Coupling Different Integer Encodings for SAT

Hendrik Bierlee[1,2 [0000−0001−6766−5435]], Graeme Gange[1 [0000−0002−1354−431X]], Guido Tack[1,2 [0000−0003−3357−6498]], Jip J. Dekker[1,2 [0000−0002−0053−6724]], and Peter J. Stuckey[1,2 [0000−0003−2186−0459]]

[1] Monash University, Department of Data Science and AI
{hendrik.bierlee,graeme.gange,guido.tack,jip.dekker,peter.stuckey}@monash.edu
[2] ARC Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA)

Abstract. Boolean satisfiability (SAT) solvers have dramatically improved their performance in the last twenty years, enabling them to solve large and complex problems. More recently MaxSAT solvers have appeared that efficiently solve optimisation problems based on SAT. This means that SAT solvers have become a competitive technology for tackling discrete optimisation problems.
A challenge in using SAT solvers for discrete optimisation is the many choices of encoding a problem into SAT. When encoding integer variables appearing in discrete optimisation problems, SAT must choose an encoding for each variable. Typical approaches fix a common encoding for all variables. However, different constraints are much more effective when encoded with a particular encoding choice. This inevitably leads to models where variables have different variable encodings. These models must then be able to *couple* encodings, either by using multiple encoding of single variables and *channelling* between the representations, or by encoding constraints using a mix of representations for the variables involved. In this paper we show how using mixed encodings of integers and coupled encodings of constraints can lead to better (Max)SAT models of discrete optimisation problems.

1 Introduction

Within the last twenty years, Boolean satisfiability (SAT) solving has increased in terms of scalability and performance. More recently optimisation approaches based on SAT, so called MaxSAT technology, have been rapidly developing. SAT and MaxSAT solvers now provide a viable alternative solving technology for many discrete optimisation problems.

Translating discrete optimisation problems to (Max)SAT is a difficult task. Currently, expert SAT modellers build models directly from clauses, or use libraries to encode common constraints such as at-most-one, cardinality or pseudo-Boolean constraints [**?**,**?**]. Alternatively, SAT compilers such as FznTini [**?**] and Picat-SAT [**?**], or some modelling languages such as Essence [**?**], determine a translation of a high-level model.

The first fundamental choice a SAT modeller or compiler faces is how an integer decision variable x should be translated to Boolean decisions. At least three possibilities arise: the *direct encoding*, introducing a Boolean that represents $x = d$ for each value d in the domain of x; the *order encoding*, introducing a Boolean that represents $x \geq d$ for each value d in the domain of x; or the *binary encoding*, introducing a Boolean that represents a bit in the (two's complement) binary encoding of x.

Which encoding should be used for each variable in a discrete optimisation problem is a non-trivial question. The answer may depend on the constraints in which the variable appears, and how important they are to solving the overall problem. In the existing approaches, the variable encoding choices made by the SAT compiler or modeller are hard or impossible to change, since the encoding of every constraint depends on it. Some constraints will be more effective if encoded using a particular variable choice, and if these choices are not all the same for the problem of interest we inevitably have to *couple* different integer encodings. This can be managed by encoding an integer variable in two ways and *channelling* the two encodings, or directly encoding constraints that couple different encodings of the variables involved.

In this paper we show that choosing mixed encodings of integer variables in the model can lead to better solving performance. We investigate new possibilities for channelling and coupling encodings of constraints, and also show how *(partial) views* can be used to extend coupling encodings. The experimental results show how the novel channel, coupling, and view constraints enable unique encodings that outperform existing SAT encoding approaches for discrete optimisation problems of interest.

2 Preliminaries

2.1 Constraint Programming

A *constraint satisfaction problem (CSP)* $P = (\mathcal{X}, D_i, \mathcal{C})$ consists of a set of variables $\mathcal{X}$, with each $x \in \mathcal{X}$ restricted to take values from some initial domain $D_i(x)$. For this paper we assume domains are ordered sets of integers, and denote by $lb(x)$ and $ub(x)$ the least and greatest values in $D_i(x)$. We will use interval notation $l..u$ to represent the set of integers $\{l, l+1, .., u\}$. A set of constraints $\mathcal{C}$ expresses relationships between the variables. A *constraint optimisation problem (COP)* is a CSP P together with objective function o, w.l.o.g. to be maximised.

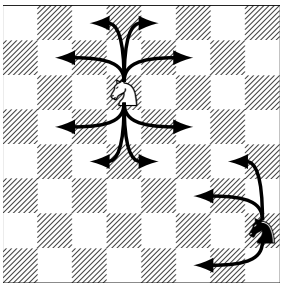
Constraint programming solvers work by propagation over atomic constraints. An *atomic constraint* is (for our purposes) a unary constraint from a given language (which includes the always false constraint *false*). The usual atomic constraints for CP are $x = d$, $x \neq d$, $x \geq d$ and $x \leq d$. A propagator f for constraint C takes a current domain $\mathcal{D}$ given as a set of atomic constraints, and determines a set of new atomic constraints. These are a consequence of the current domain and C, i.e., $a \in f(\mathcal{D})$ implies that $\mathcal{D} \wedge C \rightarrow a$.

