Open Contractible Global Constraints

Michael J. Maher
NICTA* and University of NSW
Michael.Maher@nicta.com.au

Abstract

Open forms of global constraints allow the addition of new variables to an argument during the execution of a constraint program. Such forms are needed for difficult constraint programming problems where problem construction and problem solving are interleaved. However, in general, filtering that is sound for a global constraint can be unsound when the constraint is open. This paper provides a simple characterization, called contractibility, of the constraints where filtering remains sound when the constraint is open. With this characterization we can easily determine whether a constraint is contractible or not. In the latter case, we can use it to derive the strongest contractible approximation to the constraint. We demonstrate how specific algorithms for some closed contractible constraints are easily adapted to open constraints.

1 Introduction

The classic CSP model of constraint satisfaction [Dechter, 2003; Rossi *et al.*, 2006] has a fixed static collection of variables over which a solution must be found. However, in many problems it is natural for the presence of some variables to be contingent on the value of other variables. This is true of configuration problems and scheduling problems that involve process-dependent activities [Barták, 2003]. More generally, for difficult problems the intertwining of problem construction and problem solving provides a way to manage the complexity of a problem, and thus new variables and constraints may arise after solving has begun. Programming languages supporting constraint programming generally have the flexibility to add variables and constraints during the execution of a model.

However, global constraints do not have this flexibility: all variables must be available at the time the constraint is imposed, so variables cannot simply be added when they become available. The collection of variables they constrain is *closed*, rather than *open*. This can leave the filtering effect of the global constraint until too late in the execution, resulting in a large search space.

Recent work has focused on supporting open versions of global constraints. Barták [Barták, 2003] first formulated this issue and described a generic implementation technique to make open versions for the class of *monotonic* global constraints. But this technique is inefficient, and he also provided a specific implementation of the open ALLDIFFERENT constraint by modifying Régin's algorithm [Régin, 1994] for the closed ALLDIFFERENT.

A notion of Open CSP was investigated in [Faltings and Macho-Gonzalez, 2005]. In that work the set of variables is closed but the domains are open, that is, extra values can be added to variable domains.

Later work [van Hoeve and Régin, 2006] addressed the problem when the variables that might be added to a constraint are specified in advance and a set variable describes the set of variables that will participate in the constraint. In this formulation, it becomes possible to filter the set variable in addition to the individual variables. [van Hoeve and Régin, 2006] also gives an implementation the global cardinality constraint GCC and of multiple open GCC constraints over disjoint variables. Another work [Maher, 2009] allows only the number of variables to be constrained. However, in this paper we address Barták's model of open constraints.

A major difficulty in implementing open constraints is that a propagator for a closed constraint may be unsound for the corresponding open constraint. That is, the propagator may make an inference that turns out to be unjustified once the sequence of variables is extended. In this paper we focus on the issue of identifying constraints for which a closed propagator is sound as an open propagator. It turns out that these constraints have a simple characterization, which we call *contractibility*, and which allows us to easily determine whether a constraint is contractible or not. This characterization is also convenient for finding the tightest contractible approximation of an uncontractible constraint. We illustrate our results with a wide variety of global constraints.

For contractible constraints we can, in theory, implement an open constraint by interleaving the propagator for the closed constraint and variable extension. But this does not ensure that propagators for closed contractible constraints can be cleanly and easily modified to implement open constraints.

*NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

We examine specific propagator algorithms for several closed constraints and demonstrate that they are easily adapted to implement the corresponding open constraint.

After some preliminaries in Section 2 we introduce contractibility in Section 3. We show that it characterizes those constraints for which closed propagators remain sound when the constraint is open, and develop an algebra for constructing contractible constraints. We characterize contractibility in language-theoretic terms in Section 4, and use that characterization to identify contractible constraints and tight approximations of uncontractible constraints. In Section 5 we describe modified algorithms for several open contractible constraints.

2 Background

The reader is assumed to have a basic knowledge of constraint programming, CSPs, global constraints, and filtering, as might be found in [Dechter, 2003; Rossi *et al.*, 2006; Beldiceanu *et al.*, 2005].

