Reformulation of Global Constraints in Answer Set Programming

Christian Drescher
Vienna University of Technology
Austria

Toby Walsh
NICTA and University of New South Wales
Australia

Abstract

We show that global constraints on finite domains like *all-different* can be reformulated into answer set programs on which we achieve arc, bound or range consistency. These reformulations offer a number of other advantages beyond providing the power of global propagators to answer set programming. For example, they provide other constraints with access to the state of the propagator by sharing variables. Such sharing can be used to improve propagation between constraints. Experiments with these encodings demonstrate their promise.

Introduction

There are several approaches to representing and solving constraint satisfaction problems: constraint programming (CP; Rossi, van Beek, and Walsh 2006), answer set programming (ASP; Baral 2003), propositional satisfiability checking (SAT; Biere et al. 2009), its extension to satisfiability modulo theories (SMT; Nieuwenhuis, Oliveras, and Tinelli 2006), and many more. Each has its particular strengths: for example, CP systems support global constraints, ASP systems permit recursive definitions and offer default negation, whilst SAT solvers often exploit very efficient implementations. In many applications it would often be helpful to exploit the strengths of multiple approaches. Consider the problem of timetabling at an university (Järvisalo et al. 2009). To model the problem, we need to express the mutual exclusion of events (for instance, we cannot place two events in the same room at the same time). A straightforward representation of such constraint with clauses and rules uses quadratic space. In contrast, global constraints such as *all-different* typically supported by CP systems can give a much more concise encoding. On the other hand, there are features which are hard to describe in traditional constraint programming, like the temporary unavailability of a particular room. However, this is easy to represent with non-monotonic rules such as those used in ASP. Such rules also provide a flexible mechanism for defining new relations on the basis of existing ones.

Answer set programming has been put forward as a powerful paradigm to solve constraint satisfaction problems in Niemelä (1999), which shows that ASP embeds SAT but provides a more expressive framework from a knowledge representation point of view. Moreover, modern ASP solvers compete[1] with the best SAT solvers. An empirical comparison of the performance of ASP and constraint logic programming (CLP; Jaffar and Maher 1994) systems on solving combinatorial problems conducted by Dovier, Formisano, and Pontelli shows ASP encodings to be more compact, more declarative, and highly competitive. However, as some CSP are more naturally modelled by using non-propositional constructs, like resources or functions over finite domains, and by using global constraints in particular, there is an increasing desire to handle constraints beyond pure ASP.

One approach to combining ASP and CSP is to integrate theory-specific predicates into propositional formulas (motivated by SMT), and to extend the ASP solver's decision engine with a higher level proof procedure (Baselice, Bonatti, and Gelfond 2005; Mellarkod and Gelfond 2008; Gebser, Ostrowski, and Schaub 2009). However, the resulting systems have a number of limitations. First, they are tied to particular ASP and CP solvers. Second, the support for global constraints is limited. Third, communication between the ASP and CP solver is restricted. Alternative techniques, such as reformulating constraints into ASP received little attention. The key contribution of our work is an investigation of reformulation in the context of answer set programming, illustrated by reformulations of the popular *all-different* constraint. The resulting approach has been implemented in the new preprocessor *inca*. Empirical evaluation demonstrates its computational potential.

Background

Answer Set Programming A *(normal) logic program* Π over a set of primitive propositions $\mathcal{A}$ is a finite set of rules of the form $a_0 \leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n$ where $0 \leq m \leq n$ and $a_i \in \mathcal{A}$ is an *atoms* for $0 \leq i \leq n$. A *literal* $\hat{a}$ is an atom a or its default negation $not\ a$. For a rule r, let $head(r) = a_0$ be the *head* of r and $body(r) = \{a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n\}$ the *body* of r. The set of atoms occurring in a logic program Π is denoted by $atom(\Pi)$, and the set of bodies in Π is $body(\Pi) = \{body(r) \mid r \in \Pi\}$. For regrouping bodies sharing the same