A propagator f is *propagation complete* for constraint C and a language of atomic constraints $\mathcal{L}$ iff for any $\mathcal{D} \subseteq \mathcal{L}$ and new atom $a \in \mathcal{L}$, $a \notin \mathcal{D}$: if $\mathcal{D} \wedge C \rightarrow a$

then $a \in f(\mathcal{D})$. That is, the propagator finds all atomic constraints in $\mathcal{L}$ that are consequences of the constraint and the current domain.

Let $DIRECT(x)$ be the language of atomic constraints $\{false\} \cup \{x = d, x \neq d \mid x \in \mathcal{X}, d \in D_i(x)\}$. A propagator is *domain consistent* if it is propagation complete for $\mathcal{L} = DIRECT(x)$. Let $ORDER(x)$ be the language of atomic constraints $\{false\} \cup \{x \leq d, x \geq d \mid x \in \mathcal{X}, d \in D_i(x)\}$. A propagator is *bounds(Z) consistent* [**?**] if it is propagation complete for $\mathcal{L} = ORDER(x)$.

CSPs and COPs are typically expressed using a modelling language such as MiniZinc [**?**]. To illustrate, we show a model for the popular Knight's tour problem in which we are looking for a trajectory of n^2 legal knight moves around an $n \times n$ board. The knight starts and ends at the top-left square (numbered 1), visiting each square exactly once.



The variable `x[p]` gives the next position (in $1..n^2$) to move to from position `p`. MiniZinc allows us to specify the integer variable domains exactly according to legal knight moves (shown in the figure above). The symmetry breaking constraints reduce the domains of two variables to single, fixed values. The global constraint constrains the `x` variables to describe a single Hamiltonian circuit traversal of the graph. The constraint may be implemented directly by the solver or defined in terms of basic constraints using the standard library, or a library specialised for a particular solver. For a given target solver, MiniZinc flattens the high-level model into low-level FlatZinc consisting of built-in constraints.

2.2 Boolean Satisfiability

A Boolean Satisfiability (SAT) problem can be seen as a special case of a CSP, where the domain for all variables x is $D_i(x) \in \{0, 1\}$, representing the values *false* (0) and *true* (1). In SAT problems, we usually talk about *literals*, which are either a Boolean variable x or its negation $\neg x$. We extend the negation operation to operate on literals, i.e., $\neg l = \neg x$ if $l = x$ and $\neg l = x$ if $l = \neg x$. We use the notation $l = v$ where l is a literal and $v \in \{0, 1\}$ to encode the appropriate form of the literal, i.e., if $v = 1$ it is equivalent to l and if $v = 0$ it is equivalent to $\neg l$. The notation $l \neq v$ is defined similarly to encode $\neg(l = v)$. A *clause* is a disjunction of literals. In a SAT problem P, the constraints $\mathcal{C}$ are clauses.

MaxSAT problems are a subclass of COP. A MaxSAT problem[3] (P, o) is a SAT problem P and a (usually non-negative) weight o_b associated to each $b \in \mathcal{X}$. The aim is to minimise $\sum_{b \in \mathcal{X}} o_b b$.

The core operation of SAT solvers is *unit propagation*. Given a set of currently true literals ϕ, if there is a clause $C = l_1 \vee \cdots \vee l_n$ where $\neg l_i \in \phi$ for all $1 \leq i \leq n$ then unit propagation detects failure. Similarly, if $\neg l_i \in \phi$ for all $1 \leq i \neq j \leq n$ then unit propagation adds literal l_j to the current partial assignment ϕ.

3 Encoding integer variables

In order to use MaxSAT solvers for an arbitrary COP (P, o), we need to encode the variables, constraints, and objective of the COP as a MaxSAT problem. We first examine encoding the variables.

Given an integer x with (possibly non-contiguous) initial domain $D_i(x) = \{d_1, d_2, \ldots, d_m\}$, we will now define three encoding methods: the direct encoding, the order encoding, and the binary encoding. Each encoding method maps x to a set of Boolean *encoding variables*. Additionally, *consistency constraints* on the encoding variables ensure that they correctly represent an integer. We use consistent semantic brackets to name the encoding Booleans, where Boolean $[\![f]\!]$ is true iff the formula f holds.

