To Be or not To Be.. a Global Constraint

Christian Bessière, Pascal van Hentenryck

To Be or Not to Be . . . a Global Constraint*

Christian Bessière[1] and Pascal Van Hentenryck[2]

[1] LIRMM-CNRS, 161 rue Ada, 34392 Montpellier Cedex 5, France
bessiere@lirmm.fr
[2] Brown University, P.O. Box 1910, Providence, RI 02912
pvh@cs.brown.edu

Abstract. Constraint propagation is widely recognized as a fundamental reasoning component in constraint programming. In the last decade, the concept of "global constraint" has attracted significant attention, since it is critical to achieve reasonable pruning, and efficiency, in many applications. However, even if the name "global constraint" carries a strong intuition in itself, there is no formal definition of this important concept. This paper proposes various notions of globality in order to understand this concept more thoroughly.

1 Introduction

Constraint technology is widely used to solve a large scope of combinatorial problems arising in various application fields such as resource allocation, hardware verification, diagnosis, scheduling, etc. Progresses in constraint technology usually come from two close subareas traditionally named 'constraint reasoning' (or CSP), and 'constraint programming' (or CP). Thanks to common events such as the CP conference series, these two communities became closer and closer, and their border became more fuzzy. Nevertheless, scientists from these two subfields often have different cultural origins, vocabulary, and ways of approaching theoretical and practical issues. An example that illustrates such differences appeared during the CP'02 conference, held at Ithaca NY. CP'02 featured a tutorial, whose title was "global constraints". This tutorial gave rise to a heated debate, not because of its content, but rather because nobody seemed to agree on the definition of "global constraint".

This paper tries to characterize the concept of "global constraint" formally. We understand that proposing a definition for a concept already widely used is difficult and inevitably controversial. We also understand that other definitions may be proposed and that our definitions represent our own biases. However, we believe that this endeavour can only increase our understanding of global constraints and thus benefits the community as a whole. In particular, we believe that our definitions, which build on well-known concepts, isolate some fundamental intuitions in the folklore of the communities, and are consistent with the "practice of constraint programming". In the worst case, these definitions will be a first step toward a fundamental understanding of this important concept.

* [1] contains a long version of this short paper.

F. Rossi (Ed.): CP 2003, LNCS 2833, pp. 789–794, 2003.

The starting point of this paper is the recognition that a constraint C is often called "global" when "processing" C as a whole gives better results than "processing" any conjunction of constraints that is "semantically equivalent" to C. Thus, the concept of globality goes beyond "semantic equivalence" and seems to include operational and algorithmic concepts. Consider the well-known example of the *alldiff* constraint: *alldiff*$(x_1, \ldots, x_k)$ holds when all the x_i's are given different values. This constraint can easily be represented by a clique of binary inequalities on the x_i's. Hence, it may not be considered "global", since it can be decomposed into more primitive constraints. However, performing arc consistency on the clique does not usually prune as many values as performing arc consistency directly on the *alldiff* constraint. Hence, the *alldiff* constraint can be considered "global" wrt the filtering property, which is clearly one important criterion for "globality" in constraint programming. More generally, this paper distinguishes between semantic globality (expressiveness), operational globality (quality of filtering), and algorithmic globality (computational efficiency of the filtering).

The paper also addresses the issue of globality both from a CSP and CP standpoint. The CSP standpoint does not restrict the constraint language and gives considerable freedom in the choice of domains and constraints. But the CSP standpoint must harness this freedom and imposes a natural, but strong, restriction on the nature of globality. As a result, the CSP standpoint is best seen as a theory of globality for conjunctive constraints. The CP standpoint takes the dual approach and restricts the language under consideration. As a consequence, it makes it possible to encompass complex rewritings in the definition of globality.

2 Background

Constraints are defined in a slightly unusual way in order to make them independent from the constraint network in which they appear.

Definition 1 (Constraint). *A constraint (or **relation**) R of arity k is a set of sequences of k components. A component can be any entity/object in the world.*

Example 1. The constraint *alldiff* of arity k is defined by the set of all the sequences of k different components. If $k = 3$, *alldiff* is $\{(2, 3, 1), (cow, car, cup), \ldots\}$.

We now define the notion of constraint instance, which is traditionally called "constraint" in the CSP community. It links a constraint with its variables and their domains.

Definition 2 (Constraint Instance). *An **instance** c of a constraint R is a triple (X_c, D_c, R_c), where X_c is an ordered set $(x_1, \ldots, x_{|X_c|})$ of variables, $D_c = (D_c(x_1), \ldots, D_c(x_{|X_c|}))$ is an ordered set representing the domains of these variables. The set $R \cap D_c(x_1) \times \cdots \times D_c(x_{|X_c|})$ is denoted by $sol(c)$.*

Example 2. The instance of the constraint *alldiff* posted on the three variables x_1, x_2, and x_3 of respective domains $\{a, b\}$, $\{a, b\}$, and $\{a, b, c, d\}$ allows the solutions (a, b, c), (a, b, d), (b, a, c), and (b, a, d) for the variables x_1, x_2, x_3.