[1] `http://www.satcompetition.org`

head a, define $body(a) = \{body(r) \mid r \in \Pi,\ head(r) = a\}$. The semantics of a logic program is given by its answer sets, being total well-founded models of Π. For a formal introduction to ASP, we refer the reader to Baral (2003). The semantics of important extensions to logic programs, such as choice rules, integrity, and cardinality constraints, is given through program transformations that introduce additional propositions (cf. Simons, Niemelä, and Soininen 2002). A *choice rule* allows for the non-deterministic choice over atoms in $\{a_1, \ldots, a_k\}$ and has the form $\{a_0, \ldots, a_k\} \leftarrow a_{k+1}, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n$. An *integrity constraint* of the form $\leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n$ is a short hand for a rule with an unsatisfiable head, and thus forbids its body to be satisfied in any answer set. A *cardinality constraint* of the form $\leftarrow k\{\hat{a}_1, \ldots, \hat{a}_n\}$ is interpreted as no k literals of the set $\{\hat{a}_1, \ldots, \hat{a}_n\}$ are included in an answer set. Simons, Niemelä, and Soininen provide a transformation that needs just $\mathcal{O}(nk)$ rules, introducing atoms $l(\hat{a}_i, j)$ to represent the fact that at least j of the literals with index $\geq i$, i.e. the literals in $\{\hat{a}_i, \ldots, \hat{a}_n\}$, are in a particular answer set candidate. Then, the cardinality constraint can be encoded by an integrity constraint $\leftarrow l(\hat{a}_1, k)$ and the three following rules, where $1 \leq i \leq n$ and $1 \leq j \leq k$:

$$l(\hat{a}_i, j) \leftarrow l(\hat{a}_{i+1}, j) \qquad l(\hat{a}_i, j+1) \leftarrow \hat{a}_i, l(\hat{a}_{i+1}, j)$$
$$l(\hat{a}_i, 1) \leftarrow \hat{a}_i$$

Nogoods of Logic Programs We want to view inferences in ASP as unit-propagation on nogoods. Following Gebser et al. (2007), inferences in ASP rely on atoms and program rules, which can be expressed by using atoms and bodies. Thus, for a program Π, the *domain* of Boolean assignments $\mathbf{A}$ is fixed to $dom(\mathbf{A}) = atom(\Pi) \cup body(\Pi)$.

Formally, a Boolean *assignment* $\mathbf{A}$ is a set $\{\sigma_1, \ldots, \sigma_n\}$ of *signed literals* σ_i for $1 \leq i \leq n$ of the form $\mathbf{T}a$ or $\mathbf{F}a$ where $a \in dom(\mathbf{A})$. $\mathbf{T}a$ expresses that a is assigned *true* and $\mathbf{F}a$ that it is *false* in $\mathbf{A}$. (We omit the attribute *Boolean* for assignments whenever clear from the context.) The complement of a signed literal σ is denoted by $\overline{\sigma}$, that is $\overline{\mathbf{T}a} = \mathbf{F}a$ and $\overline{\mathbf{F}a} = \mathbf{T}a$. In the context of ASP, a *nogood* is a set $\delta = \{\sigma_1, \ldots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment $\mathbf{A}$ such that $\delta \subseteq \mathbf{A}$. For a nogood δ, a signed literal $\sigma \in \delta$, and an assignment $\mathbf{A}$, we say that δ is *unit* and $\overline{\sigma}$ is *unit-resulting* if $\delta \setminus \mathbf{A} = \{\sigma\}$. Let $\mathbf{A}^{\mathbf{T}} = \{a \in dom(\mathbf{A}) \mid \mathbf{T}a \in A\}$ the set of true propositions and $\mathbf{A}^{\mathbf{F}} = \{a \in dom(\mathbf{A}) \mid \mathbf{F}a \in A\}$ the set of false propositions. A *total* assignment, that is $\mathbf{A}^{\mathbf{T}} \cup \mathbf{A}^{\mathbf{F}} = dom(\mathbf{A})$ and $\mathbf{A}^{\mathbf{F}} \cup \mathbf{A}^{\mathbf{T}} = \emptyset$, is a *solution* for a set Δ of nogoods if $\delta \notin \mathbf{A}$ for all $\delta \in \Delta$.

As shown in Lee (2005), the answer sets of a logic program Π correspond to the models of the completion of Π that satisfy the loop formulas of all non-empty subsets of $atom(\Pi)$. For $\beta = \{a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n\} \in body(\Pi)$, define

$$\Delta_\beta = \left\{ \begin{array}{l} \{\mathbf{T}a_1, \ldots, \mathbf{T}a_m, \mathbf{F}a_{m+1}, \ldots \mathbf{F}a_n, \mathbf{F}\beta\}, \\ \{\mathbf{F}a_1, \mathbf{T}\beta\}, \ldots, \{\mathbf{F}a_m, \mathbf{T}\beta\}, \\ \{\mathbf{T}a_{m+1}, \mathbf{T}\beta\}, \ldots, \{\mathbf{T}a_n, \mathbf{T}\beta\} \end{array} \right\}.$$