For the purposes of this paper, a global constraint is a relation over a sequence of variables. Other arguments of a constraint are considered parameters and are assumed to be fixed before execution. While some global constraints (such as ALLDIFFERENT and GCC) are more naturally represented as relations over an unordered collection of variables, we will find it convenient to ignore this abstraction: for some constraints (such as SEQUENCE) a sequence of variables is necessary, and uniformity over all constraints will simplify our treatment of the issues. A sequence of variables will be denoted by $\vec{X}$ or $[X_1, \ldots, X_n]$.

We make no *a priori* restriction on the variables that may participate in the sequence except that, in common with most work on global constraints, we assume that no variable appears more than once in a single constraint.

There are some specific global constraints that we define for completeness. These and other global constraints are discussed more completely in [Beldiceanu *et al.*, 2005] and the references therein. The constraint ALLDIFFERENT$([X_1, \ldots, X_n])$ [Régin, 1994] states that the variables $X_1, \ldots, X_n$ take distinct values. The global cardinality constraint GCC$(\vec{v}, \vec{l}, \vec{u}, [X_1, \ldots, X_n])$ [Régin, 1996] states that, for every i, the value v_i occurs between l_i and u_i times in the list of variables. The constraint REGULAR$(\mathcal{A}, [X_1, \ldots, X_n])$ [Pesant, 2004] states that the value of the list of variables, when considered as a word, is accepted by the automaton $\mathcal{A}$. Similarly, the constraint CFG$(\mathcal{G}, [X_1, \ldots, X_n])$ [Quimper and Walsh, 2006; Sellmann, 2006] states that the value of the list of variables, when considered as a word, is generated by the context-free grammar $\mathcal{G}$.

The constraint SEQUENCE$(l, u, k, [X_1, \ldots, X_n], \vec{v})$ [Beldiceanu and Contejean, 1994] states that any consecutive sequence of k variables $X_j, \ldots, X_{j+k-1}$ contains between l and u occurrences of values from $\vec{v}$. The constraint SLIDINGSUM$(l, u, k, [X_1, \ldots, X_n])$ [Beldiceanu and Carlsson, 2001] states that the sum of any consecutive sequence of k variables lies between l and u. The constraint CONTIGUITY$([X_1, \ldots, X_n])$ [Maher, 2002] states that the variables X_i take values from $\{0, 1\}$ and the variables taking the value 1 are consecutive. The lexicographical ordering constraint $[X_1, \ldots, X_n] \leq_{lex} [Z_1, \ldots, Z_n]$ [Frisch *et al.*, 2002] states that the sequence of X variables is lexicographically less than or equal to the sequence of Z variables, where we assume some ordering on the underlying values. The precedence constraint $s \prec_{\vec{X}} t$ [Law and Lee, 2004] states that if t appears in the sequence $\vec{X}$ then s appears at a lower index.

Each variable X has a static type which defines a set of values $T(X)$ which it may take. In addition, generally, each variable has an associated set $S \subseteq T(X)$ of values, called its domain. We will view this simultaneously as: a function $D : Vars \rightarrow 2^{Values}$ where $D(X) = S$, a unary relation $D(X)$ which is satisfied only when the value of X is some $s \in S$, and the pointwise extension of D to sequences of variables.

3 Contractible constraints

We want to extend a constraint $C([X_1, \ldots, X_n])$ with an extra variable Y to $C([X_1, \ldots, X_n, Y])$. We would like to do filtering on the smaller constraint without knowing whether it will be extended to Y, or further. When we can do this, we have a kind of monotonicity property of C.

Definition 1 *We say a constraint $C([X_1, \ldots, X_n])$ is* contractible *if, there is a number m such that for all $n \geq m$ we have*

$$C([X_1, \ldots, X_n, Y]) \rightarrow C([X_1, \ldots, X_n])$$

The least such m is called the contractibility threshold[1].

Thus C is contractible iff every solution of $C([X_1, \ldots, X_n, Y])$, when restricted to $X_1, \ldots, X_n$ where $m \leq n$, is a solution of $C([X_1, \ldots, X_n])$. The property is akin to the "optimal substructure" property that is a pre-requisite for the use of dynamic programming in optimization problems [Cormen *et al.*, 2001] which requires that optimal solutions of a problem also solve subproblems optimally. Here it is only satisfiability, and not optimality, that is involved.

