mathcal{V}_{n-1}\mathcal{V}_n$ *is* accepted *by the automaton if there exists a chain of transitions*

$$\rho_0 \xrightarrow{\sigma_1} \rho_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_{n-1}} \rho_{n-1} \xrightarrow{\sigma_n} \rho_n$$

*such that* $\rho_0 \in S$*,* $\rho_n \in F$*, and* $\phi(\sigma_i)(\mathcal{V}_i)$ *is true for all* $1 \leq i \leq n$*. Such a chain of transitions will often be written as*

$$\rho_0 \xrightarrow{\phi(\sigma_1)(\mathcal{V}_1)} \rho_1 \xrightarrow{\phi(\sigma_2)(\mathcal{V}_2)} \ldots \xrightarrow{\phi(\sigma_{n-1})(\mathcal{V}_{n-1})} \rho_{n-1} \xrightarrow{\phi(\sigma_n)(\mathcal{V}_n)} \rho_n$$

Again, we often define $k$-ary predicate automata by pictures. The convention is similar to normal regular automata, except that the transition labels are predicates. We assume that each distinct predicate is associated with a distinct symbol of the alphabet $\Sigma$ (as in Figure 1), and that the function $\phi$ from symbols is defined by the predicate labels in the picture. For example, in Figure 2, the function $\phi$ could be defined by lambda expressions as follows: $\phi(1) = \lambda x, y : x = y$,

$\phi(2) = \lambda x, y : x < y$, and $\phi(3) = \lambda x, y : x > y$. Consider the binary constraint ($k = 2$) that vector $\mathcal{A}$ be lexicographically less than vector $\mathcal{B}$, which is denoted by $\mathcal{A} <_{\text{lex}} \mathcal{B}$. For $\mathcal{A} = \langle 1, 2, 5, 6\rangle$ and $\mathcal{B} = \langle 1, 3, 4, 7\rangle$, the sequence $\langle 1, 1\rangle\langle 2, 3\rangle\langle 5, 4\rangle\langle 6, 7\rangle$ of binary vectors, obtained by zipping $\mathcal{A}$ and $\mathcal{B}$ together, is accepted by the binary predicate automaton in Figure 2 because the transition chain

$$\rho_s \xrightarrow{1=1} \rho_s \xrightarrow{2<3} \rho_t \xrightarrow{5>4} \rho_t \xrightarrow{6<7} \rho_t$$

ends in the accepting state $\rho_t$.

Given a predicate automaton $\langle Q, \Sigma, \phi, S, F, \delta\rangle$, the automaton $\langle Q, \Sigma, S, F, \delta\rangle$ is referred to as the *underlying automaton* of the predicate automaton. For example, the automaton in Figure 1 is the underlying automaton of the predicate automaton in Figure 2.

In [7], constraints defined by predicate automata are implemented with the help of reification. The constraint $T$ defined in (1) is used for the following transition constraints:

$$q_0 \in S \wedge T(q_0, q_1, s_1) \wedge \cdots \wedge T(q_{n-1}, q_n, s_n) \wedge q_n \in F \tag{3}$$

These transition constraints are like (2), but are expressed for *new* decision variables $s_1, \ldots, s_n$, which are connected as follows to the problem variables via the predicates and reification: given an $n$-length sequence $\mathcal{V}_1, \ldots, \mathcal{V}_n$ of $k$-ary vectors of problem variables, we add the following constraints, called the *signature constraints*:

$$\bigwedge_{\sigma \in \Sigma} (s_i = \sigma \Leftrightarrow \phi(\sigma)(\mathcal{V}_i)) \tag{4}$$

for all $1 \leq i \leq n$, where the $s_i$ are called the *signature variables*, with domain $\Sigma$. Hence $\mathbf{Pred}_k$ contains whatever can be implemented as reified constraints in the underlying constraint solver. For example, in Figure 2, the binary predicate automaton on the two vectors $\mathcal{A} = \langle a_1, \ldots, a_n\rangle$ and $\mathcal{B} = \langle b_1, \ldots, b_n\rangle$ requires the transition constraints (3) and the signature constraints

$$(s_i = 1 \Leftrightarrow a_i = b_i) \wedge (s_i = 2 \Leftrightarrow a_i < b_i) \wedge (s_i = 3 \Leftrightarrow a_i > b_i)$$

for all $1 \leq i \leq n$.

### 2.3 Modelling Constraints with Counter Automata

While the class of constraints that can be described by (predicate) automata is very large (currently, 63 of the 354 constraints of the on-line version of the *Global Constraint Catalogue* [8] are described that way), it is often the case that (predicate) automata are very large or specific to a problem instance. The second extension in [7] is the use of counters that are initialised at the start and evolve through counter-updating operations coupled to the transitions of the automaton. Such counter automata allow the capture of non-regular languages and yield (even for regular languages) automata that are much smaller if not instance-independent (and currently enable another 57 constraints of the

catalogue to be described succinctly or generically). The two extensions are orthogonal and can be composed, so we define this second extension in isolation.