Intuitively, the nogoods in Δ_β enforce the truth of body β iff all its literals are satisfied. For an atom $a \in atom(\Pi)$ with $body(a) = \{\beta_1, \ldots, \beta_k\}$, let

$$\Delta_a = \left\{ \begin{array}{l} \{\mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k, \mathbf{T}a\}, \\ \{\mathbf{T}\beta_1, \mathbf{F}a\}, \ldots, \{\mathbf{T}\beta_k, \mathbf{F}a\} \end{array} \right\}.$$

Then, the solutions for $\Delta_\Pi = \bigcup_{\beta \in body(\Pi)} \Delta_\beta \cup \bigcup_{a \in atom(\Pi)} \Delta_a$ correspond to the models of the completion of Π. Loop formulas, expressed in the set of nogoods Λ_Π, have to be added to establish full correspondence to the answer sets of Π. Typically, solutions for $\Delta_\Pi \cup \Lambda_\Pi$ are computed by applying *conflict-driven nogood learning* (CDNL; Gebser et al. (2007)). This combines search and propagation by recursively assigning the value of a proposition and using *unit-propagation* to determine logical consequences of an assignment (Mitchell 2005).

Constraint Satisfaction Problem The classic definition of a constraint satisfaction problem is as follows (cf. Rossi, van Beek, and Walsh 2006). A CSP is a triple (V, D, C) where V is an n-tuple of *variables* $V = (v_1, \ldots, v_n)$, D is an n-tuple of finite *domains* $D = (D_1, \ldots, D_n)$ such that each variable v_i has an associated domain $dom(v_i) = D_i$, and C is a set of *constraints*. A constraint c is a pair (R_S, S) where R_S is a k-ary *relation* on the variables in $S \subseteq V^k$, called the *scope* of c. In other words, R_S is a subset of the Cartesian product of the domains of the variables in S. To access the relation and the scope of c define $range(c) = R_S$ and $scope(c) = S$. For a *(constraint variable) assignment* $A : V \to \bigcup_{v \in V} dom(v)$ and a constraint $c = (R_S, S)$ with $S = (v_1, \ldots, v_k)$, define $A(S) = (A(v_1), \ldots, A(v_k))$, and call c *satisfied* if $A(S) \in range(c)$. Given this, define the set of constraints satisfied by A as $sat_C(A) = \{c \mid A(scope(c)) \in range(c),\ c \in C\}$.

A binary constraint c has $|scope(c)| = 2$. For example, $v_1 \neq v_2$ ensures that v_1 and v_2 take different values. A global (or n-ary) constraint c has parametrized scope. For example, the *all-different* constraint ensures that a set of variables, $\{v_1, \ldots, v_n\}$ take all different values. This can be decomposed into $O(n^2)$ binary constraints, $v_i \neq v_j$ for $i < j$. However, as we shall see, such reformulation can hinder inference. An assignment A is a *solution* for a CSP iff it satisfies all constraints in C.

Constraint solvers typically use backtracking search to explore the space of partial assignments. Various heuristics affecting, for instance, the variable selection criteria and the ordering of the attempted values, can be used to guide the search. Each time a variable is assigned a value, a deterministic propagation stage is executed, pruning the set of values to be attempted for the other variables, i.e., enforcing a certain type of local consistency.

A binary constraint c is called *arc consistent* iff when a variable $v_1 \in scope(c)$ is assigned any value $d_1 \in dom(v_1)$, there exists a consistent value $d_2 \in dom(v_2)$ for the other variable v_2. An n-ary constraint c is *hyper-arc consistent* or *domain consistent* iff when a variable $v_i \in scope(c)$ is assigned any value $d_i \in dom(v_i)$, there exist compatible values in the domains of all the other variables $d_j \in dom(v_j)$ for all $1 \leq j \leq n,\ j \neq i$ such that $(d_1, \ldots, d_n) \in range(c)$.

Relational consistency (Dechter and van Beek 1997) extends the concept of local consistency. I.e. a constraint c is *relationally k-arc consistent* if any consistent assignment of a k-elementary subset of variables from $scope(c)$ extends to a consistent assignment of all variables in $scope(c)$.