Definition 3 (Constraint Network). *A finite* ***constraint network*** N *is defined as a triplet* (X_N, D_N, C_N) *where* $X_N = \{x_1, \ldots, x_n\}$ *is a set of* n ***variables***, $D_N = \{D_N(x_1), \ldots, D_N(x_n)\}$ *is a set of finite* ***domains***, *and* C_N *is a set of* ***constraint instances*** *such that* $\forall\, c \in C_N : X_c \subseteq X_N, \forall x \in X_c : D_c(x) = D_N(x)$. *Finally, the set of constraint networks is denoted by* $\mathcal{N}$.

We now introduce the concept of constraint decomposition, which is fundamental in characterizing global constraints.

Definition 4 (Constraint Decomposition). *A constraint network* N *is a* ***decomposition*** *of a constraint instance* c *if* $X_N = X_c$, $D_N = D_c$, $\forall e \in C, |X_e| < |X_c|$, $R_e = R_c[X_e]$, *and* $sol(N) = sol(c)$.

Example 3. Let $N = (X, D, C)$ be the network defined by $X = \{x_1, x_2, x_3\}$, where $D(x_1) = \{a, b\}$, $D(x_2) = \{a, b\}$, $D(x_3) = \{a, b, c, d\}$, and $C = \{c_{12}, c_{13}, c_{23}\}$, where c_{12}, c_{13}, c_{23} are the binary inequality $\neq$ posted on (x_1, x_2), (x_1, x_3), and (x_2, x_3). N is a decomposition of the *alldiff* instance posted on (x_1, x_2, x_3) in Example 2.

A constraint decomposition scheme is simply a function which decomposes the instances of a constraint. This concept simplifies subsequent definitions.

Definition 5 (Constraint Decomposition Scheme). *Let* R *be a constraint and let* $\mathcal{C}$ *be the set of instances of* R. *A* ***constraint decomposition scheme*** *for* R *is a function* $\delta : \mathcal{C} \rightarrow \mathcal{N}$ *such that* $\delta(c)$ *is a constraint decomposition of* c.

3 Global Constraints

We now propose three notions of globality: semantic globality, operational globality, and algorithmic globality. Semantic globality is the stronger notion (i.e., it implies the two others) but it does not completely capture what is generally understood as "global" (at least, in our opinion). Operational globality, which considers the quality of the filtering, implies algorithmic globality.

Definition 6 (Semantic Globality). *A constraint* R *is* ***semantically global*** *if there exists no constraint decomposition scheme for* R.

Operational globality considers both a constraint R and a consistency notion Φ. The constraint is said to be "global" if there exists no decomposition scheme for which the consistency notion removes as many local inconsistencies as on the original constraint. This concept is important because it compares the pruning of the constraint and its decompositions wrt a consistency notion. (In the following, $\Phi(N)$ denotes the closure of the network N wrt to Φ.)

Definition 7 (Operational Globality). *A constraint* R *is* ***operationally*** Φ***-global*** *if there exists no constraint decomposition scheme* δ *for* R *such that* $D_{\Phi(c)} = D_{\Phi(\delta(c))}$ *for all instances* c *of* R.

Example 4. The *alldiff* constraint is operationally AC-global. Examples 2 and 3 show an instance of a constraint for which there exists no decomposition on which arc consistency performs the same amount of filtering.

When a constraint R is not operationally global wrt a local consistency Φ, this means that, from a pruning standpoint, there is no advantage in using R in a problem formulation on which Φ is the consistency notion used. However, it can be argued that it is still beneficial to consider R wrt Φ if this provides complexity advantages. This justifies algorithmic globality (see [1]). We present its definition in the section on languages, where it is easier to introduce.

4 Constraint Programming Languages

Constraint languages raise interesting issues because they have a fixed vocabulary for expressing constraints and domains. In addition, constraint languages have specific encodings of constraints and domains, which makes it easier to discuss some complexity notions which are necessarily more abstract in the CSP community.

Definition 8 (Constraint Language). *A **constraint language** L is a triplet (L^C, L^D, L^ϵ) where L^C is the set of constraints supported in L, L^D is the set of domains supported in L, and L^ϵ is an encoding scheme which specifies how constraints and domains are represented in L. For simplicity, we often use $L(R)$ and $L(d)$ to denote $L^\epsilon(R)$ and $L^\epsilon(d)$. We also use $\|L(R)\|$ and $\|L(d)\|$ to represent the size of the encoding of a constraint R and of a domain d in L.*

We now define which constraint networks can be expressed in a language. The extension of decomposition and decomposition schemes definitions follow immediately.