The *direct* encoding of x introduces m encoding variables $[\![x = v]\!], v \in D_i(x)$. $[\![x = v]\!]$ is true iff x is assigned value v. A propagation complete [**?**] encoding of the exactly-one *consistency constraint* is posted on the encoding variables,

$$\sum_{d \in D_i(x)} [\![x = d]\!] = 1 \tag{1}$$

to ensure that the separate Booleans faithfully encode an integer (which must take a single value). Unit propagation of **??** is propagation complete for the constraint $x \in D(x)$ and the language of atomic constraints $DIRECT(x)$.

The *order* encoding of x introduces $m - 1$ encoding variables $[\![x \geq v]\!], v \in \{d_2, \ldots, d_m\}$. $[\![x \geq v]\!]$ is true iff x is assigned a value greater or equal to v. The *consistency constraint*

$$\forall\ 3 \leq i \leq m.\ ([\![x \geq d_i]\!] \rightarrow [\![x \geq d_{i-1}]\!]) \tag{2}$$

enforces that the encoding variables give a consistent view on the integer. Unit propagation on **??** is propagation complete for constraint $x \in D(x)$ and the language of atomic constraints $ORDER(x)$. The order encoding is also known as the ladder, regular, or thermometer encoding.

We extend our semantic brackets notation to make it easier to describe a broader range of bounds constraints as follows: $[\![x \geq d]\!] \equiv 1, d \leq d_1$; $[\![x \geq d]\!] \equiv$

[3] The usual definition of MaxSAT makes use of soft clauses with weights but is functionally equivalent to the definition here, and indeed internally many MaxSAT solvers treat the problem in the form we define here.

$[\![x \geq d_{i+1}]\!], d_i < d \leq d_{i+1}$; $[\![x \geq d]\!] \equiv 0, d > d_m$; $[\![x > d]\!] \equiv [\![x \geq d+1]\!]$; $[\![x < d]\!] \equiv \neg[\![x \geq d]\!]$; and $[\![x \leq d]\!] \equiv \neg[\![x > d]\!]$.

The *binary* encoding of x introduces n encoding variables $[\![\mathsf{bit}(x,k)]\!], k \in 0..n-1$. Then, $[\![\mathsf{bit}(x,k)]\!]$ is true iff the kth most significant bit in the two's complement binary representation of the value assigned to x is true, i.e, in C notation: $\mathsf{bit}(x,k) =$`(x >> k) & 1`. We consider two cases: where x is known to be non-negative ($d_1 \geq 0$), and where it may take both positive and negative values. In the first case the number of bits required $n = \lceil log(ub(x)+1) \rceil)$ is given by the upper bound of x. In the second case the number of required bits $n = max(\lceil log(-lb(x)) \rceil, \lceil log(ub(x)+1) \rceil) + 1$ is determined by the lower and upper bounds of x. Let $\mathcal{B}(x) = 0..n-1$ and let $\mathcal{R}(x)$ be the set of representable integers for the binary encoding of x. In the first case $\mathcal{R}(x) = 0..2^n - 1$ and in the second $\mathcal{R}(x) = -2^{n-1}..2^{n-1} - 1$.

The *consistency constraint* for the binary encoding enforces that the Boolean representation can only represent values in the initial domain $D_i(x)$. It is the SAT encoding of the constraints (a) $x \geq d_1$ if $d_1 > 0$ or $0 > d_1 > -2^{-n-1}$, (b) $x \leq d_m$ if $d_1 \geq 0 \wedge d_m < 2^{n-1}$ or $d_1 < 0 \wedge d_m < 2^{n-2}$ (c) $x \neq d$ for each $d_1 < d < d_m, d \notin D_i(x)$. Constraints (a) and (b) are encoded using lexicographic decompositions, while (c) can be simply encoded as a clause $\vee_{k \in \mathcal{B}(x)} [\![\mathsf{bit}(x,k)]\!] \neq \mathsf{bit}(d,k)$. For highly sparse domains more efficient approaches are possible for (c). We return to this later. Unfortunately, unit propagation on encodings of these consistency constraints does not achieve propagation completeness on the language of atomic constraints $BINARY(x) = \{false\} \cup \{\mathsf{bit}(x,k), \neg\mathsf{bit}(x,k) \mid x \in \mathcal{X}, k \in \mathcal{B}(x)\}$. The binary encoding is also known as the log(arithmic) encoding. A crucial advantage of the binary encoding is that it is exponentially smaller than the other encodings.

4 Encoding constraints

Encoding constraints into clauses is a rich area of research. Even for simple constraints defined on Boolean variables such as at-most-one, cardinality, or psuedo-Boolean constraints there are dozens of papers [**?**,**?**,**?**,**?**,**?**]. Here we consider encoding constraints over integers. A key consideration is that the encoding of a constraint crucially depends on how the integers it constrains are encoded. We will indicate whether the direct, order or binary encoding is used for a particular integer x as $x{:}\mathbb{D}$, $x{:}\mathbb{O}$ or $x{:}\mathbb{B}$, respectively.

