Circuit Complexity and Decompositions of Global Constraints

Christian Bessiere[*]
LIRMM, CNRS
Montpellier
bessiere@lirmm.fr

George Katsirelos[†]
NICTA
Sydney
gkatsi@gmail.com

Nina Narodytska[†]
NICTA and UNSW
Sydney
ninan@cse.unsw.edu.au

Toby Walsh[†]
NICTA and UNSW
Sydney
toby.walsh@nicta.com.au

Abstract

We show that tools from circuit complexity can be used to study decompositions of global constraints. In particular, we study decompositions of global constraints into conjunctive normal form with the property that unit propagation on the decomposition enforces the same level of consistency as a specialized propagation algorithm. We prove that a constraint propagator has a a polynomial size decomposition if and only if it can be computed by a polynomial size monotone Boolean circuit. Lower bounds on the size of monotone Boolean circuits thus translate to lower bounds on the size of decompositions of global constraints. For instance, we prove that there is no polynomial sized decomposition of the domain consistency propagator for the ALLDIFFERENT constraint.

1 Introduction

Global constraints are a vital component of constraint toolkits. They permit users to model common patterns and to exploit efficient propagation algorithms to reason about these patterns. A promising mechanism to implement such global constraints is to develop decompositions into sets of primitive constraints that do not hinder propagation. For example, Bacchus has shown how to decompose global propagators for the generic TABLE constraint, as well as for the REGULAR, AMONG and SEQUENCE constraints into conjunctive normal form (CNF) [Bacchus, 2007]. Such decompositions can then be used in SAT solvers, allowing us to profit from techniques like clause learning and backjumping. In recent years, many other decompositions have been proposed for a wide range of global constraints including REGULAR and GRAMMAR [Quimper and Walsh, 2006; 2007; 2008; Katsirelos *et al.*, 2008], SEQUENCE [Brand *et al.*, 2007], PRECEDENCE [Walsh, 2006], CARDPATH and SLIDE [Bessiere *et al.*, 2008]. Many other global constraints can be decomposed using ROOTS and RANGE, which can themselves be propagated effectively using some simple decompositions [Bessiere *et al.*, 2005; 2006a; 2006b]. Finally, many global constraints specified by automata can be decomposed into signature and transition constraints without hindering propagation [Beldiceanu *et al.*, 2005].

This raises the important open question of which global constraints can be effectively propagated using simple encodings [Bessiere and Van Hentenryck, 2003]. We show that circuit complexity can be used to resolve this question. Our main result is that there is a polynomial sized decomposition of a constraint propagator into CNF if and only if the propagator can be computed by a polynomial size monotone Boolean circuit. It follows therefore that bounds on the size of monotone Boolean circuits give bounds on the size of decompositions of global constraints into CNF. For instance, a super-polynomial lower bound on the size of a Boolean circuit for perfect matching in a bipartite graph gives a super-polynomial lower bound on the size of a CNF decomposition of the domain consistency propagator for the ALLDIFFERENT constraint. Our results directly extend to decompositions into CSP constraints of bounded arity with domains given in extension since such decompositions can be translated into clauses of polynomial size [Bessiere *et al.*, 2003]. The tools of circuit complexity are thus useful in understanding the limits of what we can achieve with decompositions.

2 Background

CSP. A constraint satisfaction problem (CSP) P consists of a set of variables $\mathbf{X}$, each of which has a finite domain $D(X_i)$, and a set of constraints $\mathcal{C}$. An *assignment* to a variable X_i is a mapping of X_i to a value $j \in D(X_i)$, called literal, and written $X_i = j$. We write $\mathbf{D}(\mathbf{X})$ (resp. $\mathbf{D'}(\mathbf{X})$) for sets of literals $\{X_i = j \mid X_i \in \mathbf{X} \wedge j \in D(X_i)\}$ (resp. $\{X_i = j \mid X_i \in \mathbf{X} \wedge j \in D'(X_i)\}$) and $\mathcal{P}(\mathbf{D})$ for the set of all such sets. An assignment to a set of variables $\mathbf{X}$ is a set that contains exactly one assignment to each variable in $\mathbf{X}$. A constraint $C \in \mathcal{C}$ has a *scope*, denoted $scope(C) \subseteq \mathbf{X}$ and allows a subset of the possible assignments to the variables $scope(C)$, called *solutions* of C. A solution of P is an assignment of one value to each variable such that all constraints are satisfied.

A propagator for a constraint C is an algorithm which takes as input the domains of the variables in $scope(C)$ and re-

[*]Supported by the project ANR-06-BLAN-0383-02.

[†]NICTA is funded by the Australian Government through the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

turns *restrictions* of these domains. Following [Schulte and Stuckey, 2004], we can formally define a propagation algorithm as a function:

Definition 1 (Propagator) *A propagator f for a constraint C is a polynomial time computable function $f : \mathcal{P}(\mathbf{D}) \rightarrow \mathcal{P}(\mathbf{D})$, such that f is* monotone*, i.e., $\mathbf{D}'(\mathbf{X}) \subseteq \mathbf{D}(\mathbf{X}) \implies f(\mathbf{D}'(\mathbf{X})) \subseteq f(\mathbf{D}(\mathbf{X}))$,* contracting*, i.e., $f(\mathbf{D}(\mathbf{X})) \subseteq \mathbf{D}(\mathbf{X})$, and* idempotent*, i.e., $f(f(\mathbf{D}(\mathbf{X}))) = f(\mathbf{D}(\mathbf{X}))$. If a literal $X_i = j$ is in $\mathbf{D}(\mathbf{X}) \setminus f(\mathbf{D}(\mathbf{X}))$ then $X_i = j$ does not belong to any solution of C given $\mathbf{D}(\mathbf{X})$. If f detects that C has no solutions under $\mathbf{D}(\mathbf{X})$ then $f(\mathbf{D}(\mathbf{X})) = \emptyset$.*

A propagator *detects dis-entailment* if when no possible assignment is a solution of C then $f(\mathbf{D}(\mathbf{X})) = \emptyset$. A propagator enforces *domain consistency (DC)* when $X_i = j \in f(\mathbf{D}(\mathbf{X}))$ implies that there exists a solution of C that contains $X_i = j$.