Again, we give a definition that is parametric on the class of counter-updating functions. Let $\mathbf{Cupdate}_\ell$ be a set of $\ell$-ary counter-updating functions. That is, given a function $\psi \in \mathbf{Cupdate}_\ell$ and a vector of counters $\mathcal{C} \in \mathbb{N}^\ell$, we have that $\psi(\mathcal{C})$ is a new vector in $\mathbb{N}^\ell$.

**Definition 3.** *An* $\ell$-ary counter DFA *is a tuple* $\langle Q, \Sigma, \mathcal{C}_0, S, F, \delta\rangle$ *where* $Q$, $\Sigma$, $S$, *and* $F$ *are exactly as for a deterministic finite-state automaton; vector* $\mathcal{C}_0$ *has the initial values of the* $\ell$ *counters; and* $\delta$ *is a function from* $Q \times \Sigma$ *to* $Q \times \mathbf{Cupdate}_\ell$. *If* $\delta(\rho, \sigma) = (\rho', \psi)$ *and* $\psi(\mathcal{C}) = \mathcal{C}'$, *then we write*

$$(\rho, \mathcal{C}) \xrightarrow{\sigma} (\rho', \mathcal{C}')$$

*A sequence* $\sigma_1\sigma_2\cdots\sigma_{n-1}\sigma_n$ *of alphabet symbols and the* $\ell$*-ary vector of counters* $\mathcal{C}$ *are* accepted *by the automaton if there is a chain of transitions*

$$(\rho_0, \mathcal{C}_0) \xrightarrow{\sigma_1} (\rho_1, \mathcal{C}_1) \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_{n-1}} (\rho_{n-1}, \mathcal{C}_{n-1}) \xrightarrow{\sigma_n} (\rho_n, \mathcal{C})$$

*such that* $\rho_0 \in S$ *and* $\rho_n \in F$.

Unlike the counter automata in theoretical computer science (see, e.g., [18]), the counter automata here do not have access to the values of the counters during a run, but the values of the counters are updated by the transitions. This allows quite complicated constraints to be specified, especially constraints that concern the cardinality of certain sets.

In [7], counter automata are decomposed into transition constraints that are slightly extended to include information about the values of the counters. Define a new constraint $T(\rho, \rho', \mathcal{C}, \mathcal{C}', \sigma)$ extensionally by the following set:

$$\{\langle \rho, \rho', \mathcal{C}, \mathcal{C}', \sigma\rangle \mid (\rho, \mathcal{C}) \xrightarrow{\sigma} (\rho', \mathcal{C}')\}$$

An *automaton* constraint on a sequence of $n$ problem variables, $v_1, \ldots, v_n$, and a vector of $\ell$ counters, $\mathcal{C}$, is then decomposed into the following conjunction of $n + 3$ transition constraints:

$$q_0 \in S \wedge T(q_0, q_1, \mathcal{C}_0, \mathcal{C}_1, v_1) \wedge \cdots \wedge T(q_{n-1}, q_n, \mathcal{C}_{n-1}, \mathcal{C}_n, v_n) \wedge q_n \in F \wedge \mathcal{C} = \mathcal{C}_n \quad (5)$$

where $q_0, q_1, \ldots, q_{n-1}, q_n$ are state variables, with domain $Q$, while $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}, \mathcal{C}_n$ are vectors of integer decision variables, called *counter variables*, and $\mathcal{C}_0$ has the initial values of the counters.

## 3 $\alpha$-Acyclic Automata

We represent a constraint as a pair $R(S)$, where $S$ is a tuple of decision variables $\langle w_1, \ldots, w_n\rangle$ and $R$ is a set of tuples of length $n$ from some given domain. The tuple $S$ is often referred to as the *scope* of the constraint. We assume that in

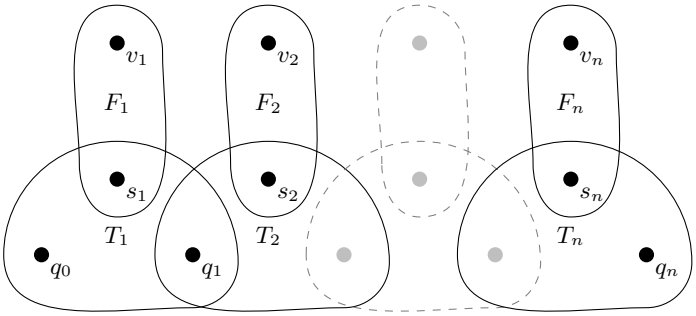


**Fig. 3.** Part of the constraint hypergraph of a predicate-automaton-induced decomposition (with $k = 1$)

all scopes all variables are distinct. A *solution* to a constraint $R(S)$ is some assignment to variables, $w_1 = d_1, \ldots, w_n = d_n$, such that the tuple $\langle d_1, \ldots, d_n \rangle$ belongs to $R$. A *constraint problem* is a conjunction of constraints. A *solution* to a constraint problem is an assignment to all its variables that satisfies all its constraints simultaneously.