The concepts of bound and range consistency are defined for constraints on ordered intervals. Let $min(D_i)$ and $max(D_i)$ be the minimum value and maximum value of the domain D_i. A constraint c is *bound consistent* iff when a variable v_i is assigned $d_i \in \{min(dom(v_i)), max(dom(v_i))\}$ (i.e. the minimum or maximum value in its domain), there exist compatible values between the minimum and maximum domain value for all the other variables in the scope of the constraint. Such an assignment is called a *bound support*. A constraint is *range consistent* iff when a variable is assigned any value in its domain, there exists a bound support. Notice that range consistency is in between domain and bound consistency, where domain consistency is the strongest of the three formalisms.

Encoding Global Constraints in ASP

In this section we explain how to reformulate multi-valued variables and constraints on finite domains into a logic program under answer set semantics. In what follows, we assume $dom(v) = [1, d]$ for all $v \in V$ to save the reader from multiple superscripts.

Direct Encoding A popular choice is called the *direct encoding* (Walsh 2000). In the direct encoding, a propositional variable $e(v, i)$, representing $v = i$, is introduced for each value i that can be assigned to the constraint variable v. Intuitively, the proposition $e(v, i)$ is true if v takes the value i, and false if v takes a value different from i. For each v, the truth-assignments of atoms $e(v, i)$ are encoded by a choice rule (1). Furthermore, there is an integrity constraint (2) to ensure that v takes at least one value, and a cardinality constraint (3) that ensures that v takes at most one value.

$$\{e(v,1),\ldots,e(v,d)\} \leftarrow \quad (1)$$

$$\leftarrow not\ e(v,1),\ldots,not\ e(v,d) \quad (2)$$

$$\leftarrow 2\ \{e(v,1),\ldots,e(v,d)\} \quad (3)$$

In the direct encoding, each forbidden combination of values in a constraint is expressed by an integrity constraint. On the other hand, when a relation is represented by allowed combinations of values, all forbidden combinations have to be deduced and translated to integrity constraints. Unfortunately, the direct encoding of constraints hinders propagation:

Theorem 1 *Enforcing arc consistency on the binary decomposition of the original constraint prunes more values from the variables domain than unit-propagation on its direct encoding.*

Support Encoding The *support encoding* has been proposed to tackle this weakness (Gent 2002). A *support* for a constraint variable v to take the value i across a constraint c is the set of values $\{i_1,\ldots,i_m\} \subseteq dom(v')$ of another variable in $v' \in scope(c) \setminus \{v\}$ which allow $v = i$, and can be encoded as follows, extending (1–3):

$$\leftarrow e(v,i), not\ e(v',i_1),\ldots,not\ e(v',i_m)$$

This integrity constraint can be read as whenever $v = i$, then at least one of its supports must hold. In the support encoding, for each constraint c there is one support for each pair of distinct variables $v, v' \in scope(c)$, and for each value i.

Theorem 2 *Unit-propagation on the support encoding enforces arc consistency on the binary decomposition of the original constraint.*

We illustrate this approach on an encoding of the global *all-different* constraint. For variables v, v' and value i it can be reduced from the definition by using the equivalence covered by (2–3) to

$$\leftarrow e(v,i), e(v',i).$$

Observe, that this is also the direct encoding of the binary decomposition of the global *all-different* constraint. However, this observation does not hold in general for all constraints. As discussed in the Background section of this paper, we can express above condition as $\mathcal{O}(d)$ cardinality constraints:

$$\leftarrow 2\ \{e(v_1,i),\ldots,e(v_n,i)\} \quad (4)$$

Corollary 2.1 *Unit-propagation on (1–4) enforces arc consistency on the binary decomposition of the global* all-different *constraint in $\mathcal{O}(nd^2)$ down any branch of the search tree.*

k-support Encoding The support encoding can be generalized to the *k-support encoding* (Bessière, Hebrard, and Walsh 2003) representing supports on subsets of $scope(c)$ for an assignment of another k-elementary subset of variables in $scope(c)$. More formal, a k-support S for an assignment A of k variables from $scope(c)$, say $v_1 = i_1,\ldots,v_k = i_k$, is an assignment $v'_1 = i'_1,\ldots,v'_l = i'_l$ such that $\{v'_1,\ldots,v'_l\} \subseteq scope(c) \setminus \{v_1,\ldots,v_k\}$ which allows A. We introduce a *support-variable* s, that evaluates to true iff S holds:

$$s \leftarrow e(v'_1,i_1),\ldots,e(v'_l,i'_l)$$

Furthermore, let $\{S_1,\ldots,S_m\}$ be the set of all k-supports of A. A k-support rule for A is defined as

$$\leftarrow e(v_1,d_1),\ldots,e(v_k,d_k), not\ s_1,\ldots,not\ s_m$$

meaning that as long as A holds then at least one of its k-supports $S_1,\ldots S_m$ must hold. In the k-support encoding, for each constraint c there is one k-support rule for each assignment A of k variables from $scope(c)$.