We also define the *consistency checker* for a constraint C as a function that returns 0 when it detects that no possible assignment is a solution of the constraint and 1 otherwise, rather than restricting domains.

Definition 2 (Consistency checker) *A consistency checker f for a constraint C is a polynomial time computable function $f : \mathcal{P}(\mathbf{D}) \rightarrow \{0, 1\}$ such that f is monotone, i.e., $\mathbf{D}'(\mathbf{X}) \subseteq \mathbf{D}(\mathbf{X}) \implies f(\mathbf{D}'(\mathbf{X})) \leq f(\mathbf{D}(\mathbf{X}))$. If $f(\mathbf{D}(\mathbf{X})) = 0$ then no possible assignment under $\mathbf{D}(\mathbf{X})$ is a solution of C.*

We can obtain a polynomial time consistency checker f_C of a constraint C from a polynomial time propagator f_P for C and vice versa [Bessiere *et al.*, 2007]. Given the propagator f_P, the corresponding consistency checker f_C is defined as:

$$f_C(\mathbf{D}(\mathbf{X})) = \begin{cases} 0 & f_P(\mathbf{D}(\mathbf{X})) = \emptyset \\ 1 & otherwise \end{cases} \quad (1)$$

Conversely, given f_C, the propagator f_P is

$$f_P(\mathbf{D}(\mathbf{X})) = \mathbf{D}(\mathbf{X}) \setminus \{X_i = j \mid f_C(\mathbf{D}(\mathbf{X})|_{X_i=j}) = 0\} \quad (2)$$

where $\mathbf{D}(\mathbf{X})|_{X_i=j} = \mathbf{D}(\mathbf{X}) \setminus \{X_i = k | k \neq j\}$.

SAT. The Boolean satisfiability problem (SAT) is a special case of the CSP where variables are Boolean. For each Boolean variable x_i there exist two *literals* x_i and $\overline{x_i}$. Constraints in conjunctive normal form (CNF) are disjunctions of literals, called *clauses* and sometimes written simply as tuples of literals.

Unit propagation *forces* a literal to TRUE if it appears in a clause where all other literals are FALSE and continues until a fix-point is reached. If all literals in a clause are made FALSE, we say that the empty clause is produced. A stronger form of inference is the *failed literal test* [Freeman, 1995]. For each literal l of an unset variable x, the failed literal test sets l to TRUE, performs unit propagation, checks whether the empty clause was produced and retracts l and its consequences. If the empty clause was produced, l is set to FALSE.

A CSP instance can be encoded as a SAT instance. The most widely used mapping of CSP variables to Boolean variables is the *direct encoding*. Each CSP variable X_i with domain $D(X_i)$ is encoded in SAT as a set of propositions $x_{i,j}$, $X_i \in \mathbf{X}, j \in D(X_i)$ such that $X_i \neq j \iff \overline{x}_{i,j}$. The property that each CSP variable has at most one value is enforced by the set of clauses $(\overline{x}_{i,j}, \overline{x}_{i,k})$ for all $k \in D(X_i), k \neq j$ and the property that each CSP variable has at least one value is enforced by the set of clauses $\bigvee_{j \in D(X_i)} x_{i,j}$. We denote this propositional representation of $\mathbf{D}(\mathbf{X})$ as $\mathbf{D}^{sat}(\mathbf{X})$.

Note that the propositional representation $\mathbf{D}^{sat}(X)$ represents the *current state* of the domains $\mathbf{D}(\mathbf{X})$ during search. This means that when the domains change, we need to be able to make the corresponding change in the direct encoding. Consequently, the fact $(X_i = j) \in \mathbf{D}(\mathbf{X})$ is represented by $x_{i,j}$ being *unset*, rather than TRUE. When the value $X_i = j$ is pruned, then $x_{i,j}$ is set to FALSE. Only when $X_i = j$ is the only possible assignment for X_i is $x_{i,j}$ set to TRUE. This means that the same domain can be represented by different partial instantiations of the direct encoding. For example, given the CSP variable X_1 with initial domain $\{1, 2, 3\}$, the instantiation $\mathbf{D}^{sat}(\{X_1\}) = \{\overline{x}_{1,2}, \overline{x}_{1,3}\}$ (with $x_{1,1}$ unset) corresponds to the same domain as $\mathbf{D}^{sat}(\{X_1\}) = \{x_{1,1}, \overline{x}_{1,2}, \overline{x}_{1,3}\}$, which is $\mathbf{D}(\{X_1\}) = \{X_1 = 1\}$.

Boolean Circuits. A Boolean circuit S is a directed acyclic graph (DAG). Each source vertex of the DAG is an *input gate* and the unique sink of the DAG is the output gate. Each non-input vertex is labelled with a logical connective, such as and ($\wedge$), or ($\vee$) and not ($\neg$). An *input* $\mathbf{b}$ to the circuit is an assignment of a *value* 0 or 1 to each input gate.[1] The value of a non-input gate is computed by applying the connective that it is labelled with to the values of its ancestor gates. The value of the circuit $S(\mathbf{b})$ is the value of its output gate.

Any polynomial time decision algorithm can be encoded as a Boolean circuit of polynomial size for a fixed length input [Papadimitriou and Steiglitz, 1982].

In this paper, we will use a restriction of Boolean circuits to $\wedge$-gates and $\vee$-gates, called *monotone circuits*. The family of functions that are computable by monotone circuits is exactly all the monotone Boolean functions. Note that there exist families of polynomial time computable monotone Boolean functions such that the smallest monotone circuit that computes them is super-polynomial in size [Razborov, 1985].