Definition 9 (Language Embedding). *Let L be a constraint language and N be a constraint network. N is **embedded** in L if $\forall\, d \in D_N : d \in L^D$ and $\forall\, c \in C_N : R_c \in L^C$. If N is embedded in L, $L(N)$ denotes its encoding in L. The size of the encoding $L(N)$, denoted by $\|L(N)\|$, is defined as*

$$\sum_{c \in C_N} \|L(R_c)\| \;+\; \sum_{x \in X_N} \|L(D_N(x))\|.$$

Definition 10 (Language Decomposition). *Let L be a constraint language, c be a constraint instance, and N be a network. N is a **L-decomposition** of c if $X_N = X_c$, $D_N = D_c$, $c \notin C_N$, $sol(N) = sol(c)$, and N is embedded in L.*

4.1 Globality in Languages

We are now in position to define globality in the context of constraint languages. The first two notions, semantic globality and operational globality, are direct generalizations of the CSP case. Algorithmic globality is defined in terms of the size of the encodings, which captures the fact that domains and constraints are encoded, sometimes very efficiently, in constraint languages.

Definition 11 (Algorithmic Globality in a Language). *Assume that a consistency notion Φ can be enforced in time $O(f(\|L(c)\|))$ and space $O(g(\|L(c)\|))$ on all instances c of a constraint R. R is **algorithmically Φ-global** wrt L if there exists no L-decomposition scheme δ for R such that, for all instances c of R,*

1. $D_{\Phi(c)} = D_{\Phi(\delta(c))}$;
2. *Φ can be enforced in time $O(f(\|L(c)\|))$ and space $O(g(\|L(c)\|))$ on $L(\delta(c))$;*
3. *$\|L(\delta(c))\|$ is $O(g(\|L(c)\|))$.*

The first two conditions are natural: the decomposition should preserve the pruning (1) and the complexity bounds (2). The third condition imposes a bound on the space complexity on the decompositions. This condition is critical to reflect the actual space complexity of the decomposition, since the consistency algorithm receives $\delta(c)$ as an input.

4.2 Strong Globality

Constraint programmers, or implementations of constraint programming systems, often rewrite complex constraints in terms of simpler ones by introducing new variables. This section generalizes the concepts to accommodate this important technique.

Definition 12 (Constraint Rewriting). *A constraint network N is a **rewriting** of a constraint instance c if $X_c \subseteq X_N$, $D_N[X_c] = D_c$, $c \notin C_N$, and $sol(N)[X_c] = sol(c)$.*

Example 5. Let $y = 4 \cdot x$ be a constraint instance on the variables x and y having domains $D_x = D_y = 1..10$. The constraint network involving x, y, and the additional variable z with $D_z = 1..10$, on which we post the constraint instances $z = 2 \cdot x$ and $y = 2 \cdot z$ is a rewriting of $y = 4 \cdot x$.

A language rewriting (*L-rewriting*) is simply a constraint rewriting which can be embedded in the language. The notions of strong globality are direct generalizations of the notions of globality, where L-decompositions are replaced by L-rewritings.

5 Illustrations

Example 6 (The Sum Constraint). Consider the language L containing constraints of the form $x_1 + x_2 = y$ and the $(n+1)$-ary sum constraint $\sum_{i \in 1..n} x_i = y$, $n > 2$. sum does not allow any decomposition scheme. There is no way to represent it with smaller arity constraints on the same variables. Hence it is semantically global (and thus operationally and algorithmically global). But sum is not strongly semantically global wrt L. Indeed, it can be rewritten by adding $n-2$ additional variables z_j that represent the sum of the j first x_i's in the following way: $x_1 + x_2 = z_2, z_2 + x_3 = z_3, \ldots, z_{n-1} + x_n = y$. It is not strongly operationally AC-global wrt L, since arc consistency on the rewriting removes the same values in the original variable domains as the original constraint. But the sum constraint is strongly algorithmically AC-global wrt L. Indeed, on the one hand, in order to enforce arc consistency on the rewriting,

the domain sizes on the intermediary variables may become exponential in the sizes of the original domains (either in the rewriting or during the consistency algorithm). (If $D_{x_1} = \{0, 10, 20, \ldots, 90\}$ and $D_{x_2} = \{0, 1, 2, \ldots, 9\}$, $x_1 + x_2$ takes values in $\{0, 1, \ldots, 99\}$.) On the other hand, there exists an AC algorithm which runs on sum in linear space wrt the initial domains and in time $O(s^n)$, where s is the size of the largest domain. Interestingly, the sum constraint is not strongly algorithmically BC-global, since only intervals are needed to compute bound consistency.

Acknowledgements

We would like to thank A. Aggoun, F. Benhamou, and E. Bourreau for the discussions we had on this topic.

References

1. C. Bessière and P. Van Hentenryck. To be or not to be ... a global constraint. Technical Report 03050, LIRMM – University of Montpellier II, Montpellier, France, June 2003. (available at http://www.lirmm.fr/~bessiere/).