Theorem 3 *Unit-propagation on the k-support encoding enforces relational k-arc consistency on the original constraint.*

Range Encoding In the *range encoding*, a propositional variable $r(v, l, u)$ is introduced for all $[l, u] \subseteq [1, d]$ to represent whether the value of v is between l and u. For each range $[l, u]$, the following $\mathcal{O}(nd^2)$ rules encode $v \in [l, u]$ whenever it is safe to assume that $v \notin [1, l-1]$ and

$v \notin [u+1, d]$, and enforce a consistent set of ranges such that $v \in [l, u] \Rightarrow v \in [l-1, u] \wedge v \in [l, u+1]$:

$$r(v, l, u) \leftarrow not\ r(v, 1, l-1), not\ r(v, u+1, d) \quad (5)$$
$$\leftarrow r(v, l-1, u), not\ r(v, l, u) \quad (6)$$
$$\leftarrow r(v, l, u+1), not\ r(v, l, u) \quad (7)$$

Constraints are encoded into integrity constraints representing conflict regions. When the combination $v_1 \in [l_1, u_1], \ldots, v_n \in [l_n, u_n]$ violates the constraint, the following rule is added:

$$\leftarrow r(v_1, l_1, u_1), \ldots, r(v_n, l_n, u_n)$$

Theorem 4 *Unit-propagation on the range encoding enforces range consistency on the original constraint.*

A propagator for the global *all-different* constraint that enforces range consistency pruning Hall intervals has been proposed by Leconte (1996) and encoded to SAT by Bessière et al. (2009). An interval $[l, u]$ is a *Hall interval* iff $|\{v \mid dom(v) \subseteq [l, u]\}| = u - l + 1$. In other words, a Hall interval of size k completely contains the domains of k variables. Observe that in any bound support, the variables whose domains are contained in the Hall interval consume all values within the Hall interval, whilst any other variable must find their support outside the Hall interval. The following reformulation of the global *all-different* constraint will permit us to achieve range consistency via unit propagation. It ensures that no interval $[l, u]$ can contain more variables than its size.

$$\leftarrow u - l + 2\ \{r(v_1, l, u), \ldots, r(v_n, l, u)\} \quad (8)$$

This simple reformulation can simulate a complex propagation algorithm like Leconte's with a similar overall complexity of reasoning.

Corollary 4.1 *Unit-propagation on (5–8) enforces range consistency on the global* all-different *constraint in $\mathcal{O}(nd^3)$ down any branch of the search tree.*

A hybrid that links the range encoding of v to its direct representation extends the range encoding as follows, for each $i \in dom(v)$:

$$e(v, i) \leftarrow r(v, i, i)$$
$$\leftarrow e(v, i), not\ r(v, i, i)$$

These rules encode the equivalence $v = i \Leftrightarrow v \in [i, i]$.

Bound Encoding A last encoding is called the *bound encoding* (Crawford and Baker 1994). In the bound encoding, a propositional variable $b(v, i)$ is introduced for each value i to represent that the value of v is bounded by i. That is, $v \leq i$ if $b(v, i)$ is assigned *true*, and $v > i$ if $b(v, i)$ is assigned *false*. Similar to the direct encoding, for each v, the truth-assignments of atoms $b(v, i)$ are encoded by a choice rule (9). In order to ensure that assignments represent a consistent set of bounds, the condition $v \leq i \Rightarrow v \leq i+1$ is posted as integrity constraints (10) $\forall i \in [1, d-1]$. Another integrity constraint (11) encodes $v \leq d$, that at least one value must be assigned to v:

$$\{b(v, 1), \ldots, b(v, d)\} \leftarrow \quad (9)$$
$$\leftarrow b(v, i), not\ b(v, i+1) \quad (10)$$
$$\leftarrow not\ b(v, d) \quad (11)$$

Constraints are encoded into integrity constraints representing conflict regions similar to the range encoding. When all combinations in the region

$$l_1 < v_1 \leq u_1, \ldots, l_n < v_n \leq u_n$$

violate a constraint, the following rule is added:

$$\leftarrow b(v_1, u_1), \ldots, b(v_n, u_n), not\ b(v_1, l_1), \ldots, not\ b(v_n, l_n)$$