Definition 3 (Monotone Boolean function) *A Boolean function f is monotone iff $f(\mathbf{b}) = 0$ implies $f(\mathbf{b}') = 0$ for all $\mathbf{b}' \leq \mathbf{b}$, where $\leq$ is the pairwise vector comparison, i.e., $b'_i \leq b_i$ for all i.*

A consistency checker f_C, previously defined as a monotone function over sets, can also be formalised as a monotone Boolean function whose input is the characteristic function of the set $\mathbf{D}(\mathbf{X})$. Literals $X_i = j$ are mapped to arguments $b_{i,j}$ of the function, with $b_{i,j} = 1$ iff $X_i = j \in \mathbf{D}(\mathbf{X})$. We use $\mathbf{D}^b(\mathbf{X})$ to denote the setting of the $b_{i,j}$ inputs for a given set of domains $\mathbf{D}(\mathbf{X})$.

3 Properties of CNF decompositions

In this section, we define formally a *CNF decomposition* of a propagator and of a consistency checker. As with propagators and consistency checkers [Bessiere *et al.*, 2007], we show that there exists a polynomial time conversion between

[1] This is in contrast to TRUE and FALSE for SAT variables.

the CNF decompositions of a propagator and of the corresponding consistency checker.

Definition 4 (CNF Decomposition of a propagator) *A CNF decomposition of a propagation algorithm f_P is a formula in CNF C_P over variables $\mathbf{x} \cup \mathbf{y}$ such that*

- *The* input variables $\mathbf{x}$ *are the propositional representation $\mathbf{D}^{sat}(\mathbf{X})$ of $\mathbf{D}(\mathbf{X})$ and $\mathbf{y}$ is a set of auxiliary variables whose size is polynomial in $|\mathbf{x}|$.*
- *$x_{i,j}$ is set to* FALSE *by unit propagation if and only if $X_i = j \notin f_P(\mathbf{D}(\mathbf{X}))$.*
- *Unit propagation on C_P produces the empty clause when $f_P(\mathbf{D}(\mathbf{X})) = \emptyset$.*

Example 1 *To illustrate Definition 4, consider a* TABLE *constraint over the variables X_1, X_2 with $D(X_1) = D(X_2) = \{a, b\}$ and the satisfying assignments: $\{\langle a, a\rangle, \langle b, b\rangle\ \langle a, b\rangle\}$. [Bacchus, 2007] decomposes such a* TABLE *constraint into CNF using the following set of clauses:*

$$\begin{array}{llll} x_{1a} \Rightarrow y_1 \vee y_3 & x_{2a} \Rightarrow y_1 & y_1 \Rightarrow x_{1a} & y_1 \Rightarrow x_{2a} \\ x_{1b} \Rightarrow y_2 & x_{2b} \Rightarrow y_2 \vee y_3 & y_2 \Rightarrow x_{1b} & y_2 \Rightarrow x_{2b} \\ y_3 \Rightarrow x_{1a} & y_3 \Rightarrow x_{2b} & y_1 \vee y_2 \vee y_3 & \end{array}$$

where $\mathbf{x} = \{x_{i,j}\}$, $i \in \{1, 2\}$, $j \in \{a, b\}$ is the propositional representation $\mathbf{D}^{sat}(\mathbf{X})$ of $\mathbf{D}(\mathbf{X})$ and $\mathbf{y} = \{y_i\}$, $i \in \{1, 2, 3\}$ are auxiliary variables that correspond to satisfying tuples. Note that we have extended Bacchus's encoding with the clause $(y_1 \vee y_2 \vee y_3)$ to detect failure. Suppose the value a is removed from the domain of X_1. The assignment $x_{1a} =$ FALSE forces the variable y_1 to FALSE*, which in turn causes the variable x_{2a} to* FALSE*, removing the value a from the domain of X_2 as well.*

In example 1, we have decomposed a constraint into clauses by introducing variables. In general, an encoding might be exponentially bigger if auxiliary variables are not used (e.g., the parity function [Darwiche and Marquis, 2002]).

Definition 5 (CNF Decomposition of a consistency checker) *A CNF decomposition of a consistency checker f_C is a CNF C_C over variables $\mathbf{x} \cup \mathbf{y} \cup \{z\}$ such that*

- *The input variables $\mathbf{x}$ are the propositional representation $\mathbf{D}^{sat}(\mathbf{X})$ of $\mathbf{D}(\mathbf{X})$ and $\mathbf{y}$ is a set of auxiliary variables whose size is polynomial in $|\mathbf{x}|$. The variable z is the* output variable*.*
- *Unit propagation on C_C never forces any variable from $\mathbf{x}$ or generates the empty clause if no variable in $\mathbf{y}$ is set externally to C_C, i.e., every variable $y \in \mathbf{y}$ is either unset or forced by a clause in C_C.*
- *z is set to* FALSE *by unit propagation if and only if $f_C(\mathbf{D}(\mathbf{X})) = 0$.*

Example 2 *Consider the* TABLE *constraint from Example 1. We construct a CNF decomposition of a consistency checker using the CNF decomposition of a propagator. The clauses that cause pruning of input variables domains are removed and the last clause is augmented with the output variable z to avoid generation of the empty clause in the case of failure:*

$$\begin{array}{llll} y_1 \Rightarrow x_{1a} & y_1 \Rightarrow x_{2a} & y_2 \Rightarrow x_{1b} & y_2 \Rightarrow x_{2b} \\ y_3 \Rightarrow x_{1a} & y_3 \Rightarrow x_{2b} & \overline{y}_1 \wedge \overline{y}_2 \wedge \overline{y}_3 \Rightarrow \overline{z} & \end{array}$$

In this case, if the value a is removed from the domain of X_1, unit propagation will not deduce that a has to be removed from the domain of X_2. Consider instead the case when the values a and b are removed from the domains of X_1 and X_2, respectively. The literals $x_{1a} =$ FALSE and $x_{2b} =$ FALSE force the auxiliary variables y_1, y_2 and y_3 to be FALSE*. Therefore, the output variable z is forced to* FALSE*, signalling that the* TABLE *constraint does not have a solution under $\mathbf{D}(\mathbf{X})$.*

In example 2, we transformed the propagator of example 1 into a consistency checker in an ad-hoc manner. The next theorem shows that this can be done in a generic way. We give a polynomial transformation of CNF decompositions of a propagator into consistency checkers This mirrors the results of [Bessiere *et al.*, 2007] for CNF decompositions.