It follows that any sound form of filtering (such as arc consistency or bounds consistency) on a contractible constraint $C([X_1, \ldots, X_k])$ is safe in the sense that any values deleted from domains in that process could also be deleted while filtering on $C([X_1, \ldots, X_n])$ for any $n \geq k$.

Proposition 1 *Let C be a contractible constraint. Suppose a sound filtering algorithm for $C([X_1, \ldots, X_n])$ reduces the domain D for $\vec{X}$ to D'. Then*

$$D(\vec{X}) \wedge C([X_1, \ldots, X_n, Y]) \leftrightarrow D'(\vec{X}) \wedge C([X_1, \ldots, X_n, Y])$$

Furthermore, if this property holds for all sound filterings then C must be contractible.

[1] In most cases the contractibility threshold is 0, and we will not consider constraints with a higher threshold in this paper.

Consequently, for contractible constraints, filtering does not need to be undone if the list is lengthened. That is, algorithms for filtering a closed contractible constraint are valid also for the corresponding open constraint.

Conversely, any constraint that is not contractible might need to undo the effects of filtering if the list is lengthened. If σ is a solution of $C([X_1, \ldots, X_n, Y])$, but not of $C([X_1, \ldots, X_n])$ then propagation on $C([X_1, \ldots, X_n])$ might eliminate σ. For example, a constraint $\sum_i X_i = 5$ would propagate $X_1 = 5$ if the sequence $\vec{X}$ contains just one variable, thus eliminating solutions such as $X_1 = 2, X_2 = 3$. When the second variable is added, all propagation that is a consequence of the inference $X_1 = 5$ must be undone.

The second part of this proposition shows that contractibility exactly characterizes the guarantee that closed filtering is safe for open constraints. That is, it is exactly the contractible constraints for which it is always sound to interleave closed filtering and addition of new variables.

The notion of contractibility is a variation of Barták's monotonicity [Barták, 2003] where we do not explicitly discuss variable domains. Before proceeding, we make this claim precise. We formulate Barták's monotonicity as follows.

Definition 2 *Let D be a domain. We say a constraint C is* monotonic *wrt D if, for any pair of disjoint sequences of variables $\vec{X}$ and $\vec{Y}$*

$$\{\vec{X} \mid C(\vec{X}\vec{Y}) \wedge D(\vec{X}) \wedge D(\vec{Y})\} \subseteq \{\vec{X} \mid C(\vec{X}) \wedge D(\vec{X})\}$$

Contractibility differs from monotonicity in that the definition is based entirely on the constraint, independent of the domains of variables. Hence it is not tied to domain-based reasoning; it is equally compatible with the more general framework of [Maher, 2003]. On the other hand, monotonicity is more flexible in reasoning about constraints that are only "partly contractible". The close relationship between monotonicity and contractibility is clear.

Proposition 2 *If C is contractible then for any domain D, C is monotonic wrt D. Conversely, if C is monotonic wrt every domain D then C is contractible.*

The SLIDE_j meta-constraint [Bessière *et al.*, 2008] can be used to define several constraints on a sequence of variables. We use a variant of SLIDE_j that starts applying the constraint at the p^{th} position, rather than the first. $\text{SLIDE}_j^p(C, \vec{X})$ holds iff $C(X_{ij+p}, \ldots, X_{ij+p+k-1})$ holds for $i = 0, 1, \ldots, \lfloor \frac{n-p-k+1}{j} \rfloor$. SLIDE_j is equal to SLIDE_j^1. SLIDE is equal to SLIDE_1.

Constraints defined directly with SLIDE_j^p are contractible.

Proposition 3 *Any constraint C defined by the SLIDE_j^p meta-constraint as $C(\vec{X}) \leftrightarrow \text{SLIDE}_j^p(C', \vec{X})$ is contractible.*

Since the SEQUENCE and SLIDINGSUM constraints can each be defined as $\text{SLIDE}(C', \vec{X})$, for appropriate constraint C', it follows that they are both contractible.