Given a constraint $R(S)$ and a domain $D_i$ for each variable $w_i$, a *supporting tuple* for a value $d_k$ in the domain $D_k$ of variable $w_k$ is a tuple $\langle d_1, \ldots, d_k, \ldots, d_n \rangle$ satisfying $R$ such that for all $i \neq k$ the value $d_i$ is in $D_i$. A given set of domains is said to be *hyper-arc consistent* [2] for a constraint, if for every variable and every domain value there is a supporting tuple. Informally, every element $d_i$ of a domain $D_i$ participates in some solution to the constraint. The implementation in [7] of the transition constraints prunes the domains of the variables until hyper-arc consistency is achieved. A given set of domains is said to be *hyper-arc consistent* for a constraint problem or a set of constraints if every value in the domain of every variable participates in some solution. In general, hyper-arc consistency of the individual constraints of a problem is not sufficient to ensure hyper-arc consistency of the whole problem.

A *hypergraph* [9] is a pair $(V, E)$ where $E$ is a set of subsets of $V$. Given a conjunction of constraints $R_1(S_1) \wedge \cdots \wedge R_m(S_m)$, the set of scopes can be considered as a hypergraph $(\cup_{i=1}^{m} S_i,\ \{S_i\}_{i=1}^{m})$, where the tuples $S_i$ are considered as sets. For example, the conjunction of constraints $T(q_0, q_1, x_1) \wedge T(q_1, q_2, x_2)$ corresponds to the hypergraph $(\{q_0, q_1, q_2, x_1, x_2\}, \{\{q_0, q_1, x_1\}, \{q_1, q_2, x_2\}\})$. Such a hypergraph is referred to as a *constraint hypergraph*. In Figure 3, there is part of the constraint hypergraph of the transition constraints (3) and signature constraints (4) of a predicate-automaton-induced decomposition with $k = 1$.

It is well-known [15] that the structure of the constraint hypergraph can determine how easy it is to solve a constraint problem. Here we review some of these well-known results and apply them to automaton-induced constraint decompositions.

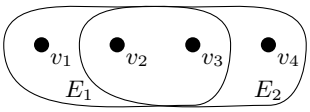


**Fig. 4.** A simple constraint hypergraph

For any hypergraph $(V, E)$, we say that it is *connected* if for every pair of vertices $s, t \in V$, there exists a sequence of edges $e_1, \ldots, e_\ell$ such that $s \in e_1$, $t \in e_\ell$, and for all $1 \leq i < \ell$ we have that $e_i \cap e_{i+1}$ is non-empty.

**Definition 4.** *A* Berge cycle *[4, 9] is a sequence $S_1, x_1, S_2, x_2, \ldots, S_n, x_n, S_{n+1}$ in a hypergraph $H$ such that: $x_1, \ldots, x_n$ are distinct vertices of $H$; $S_1, \ldots, S_n$ are distinct edges of $H$; the edges $S_{n+1}$ and $S_1$ are equal; for all $1 \leq i \leq n$ we have that $x_i$ is in $S_i \cap S_{i+1}$; and $n$ is greater than or equal to 2.*

A hypergraph is considered *Berge-cyclic* if it has a Berge cycle; it is considered *Berge-acyclic* otherwise.

*Example 1.* The hypergraph in Figure 4 is Berge-cyclic because it contains the Berge cycle $E_1, v_2, E_2, v_3, E_1$. The hypergraph in Figure 3 is Berge-acyclic.

In general, if there is a pair of edges of a hypergraph that share more than one node, then the hypergraph is Berge-cyclic.

It is well known that a sufficient method for achieving hyper-arc consistency for a constraint set with a Berge-acyclic constraint hypergraph is to ensure each constraint of the hypergraph is hyper-arc consistent [16, 17].

In the on-line version of the *Global Constraint Catalogue* [8], there are currently 38 of 120 constraint automata that induce Berge-acyclic constraint hypergraphs. Nevertheless, it can be seen from Figure 5 that, in general, if an automaton has counters, then the resulting hypergraph is Berge-cyclic. However, it is possible for an automaton with one counter and only one state to be transformed the decomposition into a constraint hypergraph that is Berge-acyclic by projecting away the state variables [5]; we will see later on (in Theorem 1) that this is a special case of a more general result. Next, we generalise the known results to a wider class of automata with counters.