Theorem 1 *There exists a polynomial time and space conversion between the CNF decomposition of a propagator f_P and that of the corresponding consistency checker f_C.*

Proof: $(\rightarrow)$ We construct C_C as a transformation of C_P such that the output variable z of C_C is FALSE iff unit propagation on C_P produces the empty clause.

Let the set of clauses of C_P be $c_1 \ldots c_m$. For each variable $p \in \mathbf{x} \cup \mathbf{y}$, we introduce 2 variables p_t and p_f in C_C so that p_t and p_f are true if p is forced to TRUE or FALSE, respectively:

$$p \implies p_t \qquad \overline{p} \implies p_f \tag{3}$$

Then, we simulate unit propagation for each clause c_k by replacing it with 3 implications[2] that contain the variables p_t and p_f rather than p. For example, to simulate unit propagation for the clause $c_1 = (p, q, \overline{r})$, we replace it with

$$p_f \wedge q_f \implies r_f \quad p_f \wedge r_t \implies q_t \quad q_f \wedge r_t \implies p_t \tag{4}$$

Unit propagation on (4) can never derive the empty clause, because the true and false values of p are encoded in different variables p_t and p_f, which may be true simultaneously. When this happens, unit propagation on C_P would generate the empty clause, therefore we must set the output variable z to FALSE, using the following clauses:

$$p_t \wedge p_f \implies \overline{z} \tag{5}$$

The union of the clauses (3), (4) and (5) is a CNF decomposition of f_C with size $O(|\mathbf{x} \cup \mathbf{y}| + |C_P|) = O(|C_P|)$, therefore the transformation is polynomial.

$(\leftarrow)$ We outline the proof here. We replicate the equation (2) by simulating the failed literal test on $C_C \cup \{(z)\}$. For each literal $x_{i,j}$ we create a copy of C_C, denoted by $C_C|_{x_{i,j}}$, in which all literals $x_{i,k}, k \neq j$ are FALSE. We use $C_C|_{x_{i,j}}$ to record the results of unit propagation when $X_i = j$. When unit propagation sets the output variable $z_{x_{i,j}}$ of the copy $C_C|_{x_{i,j}}$ to FALSE then the propositional literal $x_{i,j}$ is made FALSE by the additional clause $(\overline{z}_{x_{i,j}} \implies \overline{x}_{i,j})$.

[2] We assume that formulas are given in 3-CNF form. We can convert any CNF formula to 3-CNF, increasing its size by at most a constant factor and without hindering unit propagation [Garey and Johnson, 1979, section 3.1.1].

The decomposition C_P is then the union of the copies of C_C and the clauses $(\overline{z}_{x_{i,j}} \implies \overline{x}_{i,j})$:

$$C_P = \bigcup_{x_{i,j} \in \mathbf{x}} (C_C|_{x_{i,j}} \cup (z_{x_{i,j}}, \overline{x}_{i,j})) \quad (6)$$

The size of C_P is $O(|\mathbf{x}| \cdot |C_C|)$, therefore the transformation is polynomial. □

Using the encoding of theorem 1, a CNF decomposition of a consistency checker that detects dis-entailment can be made into a propagator that enforces domain consistency. As an example, consider the CNF decomposition of a propagator that detects dis-entailment for the SEQUENCE constraint, proposed in [Bacchus, 2007]. The size of this decomposition is $O(n^2)$, where n is the number of variables in the SEQUENCE constraint. These variables are binary, hence the transformation of theorem 1 yields a decomposition of a DC propagator with size $O(n^3)$. This is also the complexity of the DC propagator proposed in [van Hoeve *et al.*, 2006].

Since all definitions of CNF decompositions that we introduced in this section are polynomially equivalent, in the remainder of this paper we only prove results for CNF decompositions of consistency checkers.

4 Equivalence to monotone circuits

In this section, we show our main result, which establishes a connection between CNF decompositions of constraints and circuit complexity.

Theorem 2 *A consistency checker f_C can be decomposed to a CNF of polynomial size if and only if it can be computed by a monotone circuit of polynomial size.*

The proof of theorem 2 is constructive. We will first show the reverse direction, using the Tseitin encoding [Tseitin, 1983] of a monotone circuit.

Definition 6 (Tseitin encoding of a Boolean circuit) *The Tseitin encoding of a circuit S into clausal form has one propositional variable for each input of S and for each gate of S. W.l.o.g, we assume all gates have fan-in 2. For each $\wedge$-gate g with inputs x_1, x_2, the Tseitin encoding contains the clauses $(x_1, \overline{g})$, $(x_2, \overline{g})$, $(\overline{x}_1, \overline{x}_2, g)$ and for each $\vee$-gate it contains the clauses $(\overline{x}_1, g)$, $(\overline{x}_2, g)$, $(x_1, x_2, \overline{g})$. Given any complete instantiation of the input variables, unit propagation on the Tseitin encoding sets the variable corresponding to the output gate of S to* TRUE *if the circuit computes 1 and to* FALSE *otherwise.*

Suppose that a consistency checker f_C can be encoded into a monotone circuit S_C of polynomial size. The Tseitin encoding of S_C turns out to be a CNF decomposition of f_C. This is a direct consequence of the following lemma.