For constraints where the order of variables is unimportant we can define a meta-constraint analogous to SLIDE, which we will call SPLASH. Like SLIDE, it takes a fixed arity constraint C' and a sequence of variables $\vec{X}$ as arguments. Let C' have arity k, and $\vec{X}$ have length n, and let $S_k(\vec{X}) = \{[X_{i_1}, \ldots, X_{i_k}] \mid i_j < i_{j+1} \text{ for } j = 1, ..., k-1\}$. Then we define $\text{SPLASH}(C', \vec{X}) \leftrightarrow \bigwedge_{\vec{Y} \in S_k(\vec{X})} C'(\vec{Y})$. $\text{SPLASH}(C', \vec{X})$ applies C' to every subsequence of $\vec{X}$ of length k. For example, we can define $\text{ALLDIFFERENT}(\vec{X})$ as $\text{SPLASH}(\neq, \vec{X})$ and $\text{INTERDISTANCE}(\vec{X})$ as $\text{SPLASH}(C', \vec{X})$ where $C'(Z_1, Z_2) \leftrightarrow |Z_1 - Z_2| \geq p$. Thus, by the following proposition, ALLDIFFERENT and INTERDISTANCE are contractible.

Proposition 4 *Any constraint C defined by the* SPLASH *meta-constraint as $C(\vec{X}) \leftrightarrow \text{SPLASH}(C', \vec{X})$ is contractible.*

A constraint C of fixed arity k, when applied to a sequence of variables $\vec{X}$, is assumed to be applied only to the initial segment $X_1, \ldots, X_k$, or not at all if $\vec{X}$ is shorter than k. With this definition, C is contractible.

Once we have some contractible constraints, there are many ways to build other contractible constraints, as the following proposition demonstrates.

Proposition 5 *Let $C_1(\vec{X})$ and $C_2(\vec{X})$ be contractible constraints on the same sequence of variables. Let $C(X_1, \ldots, X_k)$ be a constraint of fixed arity. Then*

- *C is contractible*
- *$C_1 \wedge C_2$ is contractible*
- *$C_1 \vee C_2$ is contractible*
- *$\exists X_i\ C_1$ is contractible*
- *$\forall X_i\ C_1$ is contractible*

In general, the negation or implication of contractible constraints is not contractible.

The previous results give us an algebra for constructing complex contractible constraints, and can be used to demonstrate that some existing constraints are contractible. For example, CONTIGUITY is implemented in [Maher, 2002] essentially as $\exists \vec{L}, \vec{R}\, SLIDE_3^2(C', [L_1, X_1, R_1, L_2, \ldots, X_n, R_n])$, where C' has arity 7. Similarly, $(\vec{X} \leq_{lex} \vec{Y})$ is encoded in [Bessière *et al.*, 2008] essentially as $\exists \vec{B}\quad SLIDE_3(C', [B_1, X_1, Y_1, B_2, \ldots, X_n, Y_n])$ where C' has arity 4. By the previous propositions, CONTIGUITY and $\leq_{lex}$ are contractible. Similarly, we can define a weak version of GCC where there are no lower bounds $\text{GCC}(\vec{v}, \vec{0}, \vec{u}, [X_1, \ldots, X_n])$ as $\bigwedge_{v_i \in \vec{v}} \text{SPLASH}(C'_i, \vec{X})$, where C'_i has arity $u_i + 1$ and states that not all its arguments are equal to v_i. By the previous propositions, this weak form of GCC is contractible.

However, it is notable that the REGULAR constraint is not contractible, despite the implementation in terms of SLIDE outlined in [Bessière *et al.*, 2008].

Example 1 *Let $\mathcal{A}$ be an automaton that accepts the language $a + b^2$. Then $\text{REGULAR}(\mathcal{A}, [X_1]) \rightarrow X_1 = a$ but $\text{REGULAR}(\mathcal{A}, [X_1, Y]) \rightarrow X_1 = b$. Thus* REGULAR *is not contractible.*

The discrepancy arises because REGULAR is not constructed from the operations in the above propositions. Essentially, the implementation defines

$$\begin{aligned}&\text{REGULAR}(\mathcal{A}, [X_1, \ldots, X_n]) \leftrightarrow \\ &\quad \exists \vec{Q}\ \text{SLIDE}_2(Transition, [Q_0, X_1, Q_1, \ldots, X_n, Q_n]) \\ &\qquad \wedge Start(Q_0) \wedge Final(Q_n)\end{aligned}$$

where the 3-ary constraint $Transition$ expresses the state transitions of $\mathcal{A}$, $Start$ defines the start state(s) and $Final$ defines the final state(s). It is the constraint on the final variable Q_n that leads to uncontractibility; the remainder is expressible within the algebra.