We need some technical definitions; more details and examples can be found in [4]. Without loss of generality, in the rest of the paper, we only consider constraint satisfaction problems whose constraint hypergraph is connected and reduced. The *reduction* of a hypergraph is obtained by removing each edge that is properly contained in another edge. A hypergraph is called *reduced* if it is equal to its reduction.

The set of *partial edges* generated by a set $M$ is obtained by intersecting the edges in set $E$ of a hypergraph $(V, E)$ with a set of vertices $M \subset V$, that is, the set of edges $\{e \cap M \mid e \in E\} \setminus \{\emptyset\}$. This set of partial edges is said to be a *vertex-generated* set of partial edges. We sometimes refer to the original edges

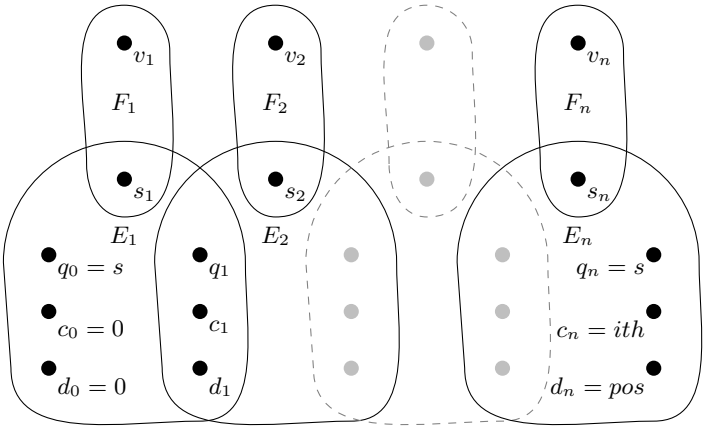


**Fig. 5.** Part of the constraint hypergraph induced by the counter automaton of the $ith\_pos\_different\_from\_zero(ith, pos, \langle v_1, \ldots, v_n \rangle)$ constraint, with counters $c$ and $d$ [8]

as *full edges*, in order to establish a clear difference with partial edges when it is needed.

**Definition 5.** *Let $H$ be a connected, reduced set of partial (or full) edges of some hypergraph, and let $G$ and $F$ be edges of $H$. Define $Q$ to be $G \cap F$. We say that $Q$ is an* articulation set *of $H$ if the result of removing $Q$ from every edge of $H$, that is $\{e \setminus Q \mid e \in F\} \setminus \{\emptyset\}$, is not a connected set of partial edges.*

The concept of an articulation set is a generalisation of an articulation point in a graph. For example, in Figure 4, the set $Q = E_1 \cap E_2$ is an articulation set.

A *block* of a reduced hypergraph is a connected vertex-generated set of partial edges with no articulation set. A block is said to be *trivial* if it contains zero or one edge. A reduced hypergraph is said to be *$\alpha$-acyclic* if all its blocks are trivial [4]. Intuitively, a hypergraph is $\alpha$-acyclic if removing articulation sets leaves the hypergraph disconnected; vertex-generated sets have to be considered so that the right edges are taken into account when looking at connected components.

*Example 2.* Consider again the hypergraph in Figure 4. It it is a reduced hypergraph since none of its edges are properly contained in any other edge. The set of edges $\{E_1, E_2\}$ is a connected set of full edges, but if we remove the articulation set $Q = E_1 \cap E_2 = \{v_2, v_3\}$ from them, then we obtain the set of partial edges $\{F \setminus \{v_2, v_3\} \mid F \in \{E_1, E_2\}\} \setminus \{\emptyset\} = \{v_1, v_4\}$. Thus the only blocks are $\{E_1\}$ and $\{E_2\}$. Since each block contains at most one edge, these blocks are trivial.

*Example 3.* Consider again the hypergraph in Figure 5. It is a reduced hypergraph. It has many articulation sets, for example $\{s_1\}$, $\{s_2\}$, $\ldots$, $\{s_n\}$, $\{q_1, c_1, d_1\}$, $\{q_2, c_2, d_2\}$, $\ldots$, $\{q_{n-1}, c_{n-1}, d_{n-1}\}$. In order to verify that the hypergraph is $\alpha$-acyclic, all its blocks must be checked for triviality. For example,

removing the articulation set $\{s_1\}$ from the hypergraph disconnects the partial edge $F_1 \setminus \{s_1\}$, which is a *trivial* block, from the rest of the hypergraph, which then still has many articulation sets. Removing any of the articulation sets of the form $\{q_i, c_i, d_i\}$ results in two disconnected components that both have articulation sets and hence are not blocks. So, any vertex-generated set of partial edges is either not a block or a trivial block.