Lemma 1 *Let S_C be a monotone circuit and C_C be its Tseitin encoding. Let I be a partial instantiation of the input variables $\mathbf{x}$ of C_C and $\mathbf{b}$ be the corresponding input to S_C, where $b_i = 0$ iff $\overline{x}_i \in I$. Then, unit propagation on C_C with I forces the output variable z to* FALSE *if and only if $S_C(\mathbf{b}) = 0$.*

Proof: ($\rightarrow$) This follows from the correctness of the Tseitin encoding.

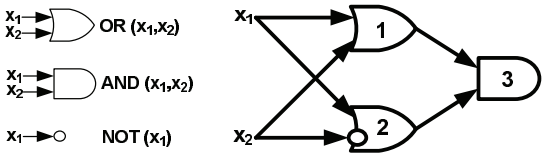
($\leftarrow$). Suppose that $S_C(\mathbf{b}) = 0$, but the output variable z is not forced to FALSE by unit propagation under I. Consider an instantiation I' of the input variables of C_C, which is the same as I with unset variables fixed to TRUE. Let $y \in \mathbf{y} \cup \{z\}$ be an auxiliary variable that is unset under I. All such variables correspond to a gate in S_C. Since C_C is an encoding of the monotone circuit S_C, y will be set to TRUE under I'. This means that the output variable z is also set to TRUE. By the correctness of the Tseitin encoding, $S_C(\mathbf{b}) = 1$, a contradiction. □

Corollary 1 *Let S_C be a monotone circuit and C_C be its Tseitin encoding. Let I be a partial instantiation of the input variables $\mathbf{x}$ of C_C. Then, unit propagation on C_C with I forces the output variable z to* FALSE *if and only if $S_C(\mathbf{b}) = 0$, for all $\mathbf{b}$ where $\mathbf{b}$ is the input to S_C that corresponds to any extension of I to a complete instantiation.*

Proof: This follows from lemma 1 and the fact that S_C is a monotone circuit. □

Interestingly, lemma 1 cannot be generalised to non-monotone Boolean circuits. The next example shows that there exists a non-monotone Boolean circuit S that computes a monotone function, and a partial instantiation I with $\mathbf{b}$ the corresponding input to S, such that $S(\mathbf{b}) = 0$ but unit propagation on the Tseitin encoding of S under the instantiation I does not set the output variable to FALSE.

Figure 1 A circuit whose Tseitin encoding is incomplete.



Example 3 *Consider the non-monotone circuit S shown in figure 1. Note that S computes a monotone function.*

The Tseitin encoding of S introduces three Boolean variables g_1, g_2 and g_3 for the gates OR_1, OR_2 and AND_3, respectively, and the clauses $(\overline{x}_1, g_1)$, $(\overline{x}_2, g_1)$, $(\overline{g}_1, x_1, x_2)$, $(\overline{x}_1, g_2)$, (x_2, g_2), $(\overline{g}_2, x_1, \overline{x}_2)$, $(\overline{g}_3, g_1)$, $(\overline{g}_3, g_2)$, $(\overline{g}_1, \overline{g}_2, g_3)$.

Now suppose that $I = \{\overline{x}_1\}$. Then, $\mathbf{b} = \{x_1 = 0, x_2 = 1\}$ and $S(\mathbf{b}) = 0$. Since S computes a monotone function, all possible extensions of $\mathbf{x}$ evaluate to 0. But in the Tseitin encoding, setting x_1 to FALSE *does not make any clauses unit, therefore unit propagation does not set g_3 to* FALSE. □

We now show the forward direction of theorem 2: every CNF decomposition C_C of a consistency checker f_C can be converted to a monotone circuit that computes f_C with at most a polynomial increase in size.

This transformation exploits two properties of CNF decompositions, namely, that only positive literals of input variables appear in C_C, and that unit propagation only makes auxiliary variables FALSE. We show the former property in lemma 2 and the latter in lemma 3.

Lemma 2 *Let C_C be the CNF decomposition of a consistency checker f_C. There exists a polynomial size CNF decomposition C'_C of f_C such that negative literals of the input variables do not appear in any clause in C'_C.*

Proof: We construct C'_C by removing from C_C all clauses that contain a negative literal of an input variable. We show by contradiction that unit propagation on C'_C and C_C produces identical results for the output variable z.

Let I be a partial instantiation of the input variables such that unit propagation on C_C under I sets z to FALSE but leaves z unset on C'_C. Since unit propagation on C_C and C'_C produces different results, at least one of the removed clauses becomes unit under I in C_C. By definition, C_C never forces any literal of an input variable, so for any removed clause to become unit, all the literals of input variables in it have to be FALSE. Since at least one of these literals is negative, at least one input variable has to be set to TRUE in I.

We construct another partial instantiation I' from I by setting the same literals to FALSE as I and leaving the rest unset, i.e., $I' = \{\overline{x}_{i,j} | \overline{x}_{i,j} \in I\}$. The partial instantiations I and I' represent the same domains $\mathbf{D}(\mathbf{X})$, because the mapping from partial instantiation to domain depends only on the literals that are FALSE. By this and the fact that C_C is a decomposition of f_C, unit propagation on C_C under I' forces the output variable z to the same value as under I, FALSE.