4 Characterizing Contractibility

We can characterize when global constraints, such as REGULAR, are contractible by viewing the solutions of a global constraint as a formal language.

By insisting that variables in a constraint form a sequence we are able to view global constraints as defining a formal language. Each constraint $C(\vec{X})$ defines the language $L_C = \{\sigma(X_1 \ldots X_n) \mid \sigma \text{ is a solution of } C([X_1, \ldots, X_n])\}$ which is the set of solutions of C, considered as words. Let $P(L) = \{w \mid \exists u\ wu \in L\}$ denote the set of prefixes of a language L, called the prefix-closure of L. We say L is *prefix-closed* if $P(L) = L$.

We can now make a simple observation that provides a useful characterization of contractible constraints. If $\mathcal{A}$ defines a prefix-closed language then $\text{REGULAR}(\mathcal{A}, \vec{X})$ is contractible. This claim holds more generally.

Proposition 6 *Let $C(\vec{X})$ be a constraint over a sequence of variables. Then C is contractible iff L_C is prefix-closed.*

This result applies to constraints based on formal languages, such as REGULAR and CFG, but it also applies to constraints that are formulated differently. Thus, for example, the solutions of SEQUENCE and SLIDINGSUM are prefix-closed. Conversely, we see that constraining the final variable in a sequence, as in $Final(Q_n)$, is not contractible.

We can use this characterization both to determine whether a constraint is contractible or not, and as the basis for approximations of uncontractible constraints. We explore these possibilities in the following subsections.

4.1 Classifying Constraints

It is not within the scope of this paper to determine the contractibility of every global constraint. Nevertheless, we can outline and demonstrate some principles that make it easy, in most cases, to classify a global constraint as contractible or not.

In general, constraints based on counting with a lower bound (or equality) are not contractible. We can see this by noting that any non-trivial lower bound on the number of things in a sequence (or satisfied by a sequence) may be violated by a prefix of the sequence. This was already touched upon in [Barták, 2003], where the SUM constraint $\sum_{i=1}^{n} X_i = N$ was shown to be non-monotonic, but the argument holds for a wide range of constraints.

For example, PEAK counts the number of peaks in a sequence, but a prefix of the sequence may have fewer peaks. Similarly, STRETCH places lower bounds on the span of stretches, so that 1122 might be a solution, while 112 is not. By a similar argument, constraints identifying properties of an extreme element in a sequence, such as HIGHESTPEAK, are not contractible. On the other hand NOPEAK is contractible since, to the extent that there is counting, there is no lower bound – only an upper bound of 0.

We can generalize and formalize these observations. A function f is a *non-decreasing accumulation function* if it maps sequences of values to values such that, for every sequence $\vec{X}$ and value Y, $f([X_1, \ldots, X_n, Y]) \geq f([X_1, \ldots, X_n])$. Among such functions are counting the number of elements in a sequence with a fixed property, identifying the highest peaks, and summing (some) non-negative elements of a sequence. Note that summing possibly negative elements of a sequence is not non-decreasing. The first part of the following proposition is an almost direct consequence of the definitions of contractible and non-decreasing accumulation function.

Proposition 7 *Let C be a global constraint.*

- *Suppose C can be expressed as $f(\vec{X}) \leq Z$. Then C is contractible iff f is a non-decreasing accumulation function.*
- *Suppose C can be expressed as $f(\vec{X}) \geq Z$. Then C is contractible iff f is a non-increasing accumulation function.*
- *Suppose C can be expressed as $f(\vec{X}) = Z$. Then C is contractible iff f is a constant function.*

Thus the constraint $\sum_{i=1}^{n} X_i = N$ is not contractible, even when the X_i's are constrained to be non-negative. Similarly, $\sum_{i=1}^{n} X_i \geq N$ is not contractible while, under the restriction on the X_i's $\sum_{i=1}^{n} X_i \leq N$ is contractible. This result can be used to establish that PEAK, and HIGHESTPEAK are not contractible and that NOPEAK is contractible, but it also applies to many other counting and summing constraints in [Beldiceanu *et al.*, 2005].