In the case of constraint sets with $\alpha$-acyclic constraint hypergraphs, it is not straightforward to achieve hyper-arc consistency. But it can be done by using techniques from database theory [4].

Again we need some technical definitions. Given a constraint $R(\langle w_1, \ldots, w_n \rangle)$ and a tuple of distinct variables $\langle v_1, \ldots, v_k \rangle$ with $k < n$ such that for all $i$ the variable $v_i$ is equal to some variable $w_j$ in $\langle w_1, \ldots, w_n \rangle$, the *projection* of $R(\langle w_1, \ldots, w_n \rangle)$ onto $\langle v_1, \ldots, v_k \rangle$, denoted by $\text{proj}_{\langle v_1, \ldots, v_k \rangle}(R)$, is the restriction of $R$ to the variables $\langle v_1, \ldots, v_k \rangle$. Given a pair of variable tuples, $x = \langle x_1, \ldots, x_j \rangle$ and $y = \langle y_1, \ldots, y_\ell \rangle$, let $x \cap y$ denote the tuple that only contains the variables in the set-theoretic intersection, such that they appear in the same order as they appear in $x$ or $y$.

**Definition 6.** *A pair of constraints, $R_1(S_1)$ and $R_2(S_2)$, are said to be* pairwise consistent *if* $\text{proj}_{S_1 \cap S_2} R_1$ *is equal to* $\text{proj}_{S_1 \cap S_2} R_2$*. A set of constraints is said to be* pairwise consistent *if every constraint pair in that set is pairwise consistent.*

Let $\{R_1(S_1), \ldots, R_m(S_m)\}$ be a set of constraints with an $\alpha$-acyclic constraint hypergraph that are pairwise consistent, and let $R(S)$ be the constraint representing the set of solutions of the conjunction of these constraints, where $S$ is a tuple containing all the variables in $S_1, \ldots, S_m$. It can be shown [4] that for all $i$ the projection $\text{proj}_{S_i} R(S)$ is equal to $R_i(S_i)$. Thus, all tuples in the constraints $R_i(S_i)$ participate in a solution. This, unlike for general constraint problems, makes the solutions very easy to compute. In particular, if all the constraints are hyper-arc consistent, then the conjunction of the constraints is hyper-arc consistent.

**Theorem 1.** *Consider the transition constraints (5) from a counter-automaton-induced constraint decomposition:*

$$q_0 \in S \wedge T(q_0, q_1, \mathcal{C}_0, \mathcal{C}_1, v_1) \wedge \cdots \wedge T(q_{n-1}, q_n, \mathcal{C}_{n-1}, \mathcal{C}_n, v_n) \wedge q_n \in F \wedge \mathcal{C} = \mathcal{C}_n$$

*Maintaining these transition constraints pairwise consistent is equivalent to doing reachability analysis on the corresponding automaton.*

*Proof.* The transition constraint $T(q_{i-1}, q_i, \mathcal{C}_{i-1}, \mathcal{C}_i, v_i)$ codes the transition relation of the automaton. Achieving pairwise consistency on the pair $q_0 \in S$ and $T(q_0, q_1, \mathcal{C}_0, \mathcal{C}_1, v_1)$ forces the domain of $q_1$ and $\mathcal{C}_1$ to include only states and counter values that are reachable after consuming one alphabet symbol of the automaton. Thus $T(q_0, q_1, \mathcal{C}_0, \mathcal{C}_1, v_1)$ can be replaced by a constraint $T_0$, which is a sub-constraint of $T$ that only contains the tuples corresponding to transitions

that can happen in one step. Assume that hyper-arc consistency and pairwise consistency have been achieved up to step $i$ by the constraints:

$$q_0 \in S \wedge T(q_0, q_1, \mathcal{C}_0, \mathcal{C}_1, v_1) \wedge \cdots \wedge T(q_{i-1}, q_i, \mathcal{C}_{i-1}, \mathcal{C}_i, v_i)$$

By induction, the domains of $q_i$ and $\mathcal{C}_i$ are all states and counter values that are reachable after $i$ transitions in the automaton. Thus, achieving pairwise consistency on the pair $T_i(q_{i-1}, q_i, \mathcal{C}_{i-1}, \mathcal{C}_i, v_i)$ and $T(q_i, q_{i+1}, \mathcal{C}_i, \mathcal{C}_{i+1}, v_{i+1})$ gives a new constraint $T_{i+1}$ that contains only the subset of the tuples of $T_i$ that can take part in a chain of transitions of length $i + 1$.