Consider the result of unit propagation on C'_C under I'. Recall that by definition C_C does not modify input variables and I' does not have literal set to TRUE by construction. Hence, none of the clauses that we remove from C_C to get C'_C can become unit after performing UP on C_C under I'. Hence, unit propagation in C'_C under I' sets z to FALSE as in C_C. On the other hand, I sets a superset of the literals that I' sets, so unit propagation on C'_C under I also sets z to FALSE, a contradiction, since we assumed that C'_C leaves z unset under I. □

In practice, a CNF decomposition of a consistency checker may not be self contained and may depend on the existence of clauses in the direct encoding of variable domains. In this case, we cannot just remove clauses that contain negative literals of input variables, as lemma 2 suggests. However, using the clauses of the direct encoding, we can substitute negative literals with the disjunction of positive literals. For instance, consider a variable X_2 with the domain $\{1, 2, 3\}$ and a clause $(x_{1,1}, \overline{x}_{2,2}, \overline{y})$ in C_C. The literal $x_{2,2}$ can make this clause unit. The direct encoding of $D(X_2)$ includes a clause $(x_{2,1}, x_{2,2}, x_{2,3})$. Note that the literal $x_{2,2}$ is TRUE if and only if literals $x_{2,1}$ and $x_{2,3}$ are FALSE. Therefore, the literal $\overline{x}_{2,2}$ can be replaced with the disjunction $(x_{2,1}, x_{2,3})$ and the clause $(x_{1,1}, \overline{x}_{2,2}, \overline{y})$ is transformed to the clause $(x_{1,1}, x_{2,1}, x_{2,3}, \overline{y})$.

The next step is to show that we can transform a CNF decomposition so that each auxiliary variable is unset or FALSE for *all* inputs that make the output variable FALSE. The transformation is a renaming of the auxiliary variables. Lemma 3 describes the property that allows this transformation.

Lemma 3 *Let C_C be a CNF decomposition of a consistency checker f_C over the variables $\mathbf{x} \cup \mathbf{y} \cup \{z\}$, $I_1 = \mathbf{D}_1^{sat}(\mathbf{X}), I_2 = \mathbf{D}_2^{sat}(\mathbf{X})$ be the propositional representations of any two domain settings such that unit propagation on C_C forces z to* FALSE *under both I_1 and I_2. For any variable $y \in \mathbf{y}$, if y is forced to* FALSE *(*TRUE*) by unit propagation under I_1 then it is not forced to* TRUE *(*FALSE*) by unit propagation under I_2.*

Proof: Let a variable y be forced to TRUE by unit propagation under I_1 and to FALSE under I_2, but z is FALSE under both I_1 and I_2. Consider the partial instantiation I such that if a variable $x \in \mathbf{x}$ is FALSE in either I_1 or I_2, it is also FALSE in I, otherwise it is unset. Since I fixes a superset of the literals that are fixed in either I_1 or I_2, all clauses that became unit by either I_1 or I_2 will also be unit in I. Therefore, unit propagation under I will force at least the union of the sets of literals forced by I_1 and I_2. This means that unit propagation under I will make both y and $\overline{y}$ TRUE, which generates the empty clause. This is a contradiction, as C_C can never produce the empty clause. □

Corollary 2 *A CNF decomposition C_C of a consistency checker f_C over variables $\mathbf{x} \cup \mathbf{y} \cup \{z\}$, can be polynomially converted into a decomposition C'_C of f_C such that every variable in $\mathbf{y}$ is either unset or* FALSE *when z is* FALSE*.*

Proof: We construct C'_C from C_C by flipping the polarity of those variables that are set to TRUE when z is FALSE. □

Lemma 2 and corollary 2 allow us to precisely characterize the form of the clauses in a CNF decomposition.

Corollary 3 *Let C_C be a CNF decomposition of a consistency checker f_C. The variables of C_C can be renamed so that each clause has exactly one negative literal.*

Proof: By lemma 2, all input variables are positive literals in the decomposition and by definition 5 they are never forced by unit propagation on C_C. In addition, by corollary 2, we can rename the auxiliary variables so that unit propagation on C_C may only ever set them to FALSE. Then, in any clause that consists of input variables and one auxiliary variable y, y must be negative, otherwise it may be set to TRUE, a contradiction.

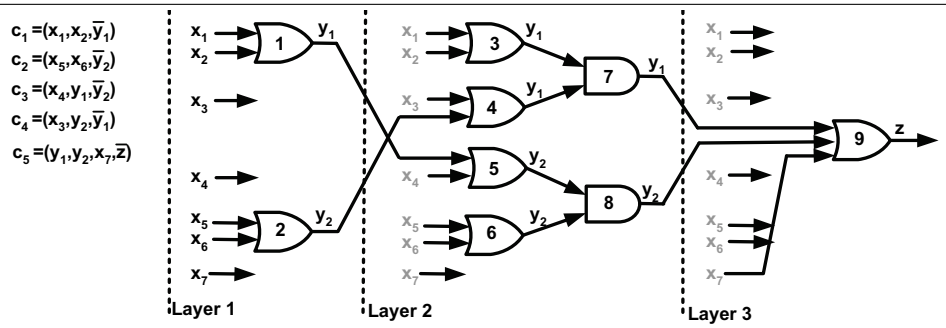
Suppose there exists a clause c with two auxiliary variables y_1 and y_2 and both are negative in c. Since neither y_1 nor y_2 can ever be made TRUE, this clause can never become unit and can be ignored. Suppose the literals of both y_1 and y_2 are positive in c. Then, if c becomes unit, it makes one of the auxiliary variables TRUE, a contradiction. Thus, exactly one of the literals of y_1 and y_2 is negative in c. The same reasoning can be extended to clauses with more than two auxiliary variables. □

The condition described by corollary 3 is similar to C_C being re-nameable anti-Horn, but is stronger as it requires *exactly* one negative literal in each clause, rather than at most one. This condition allows us to build a monotone circuit from a decomposition, using the construction of the next lemma.

Lemma 4 *Let C_C be a CNF decomposition of a consistency checker f_C. Then, there exists a monotone circuit S_C of size $O(n|C_C|)$ that computes f_C.*

Proof: We assume that C_C is in the form described in corollary 3.

Figure 2 Conversion of a CNF decomposition of a consistency checker into a monotone Boolean circuit.