Theorem 5 *Unit-propagation on the bound encoding enforces bound consistency on the original constraint.*

In order to get a representation of the global *all-different* constraint that can only prune bounds, the bound encoding for variables is linked to (8) as follows:

$$r(v, l, u) \leftarrow not\ b(v, l-1), b(v, u) \quad (12)$$
$$\leftarrow r(v, l, u), b(v, l-1) \quad (13)$$
$$\leftarrow r(v, l, u), not\ b(v, u) \quad (14)$$

Corollary 5.1 *Unit-propagation on (8–14) enforces bound consistency on the global* all-different *constraint in $\mathcal{O}(nd^2)$ down any branch of the search tree.*

Note that an upper bound h can be posted on the size of Hall intervals. The resulting encoding with only those cardinality constraints (5) for which $u - l + 1 \leq h$ detects Hall intervals of size at most h, and therefore enforces a weaker level of consistency.

To access the value of v, the bound encoding can be extended to a hybrid by adding the following rules to the bound encoding for each $i \in [1, d]$:

$$e(v, i) \leftarrow b(v, i), not\ b(v, i-1)$$
$$\leftarrow e(v, i), not\ b(v, i)$$
$$\leftarrow e(v, i), b(v, i-1)$$

The first rule enforces $e(v, i)$ to be true if possible values for v are bound to the singleton i, i.e. $v \leq i$ and $v \not\leq i-1$ are in the assignment. On the other hand, the condition $v = i \Rightarrow v \leq i \wedge v \not\leq i-1$ is represented as integrity constraints.

Non-ground Logic Programs Although our semantics is propositional, atoms in $\mathcal{A}$ and can be constructed from a first-order signature $\Sigma = (\mathcal{F}, \mathcal{V}, \mathcal{P})$, where $\mathcal{F}$ is a set of function symbols (including constant symbols), $\mathcal{V}$ is a denumerable collection of first-order variables, and $\mathcal{P}$ is a set of predicate symbols. The logic program over $\mathcal{A}$ is then obtained by a grounding process, systematically substituting all occurrences of variables $\mathcal{V}$ by terms in $\mathcal{T}(\mathcal{F})$, where $\mathcal{T}(\mathcal{F})$ denotes the set of all ground terms over $\mathcal{F}$. Atoms in $\mathcal{A}$ are formed from predicate symbols $\mathcal{P}$ and terms in $\mathcal{T}(\mathcal{F})$.

Experiments

To evaluate these reformulations, we conducted experiments on encodings of CSPs containing *all-different* and *permutation* constraints. The global *permutation* constraint is a special case of *all-different* when the number of variables is equal to the number of all their possible values. A reformulation of *permutation* extends (4) by

$$\leftarrow not\ e(v_1, i), \ldots, not\ e(v_n, i)$$

or (8) by the following rule where $1 \leq l \leq u \leq k$:

$$\leftarrow d - u + l\ \{not\ r(v_1, l, u), \ldots, not\ r(v_n, l, u)\}$$

This can increase propagation. Our reformulations have been implemented within the prototypical preprocessor *inca*[2] which compiles an (extended) logic programs with high-level statements for global constraints, constraint variables, first-order variables, function symbols, and aggregates, etc. in linear time and space, such that the logic program can be obtained by a *grounding* process. Experiments consider *inca* in different settings using different reformulations. We denote the support encoding of the global constraints by S, the bound encoding of the global constraints by B, and the range encoding of the global constraints by R. To explore the impact of small Hall intervals, we also tried B_k and R_k, an encoding of the global constraints with only those cardinality constraints (8) for which $u - l + 1 \leq k$. The consistency achieved by B_k and R_k is therefore weaker than full bound and range consistency, respectively.

We also include the pure CP system *gecode*[3] (3.2.0), and the integrated system *ezcsp*[4] (1.6.9; Balduccini 2009) in our empirical analysis. The latter combines the grounder *gringo* (2.0.3) and ASP solver *clasp* (1.3.0) with *sicstus*[5] (4.0.8) as a constraint solver. Since *inca* is a pure preprocessor, we select the ASP system *clingo* (2.0.3) as its backend to provide a representative comparison with *ezcsp*. Note that *clingo* stands for *clasp* on *gringo* and combines both systems in a monolithic way. All experiments were run on a 2.00 GHz PC under Linux. We report results in seconds, where each run was limited to 600 s time and 1 GB RAM.