*Example 4.* Consider the *ith_pos_different_from_zero*$(ith, pos, \langle v_1, \ldots, v_n \rangle)$ [8] constraint, where $ith$ is a constant, $pos$ is a decision variable, and $\langle v_1, \ldots, v_n \rangle$ is a sequence of decision variables. The constraint *ith_pos_different_from_zero* is satisfied when the element in the position $pos$ of the sequence $v$, that is the element $v_{pos}$, is the $ith$ non-zero element of the sequence. For example, *ith_pos_different_from_zero*$(2, 4, \langle 0, 1, 0, 3, 5 \rangle)$ is satisfied. Its predicate automaton with two counters is shown in Figure 6 and part of its constraint hypergraph is shown in Figure 5. The signature constraints are defined as $s_i = 0 \Leftrightarrow v_i \neq 0$ and $s_i = 1 \Leftrightarrow v_i = 0$. For example, given the values $ith = 2$ and $pos = 4$, the sequence $\langle 0, 1, 0, 3, 5 \rangle$ is accepted by the following chain of transitions:

$$(\rho_s, \langle c = 0, d = 0 \rangle) \xrightarrow{1} (\rho_s, \langle c = 0, d = 1 \rangle) \xrightarrow{0} (\rho_s, \langle c = 1, d = 2 \rangle) \xrightarrow{1}$$
$$(\rho_s, \langle c = 1, d = 3 \rangle) \xrightarrow{0} (\rho_s, \langle c = 2, d = 4 \rangle) \xrightarrow{0} (\rho_s, \langle c = 2, d = 4 \rangle)$$

As in the proof of Theorem 1, we denote the pairwise consistent transition relations obtained from the reachability analysis by $T_i$. The automaton has only one state, $\rho_s$, hence for any $i$ the constraint $T_i(q_{i-1}, q_i, \langle c_{i-i}, d_{i-1} \rangle, \langle c_i, d_i \rangle, s_i)$ is satisfied only when $q_{i-1} = q_i = \rho_s$.

Next, we analyse the counters $c$ and $d$. The analysis of values of the counters $c_i$ can be separated into two disjoint cases: $c_{i-1} \geq ith$ and $c_{i-1} < ith$. For the first case, when $c_{i-1} \geq ith$, the counters $c$ and $d$ are never updated because the guard $c < ith$ of the counter updates is not satisfied. In consequence, $T(q_{i-1}, q_i, \langle c_{i-i}, d_{i-1} \rangle, \langle c_i, d_i \rangle, s_i)$ is satisfied whenever $c_i = c_{i-1}$ and $d_i = d_{i-1}$, regardless of the value of $s_i$. For the second case, when $c_{i-1} < ith$, we have two cases depending on the value of the signature variable $s_i \in \{0, 1\}$. When $s_i = 0$ both counters are updated. Thus $T(q_{i-1}, q_i, \langle c_{i-i}, d_{i-1} \rangle, \langle c_i, d_i \rangle, s_i)$ is satisfied whenever $c_i = c_{i-1} + 1$ and $d_i = d_{i-1} + 1$. When $s_i = 1$ only $d$ is updated, while $c$ remains unchanged. Then $T(q_{i-1}, q_i, \langle c_{i-i}, d_{i-1} \rangle, \langle c_i, d_i \rangle, s_i)$ is satisfied if $c_i = c_{i-1}$ and $d_i = d_{i-1}$.

The constraints derived from the previous reachability analysis can be implemented straightforwardly to achieve hyper-arc consistency for the constraint *ith_pos_different_from_zero*. Note that these constraints must be enforced during search in order to maintain pairwise consistency.

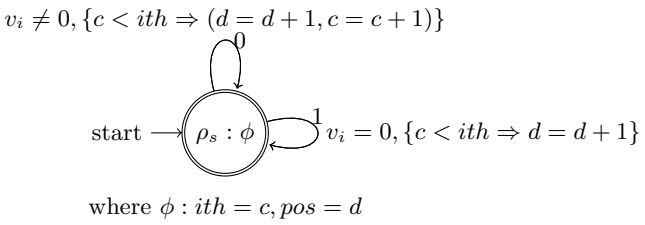


**Fig. 6.** Predicate automaton of the *ith_pos_different_from_zero*$(ith, pos, \langle v_1, \ldots, v_n \rangle)$ constraint, with counters $c$ and $d$ [8]. The labels 0 and 1 correspond to the values of the signature variables.