The inputs of the circuit correspond to the input variables of C_C. For each input variable $x_{i,j}$ of C_C, there exists an input $b_{i,j}$ of S_C which is 0 if $x_{i,j}$ is FALSE and 1 otherwise. Internal gates of the circuit correspond to auxiliary variables after a certain number of unit propagation steps, using the same mapping.

We create a circuit with $|\mathbf{y}|$ *layers* $1 \ldots |\mathbf{y}|$. Let $c_1, \ldots, c_m$ be the clauses of C_C. The i^{th} layer of the circuit contains an $\vee$-gate c_j^i for each clause c_j, called *clause gates* and an $\wedge$-gate y_k^i for each auxiliary variable y_k, called *variable gates*. Consider a clause c_j which contains $\overline{y}$ as the sole negative literal (recall that corollary 3 ensures that this is the case), the positive literals of input variables $x_{j_1}, \ldots, x_{j_q}$ and the positive literals of auxiliary variables $y_{j_{q+1}}, \ldots, y_{j_{q+r}}$. The inputs of each gate c_j^i are $b_{j_1}, \ldots, b_{j_q}$ and $y_{j_{q+1}}^{i-1}, \ldots, y_{j_{q+r}}^{i-1}$. Let the clauses with $\overline{y}_k$ as the sole negative literal be $c_{k_1}, \ldots, c_{k_s}$. Then, the inputs of each gate y_k^i are $c_{k_1}^i, \ldots, c_{k_s}^i$. The output of the circuit is $z^{|\mathbf{y}|}$. Note that in this construction the inputs of some the gates may not be defined. This is the case, for example, for the gate c_i^1, where the clause c_i contains the positive literals of some auxiliary variables. If this happens for a clause gate, we omit it, while if it happens for a variable gate, we omit the undefined input. If all the inputs of a variable gate are undefined, we omit the gate.

This construction computes one breadth first application of unit propagation at each layer. Specifically, the gate y_k^i is 0 iff y_k is forced to FALSE after i or fewer breadth first steps of unit propagation, while the gate c_j^i is 0 iff the negated variable in c_j is forced to FALSE after i or fewer breadth first steps of unit propagation. We show this by induction. For the first layer, there exist gates only for clauses with no positive literals of auxiliary variables. Consider any such gate c_j which contains the negative literal $\overline{y}_k$. All the propositional variables in c_j except y_k are FALSE iff the corresponding inputs are 0. Thus c_j^1 is 0 iff y_k is FALSE after unit propagation of c_j. If many clauses contain the negative literal $\overline{y}_k$, then at least one of them sets y_k to FALSE in one breadth first step iff there exists a clause gate that is 0 and is an input to the variable gate y_k^1, which is an $\wedge$-gate and is thus 0. For the inductive step, assume that the layers $1 \ldots k-1$ compute $k-1$ breadth first steps of unit propagation. The same reasoning as for the base case shows that the results of unit propagation are correctly computed for the k^{th} layer. Note that the k^{th} layer may also contain gates that were omitted at previous levels. Since the inputs of these gates are correctly computed by the inductive hypothesis, the gates that are new to the k^{th} layer are also correctly computed.

To conclude the proof, observe that in the extreme case, unit propagation will set one more literal at every breadth first step, thus after $|\mathbf{y}|$ steps it must either arrive at a fixpoint or set all literals. Since the circuit has $|\mathbf{y}|$ layers, it will correctly compute the result of unit propagation on C_C. □

We illustrate the construction of lemma 4 with an example.

Example 4 *Consider the CNF decomposition* $C_C = \{c_1, c_2, c_3, c_4, c_5\}$, *where* $c_1 = (x_1, x_2, \overline{y}_1)$, $c_2 = (x_5, x_6, \overline{y}_2)$, $c_3 = (x_4, y_1, \overline{y}_2)$, $c_4 = (x_3, y_2, \overline{y}_1)$, $c_5 = (y_1, y_2, x_7, \overline{z})$.

We construct a monotone circuit S_C from C_C, (figure 2). For a given instantiation of the input variables, this circuit computes 0 for the corresponding Boolean inputs if and only if unit propagation on C_C forces the output variable to FALSE.

The circuit consists of 3 layers, with gates 1 and 2 in the first layer, 3–8 in the second and gate 9 in the third. The gates 1–6 and 9 are clause gates, while gates 7 and 8 are variable gates. A strict application of the construction of lemma 4 would also have variable gates in layers 1 and 3, but we omit them here as they would be single-input gates. Note that in figure 2, inputs are replicated at each layer to reduce clutter.

We note also that the layered construction of lemma 4 is necessary. A circuit that attempts to capture unit propagation on all clauses without using layers would have to contain a cycle between the gates that compute y_1 and y_2, because y_1 would need to be an input of the clause gate c_3 that computes y_2 and y_2 would need to an input of the clause gate c_4 that computes y_1. Constructing a layered circuit allows us to remove such cycles. □

The proof of theorem 2 is now immediate from lemmas 1 and 4. Since CNF decompositions of consistency checkers can be converted in polynomial time to and from CNF decompositions of propagators, theorem 2 also holds for propagators.

5 Non decomposable global constraints

Corollary 4 now uses an existing circuit complexity result to show that, unsurprisingly, there is no polynomial size CNF decomposition of the domain consistency propagator for the

ALLDIFFERENT constraint. This also applies to generalizations of ALLDIFFERENT, such as GCC.

Corollary 4 *There is no polynomial sized CNF decomposition of the* ALLDIFFERENT *domain consistency propagator.*

Proof: Régin [Régin, 1994] showed that an ALLDIFFERENT constraint has a solution iff the corresponding bipartite value graph (i.e., the graph where the node representing a variable has an edge to every node that represents a value in its domain) has a perfect matching. In addition, every bipartite graph corresponds to the value graph of an ALLDIFFERENT constraint and DC propagators detect dis-entailment. Thus, if there exists a polynomial size CNF decomposition of the ALLDIFFERENT DC propagator, we can construct a monotone circuit that computes whether a bipartite graph has a perfect matching. But Razborov [Razborov, 1985] showed that the smallest monotone circuit that computes whether there exists a perfect matching for a bipartite graph is super-polynomial in the number of vertices in the graph. Therefore, the smallest CNF decomposition of the ALLDIFFERENT DC propagator is super-polynomial in size. □

On the other hand, bound and range consistency propagators of ALLDIFFERENT can be decomposed, as we argue in [Bessiere *et al.*, 2009].