Notice that in constraints like SEQUENCE and SLIDINGSUM the use of a lower bound in the description of the constraint C' to which SLIDE is applied does not prevent contractibility. Each lower bound applies only to a small part of the sequence. However, the RELAXEDSLIDINGSUM constraint, which weakens the SLIDINGSUM constraint by putting bounds on the number of times the C' constraint is satisfied, is not contractible, because counting is an accumulation function that is not non-increasing and the lower bound applies to the entire sequence.

Some constraints can be recognised as contractible, based only on their informal semantics. For example, DIFFN and DISJUNCTIVE enforce that objects represented by the variables are non-overlapping. Clearly, if $\vec{X}Y$ forms a non-overlapping set, then so does $\vec{X}$ alone. Thus contractibility follows directly from Definition 1. Similarly, CUMULATIVE, BINPACKING and DISJOINT are contractible.

For other constraints, their informal semantics lead easily to counterexamples to contractibility. Constraints that involve

computing the mean/average, median, mode, standard deviation, etc of the sequence are not contractible. This is easily recognised since these statistics are not, in general, preserved after eliminating part of the sample set, and hence are not prefix-closed. Alternatively, we could recognise that these functions are not non-increasing, nor non-decreasing and apply Proposition 7.

The idea of contractibility is not useful for all global constraints. For example, it appears irrelevant to cyclic constraints like the cyclic REGULAR, cyclic SEQUENCE and cyclic STRETCH constraints. In these constraints the sequence of variables is representing a cycle or circular list and there is no natural end at which to add variables. Thus it is not surprising that these constraints are not contractible.

There is sometimes a fine line between contractible and uncontractible constraints. For example, while $\leq_{lex}$ is contractible, $<_{lex}$ is not. To see the latter, observe that $111 <_{lex} 112$, but the corresponding prefixes are not strictly smaller – they are equal. If the precedence constraint $s \prec_{\vec{X}} t$ also required that t appear in $\vec{X}$ then the constraint is no longer contractible (because rst satisfies this constraint, but rs does not). Finally, notice that the SEQUENCE constraint is contractible, but it has the form $\text{SLIDE}(C', \vec{X})$ where C' is essentially a fixed-arity AMONG constraint; however, the (variable-arity) AMONG constraint is *not* contractible.

4.2 Approximating Constraints

One approach to implementing uncontractible constraints is to employ a contractible approximation of the constraint as an open constraint, and then replace it with the original constraint once the sequence of variables is closed [Barták, 2003]. The approximation must include all solutions of the original constraint, to ensure that no solutions of that constraint are lost during filtering of the approximation. Propositions 6 and 7 can be used to identify the tightest such approximation.

By Proposition 6, the tightest approximation of a constraint is its prefix-closure. This is a convenient characterization for language-based constraints. It is not difficult to show that the the prefix-closure of an automaton can be obtained by considering all states on a path between the start state and a final state in the automaton to be final states. Thus, for a REGULAR constraint, the tightest approximation is also a REGULAR constraint which can be obtained from the original simply by slightly modifying which states are final. Context-free grammars also have a straightforward construction of the prefix-closure. For CFG constraints also, the tightest approximation is also a CFG constraint.

As discussed in [Barták, 2003], a constraint $\sum_{i=1}^{n} X_i = N$ where the X_i's must be non-negative is not monotonic but is approximated by the constraint $\sum_{i=1}^{n} X_i \leq N$. Using Proposition 7 we can recognise this as the tightest contractible approximation. Similarly, for a counting constraint such as $\text{PEAK}(\vec{X}, N)$, which states that there are exactly N peaks in $\vec{X}$, the tightest contractible approximation states that N is an upper bound on the number of peaks. The tightest approximation of the GCC is the weak form of GCC discussed in Section 3. In all these cases, since counting is a non-decreasing accumulation function, the tightest contractible approximation is to eliminate the lower bounds. In $\text{HIGHESTPEAK}(\vec{X}, Z)$, the height of the highest peak is a non-decreasing accumulation function and so the tightest approximation states that Z is an upper bound on the height of the highest peak.