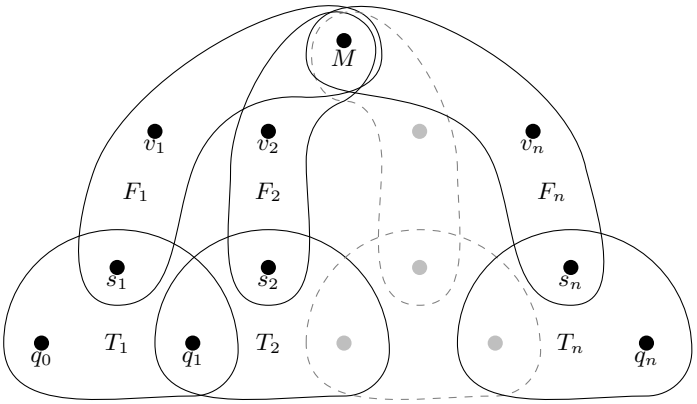


**Fig. 7.** Constraint hypergraph induced by the predicate automaton of the *maximum*$(M, \langle v_1, \ldots, v_n \rangle)$ constraint

## 4 Centred-Cyclic Automata

We now analyse another type of constraint hypergraph that appears in the online version of the *Global Constraint Catalogue* [8], namely centred-cyclic hypergraphs. We say that the constraint hypergraph of a predicate-automaton-induced decomposition is *centred-cyclic* if its signature constraints share at least one variable (vertex); see Figure 7 for an example. We show that it is possible to ensure hyper-arc consistency for this kind of constraint hypergraph. It can be verified that a centred-cyclic hypergraph is not $\alpha$-acyclic. Thus, to achieve hyper-arc consistency, new methods have to be used.

A centred-cyclic constraint decomposition has two groups of constraints: the transition constraints and the signature constraints. Considered alone, the transition constraints are Berge-acylic, while the signature constraints are $\alpha$-acyclic. In fact, if the signature constraints share exactly one variable, then they will

be Berge-acyclic, and hyper-arc consistency alone of the signature constraints is enough to achieve hyper-arc consistency of the signature constraints. So, using results from the previous section, it is possible to achieve hyper-arc consistency on each group separately, but this is not enough to achieve hyper-arc consistency of the whole decomposition. The transition constraints and the signature constraints only overlap in the sequence of signature variables $s_1, \ldots, s_n$.

In order to ensure hyper-arc consistency for the whole hypergraph, the projection of both subproblems onto their common variables must be the same: that is, the set of possible values for the sequence of signature variables $s_1, \ldots, s_n$ must be a solution to both the signature constraints and the transition constraints. Thus it is sufficient to add to the decomposition an implied constraint $\mathcal{I}(\langle s_1, \ldots, s_n \rangle)$ that is satisfied by a sequence of signature values $\langle \sigma_1, \ldots, \sigma_n \rangle$ if and only if the corresponding transition sequence is allowed by the automaton and the signature constraints. This constraint must be enforced during search in order to prune sequences of values that are not allowed by either the signature constraints or the transition constraints, thus maintaining hyper-arc consistency.

**Theorem 2.** *Given an automaton-induced decomposition, the constraint* $\mathcal{I}(\langle s_1, \ldots, s_n \rangle)$ *can be computed directly from the underlying automaton.*

*Proof.* Assuming that each predicate of the automaton is satisfiable for some assignment of the problem variables, the signature constraints (4) allow all possible values for each signature variable $s_i$. So the only restrictions on the values of the signature variables come from the transition constraints (3).

The transition constraints, together with the condition that $q_0$ and $q_n$ respectively belong to the start and accepting state sets, restrict the intermediate values to correspond to a chain of transitions from a start state to an accepting state. Thus the constraint $\mathcal{I}$ must restrict the values of the signature variables to correspond to the edge labels of accepting chains of transitions of the underlying automaton.

We now illustrate Theorem 2 on an example taken from [8].

*Example 5.* The constraint $maximum(M, \mathcal{V})$ constraints the decision variable $M$ to be the maximum of the sequence of decision variables $\mathcal{V}$. Its predicate automaton is given in Figure 8 and induces the constraint hypergraph in Figure 7, which is centred-cyclic. Since the automaton has no counters, the constraint hypergraph of the transition constraints is Berge-acyclic, as is the constraint hypergraph of the signature constraints, since they only share one variable, namely $M$. Thus, from Theorem 2, the constraint $\mathcal{I}$ corresponds to chains of transitions from 'start' to the accepting state $\rho_1$ in the underlying automaton given in Figure 9. Every sequence of signature values must contain a value other than 1, because every accepting path in Figure 9 has to pass through a transition labelled with a 2. In practice, it had been observed [5] that the *Maximum* constraint behaved as if it was hyper-arc-consistent. In this particular case, since there is only one impossible assignment for the signature variables, after any $s_i$ was assigned the value of 1, hyper-arc-consistency was achieved.