Pigeon Hole Problem The *pigeon hole problem* (PGP) is to show that it is impossible to put n pigeons into $n - 1$ holes if each pigeon must be put into a distinct hole. Clearly, our bound and range reformulations are faster compared to weaker encodings (see Table 1). On such problems, detecting large Hall intervals is essential.

Quasigroup Completion A *quasigroup* is an algebraic structure $(Q, \cdot)$, where Q is a set and $\cdot$ is a binary operation on Q such that for every pair of elements $a, b \in Q$ there exist unique elements $x, y \in Q$ which solve the equations $a \cdot x = b$ and $y \cdot a = b$. The *order* n of a quasigroup is defined by the number of elements in Q. A quasigroup can be represented by an $n \times n$-multiplication table, where for each pair a, b the table gives the result of $a \cdot b$, and it defines a *Latin square*. This means that each element of Q occurs exactly once in each row and each column of the table. The *quasigroup completion problem* (QCP) is to determine weather a partially filled table can be completed in such a way that a multiplication table of a quasigroup is obtained. Randomly generated QCP has been proposed as a benchmark domain for CP systems by Gomes and Selman since it combines the features of purely random problems and highly structured problems. Table 2 compares the runtime for solving QCP problems of size 20×20 where the first column gives the percentage of preassigned values. We included *gecode* with algorithms that enforce bound and domain consistency, denoted as $gecode_B$ and $gecode_D$ (not shown), in the experiments. Our analysis exhibits phase transition behaviour of the systems *ezcsp*, *gecode*, and $gecode_B$, while our Boolean encodings and $gecode_D$ solve all problems within seconds. Interestingly, learning constraint interdependencies as in our approach is sufficient to tackle QCP. In fact, most of the time for S, B_k, R_k is spent on grounding, but not for solving the actual problem.

n	S	B_3	B	R_3	R	*ezcsp*	*gecode*
10	5	**<1**	**<1**	<1	**<1**	2	**<1**
11	46	1	**<1**	2	**<1**	17	9
12	105	4	**<1**	3	**<1**	184	104
13	—	25	**<1**	30	**<1**	—	—
14	—	125	**<1**	197	**<1**	—	—
15	—	—	**<1**	—	**<1**	—	—
16	—	—	**<1**	—	**<1**	—	—

Table 1: Runtime results in seconds for PHP.

%	S	B_3	B	R	*ezcsp*	*gecode*	$gecode_B$
10	**3**	5	8	7	30 (7)	2 (4)	<1 (1)
20	**2**	5	8	7	21 (20)	5 (4)	<1 (3)
30	**2**	5	8	7	10 (30)	3 (13)	1 (5)
35	**2**	5	8	7	22 (24)	14 (13)	6 (7)
40	**2**	5	8	7	52 (29)	12 (20)	6 (9)
45	**2**	5	8	7	36 (35)	18 (25)	6 (13)
50	**2**	5	8	7	36 (50)	25 (32)	6 (18)
55	**2**	4	8	7	61 (51)	20 (41)	31 (29)
60	**2**	4	8	7	60 (63)	36 (51)	27 (35)
70	**2**	4	7	6	70 (66)	28 (45)	17 (27)
80	**2**	4	7	5	16 (18)	17 (13)	7 (7)
90	2	4	7	5	**1**	<1 (1)	3

Table 2: Average times over 100 runs on QCP. Timeouts are given in parenthesis, if any.

Graceful Graphs A labelling f of the nodes of a graph (V, E) is *graceful* if f assigns a unique label $f(v)$ from $\{0, 1, \ldots, |E|\}$ to each node $v \in V$ such that, when each edge $(v, w) \in E$ is assigned the label $|f(v) - f(w)|$, the resulting edge labels are distinct. The problem of determining the existence of a graceful labelling of a graph (GGP) has been modelled as a CSP in Petrie and Smith (2003). Our

[2] `http://potassco.sourceforge.net/` provides the systems *clasp*, *clingo*, *gringo*, *inca*, and the benchmark set

[3] `http://www.gecode.org/`

[4] `http://krlab.cs.ttu.edu/~marcy/ezcsp/`

[5] `http://www.sics.se/sicstus/`

n	S	B_1	B_3	B	R	*ezcsp*	*gecode*
3	3	9	12	12	11	6	**2**
5	**<1**	**<1**	**<1**	2	3	1	**<1**
6	**<1**	1	1	2	5	1	7
7	**1**	6	7	32	19	18	—
8	2	12	**1**	**1**	10	4	—
9	**2**	5	17	103	15	390	—
10	**3**	91	10	105	110	70	—
11	**8**	22	296	—	223	—	—