On the other hand, for some constraints where the accumulation function is neither non-increasing nor non-decreasing there appear to be no non-trivial approximations. For example, consider a constraint $\text{AVERAGE}(\vec{X}, M)$ stating that M is the mean/average of the values of $\vec{X}$. Given a fixed M, *any* sequence of values can be a prefix of a sequence with mean M. Hence the tightest contractible approximation of AVERAGE is the constraint that accepts any sequence, that is, the constraint $true$. For such a constraint there is no propagation until the constraint is closed.

In [Maher, 2009] these results are carried through to the design of propagators for open GCC, REGULAR and CFG.

5 Propagator Algorithms

It is perhaps unsurprising that implementations of closed contractible constraints often can easily be adapted to provide a direct implementation of the corresponding open constraint. As observed earlier, contractible constraints can use filtering algorithms designed for closed constraints also for the corresponding open constraint. Barták's generic implementation technique [Barták, 2003] exploits this fact for monotonic constraints by replacing the closed implementation of $C([X_1, \ldots, X_n])$ with the closed implementation of $C([X_1, \ldots, X_n, Y])$ when the sequence is extended. However, it is preferable to preserve the existing state of C and extend it slightly to account for the presence of the new variable, if this is possible. Such an implementation, which provides a direct implementation of the open constraint, avoids duplication of work. In this case, the main issue in implementing an open version of a contractible constraint is handling the extension of the sequence of variables; more precisely, the important invariants of the implementation must be preserved or recovered after a variable is added. In this section we describe the changes needed to make open several contractible constraints.

Constraints defined using only SLIDE, SPLASH, conjunction and existential quantification, such as CONTIGUITY and $\leq_{lex}$, can be implemented by the corresponding decomposition. In that case, the implementation can handle the sequence extension simply by asserting a copy or copies of the constraint C' appropriate to the extension.

The lexicographical ordering constraint $\vec{X} \leq_{lex} \vec{Y}$ is implemented in [Frisch *et al.*, 2002] by keeping track of (a) the lowest index i where X_i and Y_i are not both ground and equal, and (b) the lowest index i where $X_i >_{lex} Y_i$. For each property, if there is no satisfying index an infinite value is used. The extension of the sequences with new variables will not affect the validity of these indices, since the new variable is either inserted after a finite index or before an infinite index. The implementation of the precedence constraint $s \prec_{\vec{X}} t$ [Law and Lee, 2004] uses a similar method, and is similarly easily adapted.

The flow-based implementation of the SEQUENCE constraint [Maher *et al.*, 2008] keeps a feasible flow in a flow graph to represent the current state of the constraint. Suppose the width of a window is k. When a variable is added, the flow graph must be updated by adding two nodes (representing the new window) and 5 arcs (including one representing the new variable), redirecting $k-1$ arcs (to arrive at the final node instead of the previously-final node; these arcs represent the existing variables that appear in the new window), and changing the capacity of one arc (the outflow from the previously-final node). The feasible flow of the original graph can be extended to these additions in constant time, using the fact that a solution σ to SEQUENCE$([X_1, \ldots, X_n])$ can be extended to a solution of SEQUENCE$([X_1, \ldots, X_n, Y])$ by defining $\sigma(Y) = \sigma(X_{n-k+1})$.

The shortest path implementation of SEQUENCE, denoted by CD in [Brand *et al.*, 2007], uses a potential function (for an incremental shortest path algorithm [Cotton and Maler, 2006]) on a weighted graph to represent the current state of the constraint. When a variable is added, a node and four arcs are added to the graph and it is necessary to provide the potential of the new node. We can choose this potential to be $\pi(X_n) + \pi(X_{n-k+1}) - \pi(X_{n-k})$, using the fact above.