6 Conclusions and Future Work

In this paper we have shown how the tools of circuit complexity can be used to study decompositions of global propagators into CNF. Our results directly extend to decompositions into CSP constraints of bounded arity with domains given in extension since such decompositions can be translated into clauses of polynomial size. An interesting next step is to consider the decomposability of constraint propagators into more expressive primitive constraints where domains are represented in logarithmic space via their bounds. CSP solvers provide this feature which is missing in CNF. We conjecture that there exists an equivalence between such CSP decompositions of constraint propagators and monotone arithmetic circuits that are generalizations of Boolean monotone circuits to real numbers and gates for addition and multiplication. Since lower bound results on monotone circuits usually transfer to monotone arithmetic circuits, this would imply that the domain consistency propagator for ALLDIFFERENT cannot be decomposed to constraints that exploit (exponentially) large domains.

References

[Bacchus, 2007] F. Bacchus. GAC via unit propagation. In *13th Int. Conf. on Principles and Practices of CP (CP2007)*, 133–147. 2007.

[Beldiceanu *et al.*, 2005] N. Beldiceanu, I. Katriel, and S. Thiel. Reformulation of Global Constraints Based on Constraints Checkers Filtering algorithms for the same constraint. *Constraints*, 10(4): 339–362, 2005.

[Bessiere and Van Hentenryck, 2003] C. Bessiere and P. Van Hentenryck. To be or not to be ... a global constraint. In *9th Int. Conf. on Principles and Practices of CP (CP2007)*, 789–794. 2003

[Bessiere *et al.*, 2003] C. Bessiere, E. Hebrard, and T. Walsh. Local consistencies in SAT. In *6th Int. Conf. on Theory and Applications of Satisfiability Testing*, 299–314. 2003

[Bessiere *et al.*, 2005] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan and T. Walsh. The Range and Roots Constraints: Specifying Counting and Occurrence Problems. In *19th Int. Joint Conf. on AI*, 60–65. 2005.

[Bessiere *et al.*, 2006a] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan and T. Walsh. The RANGE Constraint: Algorithms and Implementation. In *3rd Int. Conf. on Integration of AI and OR Techniques in CP (CP-AI-OR)*, 59–73, 2006.

[Bessiere *et al.*, 2006b] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan and T. Walsh. The ROOTS Constraint. In *12th Int. Conf. on Principles and Practices of CP (CP2006)*, 75–90. 2006.

[Bessiere *et al.*, 2007] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of global constraints. *Constraints*, 12(2):239–259, 2007.

[Bessiere *et al.*, 2008] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan and T. Walsh. SLIDE: A Useful Special Case of the CARD-PATH Constraint. In *18th European Conf. on AI*, 475–479. 2008.

[Bessiere *et al.*, 2009] C. Bessiere, G. Katsirelos, N. Narodytska, C.-G. Quimper and T. Walsh. Decompositions of All Different, Global Cardinality and Related Constraints. In *21st Int. Joint Conf. on AI*, 2009.

[Brand *et al.*, 2007] S. Brand, N. Narodytska, C.-G. Quimper, P. Stuckey and T. Walsh. Encodings of the SEQUENCE Constraint. In *13th Int. Conf. on Principles and Practice of CP (CP2007)*, 210–224. 2007.

[Darwiche and Marquis, 2002] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[Freeman, 1995] J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, 1995.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman & Co Ltd, 1979.

[Katsirelos *et al.*, 2008] G. Katsirelos, N. Narodytska and T. Walsh. The Weighted CFG Constraint. In *5th Int. Conf. on Integration of AI and OR Techniques in CP (CP-AI-OR)*, 323–327, 2008.

[Papadimitriou and Steiglitz, 1982] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[Quimper and Walsh, 2006] C.-G. Quimper and T. Walsh. Global Grammar Constraints. In *12th Int. Conf. on Principles and Practices of CP (CP2006)*, 751–755. 2006.

[Quimper and Walsh, 2007] C.-G. Quimper and T. Walsh. Decomposing Global Grammar Constraints. In *13th Int. Conf. on Principles and Practices of CP (CP2007)*, 590–604. 2007.

[Quimper and Walsh, 2008] C.-G. Quimper and T. Walsh. Decompositions of Grammar Constraints. In *23rd National Conf. on AI*, 1567–1570. AAAI, 2008.

[Razborov, 1985] A. A. Razborov. Lower bounds on the monotone complexity of some Boolean functions. *Doklady Akademii Nauk SSSR*, 285:798–801, 1985.

[Régin, 1994] J-C. Régin. A filtering algorithm for constraints of difference in CSP. In *AAAI*, p. 362–367, 1994.

[Schulte and Stuckey, 2004] C. Schulte and P. J. Stuckey. Speeding up constraint propagation. In *CP-2004*, pages 619–633, 2004.

[Tseitin, 1983] G. Tseitin. On the complexity of proofs in propositional logics. In *Automation of Reasoning: Classical Papers in Comp. Logic 1967–1970*, vol.2. 1983.

[van Hoeve *et al.*, 2006] W. J. van Hoeve, G. Pesant, L. M. Rousseau, and A. Sabharwal. Revisiting the sequence constraint. In *CP-2006*, pages 620–634, 2006.

[Walsh, 2006] T. Walsh. Symmetry Breaking using Value Precedence. In *17th European Conf. on AI*, 168–172. 2006.