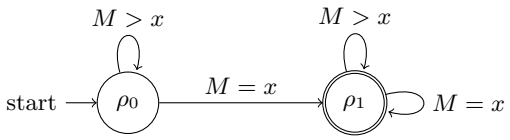


**Fig. 8.** Automaton of the *maximum*$(M, \mathcal{V})$ constraint

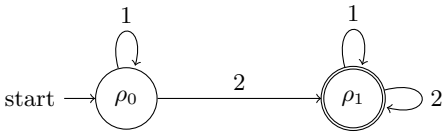


**Fig. 9.** Underlying automaton of the *maximum*$(M, \mathcal{V})$ constraint

This result can be generalised to cases where the signature constraints share more than one variable.

## 5 Conclusion

We have used the notions and results of hypergraphs and relational databases to derive properties of constraints given their hypergraph. In particular, we have shown that a constraint satisfaction problem whose hypergraph is $\alpha$-acyclic can be modified in order to achieve hyper-arc consistency. Also, we have shown a way to decompose centred-cyclic constraints and evaluate their consistency. We will now investigate the remaining five hypergraph constraint patterns currently identified in the on-line version of the *Global Constraint Catalogue* [8]. Moreover, we will study the impact of these results in practice.

**Acknowledgements.** The authors wish to thank anonymous referees for their comments on an earlier version of this paper. The second and third authors are supported by grant 2007-6445 of the Swedish Research Council (VR).

## References

1. Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.
2. Krzysztof R. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.
3. Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems.* Kluwer Academic Publishers, 2001.
4. Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, July 1983.

5. Nicolas Beldiceanu. Private communication, April 2011.
6. Nicolas Beldiceanu, Mats Carlsson, Romuald Debruyne, and Thierry Petit. Reformulation of global constraints based on constraints checkers. *Constraints*, 10(4):339–362, 2005.
7. Nicolas Beldiceanu, Mats Carlsson, and Thierry Petit. Deriving filtering algorithms from constraint checkers. In Mark Wallace, editor, *Proceedings of CP'04*, volume 3258 of *LNCS*, pages 107–122. Springer-Verlag, 2004.
8. Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. Global constraint catalogue: Past, present, and future. *Constraints*, 12(1):21–62, March 2007. The catalogue is at `http://www.emn.fr/z-info/sdemasse/gccat`.
9. Claude Berge. *Graphes et Hypergraphes.* Dunod, Paris, France, 1970.
10. Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, Claude-Guy Quimper, and Toby Walsh. Reformulating global constraints: The *slide* and *regular* constraints. In *Proceedings of SARA'07*, volume 4612 of *LNAI*, pages 80–92. Springer-Verlag, 2007.
11. Christian Bessière, George Katsirelos, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Decomposition of the NVALUE constraint. In David Cohen, editor, *Proceedings of CP'10*, volume 6308 of *LNCS*, pages 114–128. Springer-Verlag, 2010.
12. Christian Bessière, George Katsirelos, Nina Narodytska, and Toby Walsh. Circuit complexity and decompositions of global constraints. In Craig Boutilier, editor, *Proceedings of IJCAI'09*, pages 412–418, 2009.
13. Stéphane Bourdais, Philippe Galinier, and Gilles Pesant. HIBISCUS: A constraint programming application to staff scheduling in health care. In Francesca Rossi, editor, *Proceedings of CP'03*, volume 2833 of *LNCS*, pages 153–167. Springer-Verlag, 2003.
14. Mats Carlsson, Nicolas Beldiceanu, and Julien Martin. A geometric constraint over $k$-dimensional objects and shapes subject to business rules. In Peter J. Stuckey, editor, *Proceedings of CP'08*, volume 5202 of *LNCS*, pages 220–234. Springer-Verlag, 2008.
15. Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.
16. Philippe Janssen and Marie-Catherine Vilarem. Problèmes de satisfaction de contraintes : Techniques de résolution et application à la synthèse de peptides. Technical Report 54, Centre de Recherche en Informatique de Montpellier, France, 1988.
17. Philippe Jégou. *Contribution à l'étude des problèmes de satisfaction de contraintes: Algorithmes de propagation et de résolution. Propagation de contraintes dans les réseaux dynamique.* PhD thesis, Université de Montpellier II, Montpellier, France, 1991.
18. Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere! In *Proceedings of ATVA'05, the 3rd International Symposium on Automated Technology for Verification and Analysis*, volume 3707 of *LNCS*, pages 489–503. Springer-Verlag, 2005.
19. Gilles Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *Proceedings of CP'04*, volume 3258 of *LNCS*, pages 482–495. Springer-Verlag, 2004.
20. Claude-Guy Quimper and Toby Walsh. Decomposing global grammar constraints. In Christian Bessière, editor, *Proceedings of CP'07*, volume 4741 of *LNCS*, pages 590–604. Springer-Verlag, 2007.