Table 3: Runtime results in seconds for GGP.

experiments consider double-wheel graphs DW_n composed by two copies of a cycle with n vertices, each connected to a central hub. Table 3 shows that our encodings outperform all comparable systems, where the support encoding performs better than bound and range encodings. In most cases, the branching heuristic used in our approach appears to be misled by the extra variables introduced in B_k and R_k. That explains some of the variability in the runtimes.

Conclusions

We have reformulated global and other constraints into answer set programs. In particular, we have investigated various generic ASP encodings for constraints on finite domains and proved which level of consistency unit-propagation achieves on them. Our techniques were formulated as preprocessing and can be applied to any ASP system without changing its source code, which allows for programmers to select the ASP solver that best fit their needs. We have empirically evaluated the performance of such an approach on benchmarks from CP and found that such reformulations outperform integrated ASP(CP) systems as well as pure CP solvers. Our future works includes the reformulation of other useful global constraints into answer set programming like the *Regular* constraint, as well as global constraints like *Lex* which are very useful for symmetry breaking .

References

Balduccini, M. 2009. CR-prolog as a specification language for constraint satisfaction problems. In *Proc. of LPNMR'09*, 402–408. Springer.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Baselice, S.; Bonatti, P.; and Gelfond, M. 2005. Towards an integration of answer set and constraint solving. In *Proc. of ICLP'05*, 52–66. Springer.

Bessière, C.; Katsirelos, G.; Narodytska, N.; Quimper, C.-G.; and Walsh, T. 2009. Decompositions of all different, global cardinality and related constraints. In *Proc. of IJCAI'09*. AAAI Press/The MIT Press.

Bessière, C.; Hebrard, E.; and Walsh, T. 2003. Local consistencies in SAT. In *Proc. of SAT'03*, 299–314. Springer.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*. IOS Press.

Crawford, J. M., and Baker, A. B. 1994. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proc. of AAAI'94*, 1092–1097.

Dechter, R., and van Beek, P. 1997. Local and global relational consistency. *Theory of Computer Science* 173(1):283–308.

Dovier, A.; Formisano, A.; and Pontelli, E. 2005. A comparison of CLP(FD) and ASP solutions to NP-complete problems. In *Proc. of ICLP'05*, 67–82. Springer.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In *Proc. of IJCAI'07*, 386–392. AAAI Press/The MIT Press.

Gebser, M.; Ostrowski, M.; and Schaub, T. 2009. Constraint answer set solving. In *Proc. of ICLP'09*, 235–249. Springer.

Gent, I. P. 2002. Arc consistency in SAT. In *Proc. of ECAI'02*, 121–125. IOS Press.

Gomes, C. P., and Selman, B. 1997. Problem structure in the presence of perturbations. In *Proc. of AAAI'97*, 221–226. AAAI Press.

Jaffar, J., and Maher, M. J. 1994. Constraint logic programming: A survey. *Journal of Logic Programming* 19/20:503–581.

Järvisalo, M.; Oikarinen, E.; Janhunen, T.; and Niemelä, I. 2009. A module-based framework for multi-language constraint modeling. In *Proc. of LPNMR'09*, 155–169. Springer.

Leconte, M. 1996. A bounds-based reduction scheme for constraints of difference. In *CP'96, Second International Workshop on Constraint-based Reasoning*.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In *Proc. of IJCAI'05*, 503–508. Professional Book Center.

Mellarkod, V., and Gelfond, M. 2008. Integrating answer set reasoning with constraint solving techniques. In *Proc. of FLOPS'08*, 15–31. Springer.

Mitchell, D. 2005. A SAT solver primer. *Bulletin of the European Association for Theoretical Computer Science* 85:112–133.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25(3-4):241–273.

Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2006. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM* 53(6):937–977.

Petrie, K. E., and Smith, B. M. 2003. Symmetry breaking in graceful graphs. In *Proc. of CP'03*, 930–934. Springer.

Rossi, F.; van Beek, P.; and Walsh, T., eds. 2006. *Handbook of Constraint Programming*. Elsevier.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2):181–234.

Walsh, T. 2000. SAT v CSP. In *Proc. of CP'00*, 441–456. Springer.