6 Conclusions

We have characterized the constraints for which the extension of the sequence of variables does not affect the soundness of propagation, both in logical and language-theoretic terms. We also identified an algebra for constructing contractible constraints. These results were used both to easily classify constraints as contractible or not, and to identify the tightest contractible approximation of constraints. We also showed how open versions of SEQUENCE, and lexicographic and precedence constraints can be implemented. It is noteworthy that these algorithms are the first described for open constraints over a sequence of variables, rather than a set of variables as in ALLDIFFERENT and GCC.

References

[Barták, 2003] R. Barták. Dynamic global constraints in backtracking based environments. *Annals of Operations Research*, 118:101–119, 2003.

[Beldiceanu and Carlsson, 2001] N. Beldiceanu and M. Carlsson. Revisiting the cardinality operator and introducing cardinality-path constraint family. In *Proc. of the Int. Conf. on Logic Programming*, pages 59–73, 2001.

[Beldiceanu and Contejean, 1994] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 12:97–123, 1994.

[Beldiceanu *et al.*, 2005] N. Beldiceanu, M. Carlsson, and J-X. Rampon. Global constraint catalog. T-2005-08, SICS Technical Report, 2005.

[Bessière *et al.*, 2008] C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Slide: a useful special case of the cardpath constraint. In *ECAI*, pages 475–479, 2008.

[Brand *et al.*, 2007] S. Brand, N. Narodytska, C.-G. Quimper, P. Stuckey, and T. Walsh. Encodings of the sequence constraint. In *Proc. CP*. Springer, LNCS 4741, 2007.

[Cormen *et al.*, 2001] T.H. Cormen, C.E. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[Cotton and Maler, 2006] S. Cotton and O. Maler. Fast and flexible difference constraint propagation for DPLL(T). In *Proc. SAT-2006*, pages 170–183, 2006.

[Dechter, 2003] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[Faltings and Macho-Gonzalez, 2005] B. Faltings and S. Macho-Gonzalez. Open constraint programming. *Artificial Intelligence*, 161(1-2):181–208, 2005.

[Frisch *et al.*, 2002] A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographic orderings. In *Proc. CP*. Springer, LNCS 2470, 2002.

[Law and Lee, 2004] Y.C. Law and J.H.M. Lee. Global constraints for integer and set value precedence. In *Proc. CP*, pages 362–376. Springer, LNCS 3258, 2004.

[Maher *et al.*, 2008] M.J. Maher, N. Narodytska, C-G. Quimper, and T. Walsh. Flow-based propagators for the sequence and related global constraints. In *Proc. CP*, pages 159–174. Springer, LNCS 5202, 2008.

[Maher, 2002] M.J. Maher. Analysis of a global contiguity constraint. In *Proc. Workshop on Rule Based Constraint Reasoning and Programming*, 2002. Held with CP2002.

[Maher, 2003] M.J. Maher. A synthesis of constraint satisfaction and constraint solving. In F. Rossi, editor, *Proc. CP*, pages 525–539. Springer, LNCS 2833, 2003.

[Maher, 2009] M.J. Maher. Open constraints in a boundable world. In *Proc. CPAIOR*. Springer, LNCS 5547, 2009.

[Pesant, 2004] G. Pesant. A regular language membership constraint for finite sequences of variables. In *Proc. CP*, pages 482–295. Springer, LNCS 3258, 2004.

[Quimper and Walsh, 2006] C-G. Quimper and T. Walsh. Global grammar constraints. In *Proc. CP*, pages 751–755. Springer, LNCS 4204, 2006.

[Régin, 1994] J-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proc. AAAI*, pages 362–367. AAAI Press/The MIT Press, 1994.

[Régin, 1996] J-C. Régin. Generalized arc consistency for global cardinality constraints. In *Proc. AAAI*, pages 209–215. AAAI Press/The MIT Press, 1996.

[Rossi *et al.*, 2006] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. 2006.

[Sellmann, 2006] M. Sellmann. The theory of grammar constraints. In *Proc. CP*, pages 530–544. LNCS 4204, 2006.

[van Hoeve and Régin, 2006] W.J. van Hoeve and J-C. Régin. Open constraints in a closed world. In *CPAIOR*, pages 244–257, 2006.