Most encodings of constraints are *uniform*, that is they expect all the integers involved in the constraint to be encoded in the same way (e.g., $x{:}\mathbb{O} \leq y{:}\mathbb{O}$). But different constraints will prefer different integer encodings. The cardinality constraints, such as , count how many times particular values occur in an array. Hence, they essentially *prescribe* a direct encoding of integers in the array. Similarly, linear constraints involving large domains essentially blow up in encoding size unless we use binary encoded integers. Many discrete optimisation models will therefore need to deal with different integers being encoded in different ways.

Hence, somewhere in the model we must encode constraints with non-uniform encodings of integers (e.g., $x{:}\mathbb{O} \leq y{:}\mathbb{B}$). We call these *coupling* encodings.

There is little existing work on coupling encodings except for channelling constraints, that is, coupling encodings of equality and coupling encodings of linear inequality constraints [**?**].

4.1 Coupling equality constraints

We can couple different integer encodings by allowing a variable x to have multiple encodings, such as $x{:}\mathbb{D}$ and $x{:}\mathbb{O}$, and adding a *channelling* equality constraint between them to our model (e.g., $x{:}\mathbb{D} = x{:}\mathbb{O}$). We now discuss the three coupling encodings of the integer equality constraint $x = y$.

$x{:}\mathbb{D} = y{:}\mathbb{O}$ This is the most common channelling constraint. Its encoding amounts to just defining $[\![x = k]\!]$ in terms of the order variables, for all $k \in D_i(x)$:

$$\forall_{d \in D_i(x)} [\![x = d]\!] \leftrightarrow [\![y \geq d]\!] \wedge [\![y \leq d]\!]. \tag{3}$$

Having introduced this for all d, we no longer need consistency constraints on the direct encoding. It is easy to show [**?**] that unit propagation for **??** for y and (**??**) is propagation complete for $x \in D(x) \wedge x = y \wedge y \in D(y)$ and $\mathcal{L} = DIRECT(x) \cup ORDER(y)$. Hence, we can omit the exactly-one constraint, **??** on x.

$x{:}\mathbb{D} = y{:}\mathbb{B}$ For this constraint, there is a trade-off between propagation strength and the size of the encoding. This encoding was initially devised in the context of encoding at-most-one [**?**,**?**,**?**] constraints, where channelling to the binary representation implicitly prevents two $[\![x = d]\!]$ literals from becoming true:

$$\forall_{d \in D_i(x), k \in \mathcal{B}(y)} [\![x = d]\!] \rightarrow [\![\mathsf{bit}(y, k)]\!] = \mathsf{bit}(d, k). \tag{4}$$

It is extended by Frisch and Peugniez [**?**] to exactly-one by adding channelling from the binary representation back to the equality literals:

$$\forall_{d \in D_i(x)} \bigwedge_{k \in \mathcal{B}(y)} [\![\mathsf{bit}(y, k)]\!] = \mathsf{bit}(d, k) \rightarrow [\![x = d]\!]. \tag{5}$$

This encoding does not enforce domain consistency between the two representations. For example, consider the case where we have removed all the even values from $D(x)$. In order to achieve domain consistency, we replace this second set of clauses with the modified ones, for each $k \in \mathcal{B}(y), v \in \{0, 1\}$:

$$[\![\mathsf{bit}(y, k)]\!] = v \rightarrow \bigvee_{d \in D_i(x), \mathsf{bit}(d,k) = v} [\![x = d]\!]. \tag{6}$$

This ensures that once all values consistent with $\mathsf{bit}(x, d) = v$ are eliminated, $[\![\mathsf{bit}(x, k)]\!]$ is set to $\neg v$.

As with $x{:}\mathbb{D} = y{:}\mathbb{O}$, the $x{:}\mathbb{D} = y{:}\mathbb{B}$ constraint allows us to drop consistency constraints: both the exactly-one (for direct), and all the binary encoding consistency constraints.

Theorem 1. *Unit propagation on the clauses of* **????** *is propagation complete for constraint* $x \in D(x) \wedge x = y \wedge y \in D(y)$ *and* $\mathcal{L} = DIRECT(x) \cup BINARY(y)$. *Hence, it enforces domain consistency on* x. □

$x{:}\mathbb{O} = y{:}\mathbb{B}$ We are not aware of any previous channelling between order and binary representations. The information that can be exchanged between the order and binary representations is necessarily limited, and can only propagate in two cases. If the lower/upper bound rests on a value d which is inconsistent with some fixed bit $[\![\mathsf{bit}(y,k)]\!]$, we can adjust the bound to the nearest value d' such that $\mathsf{bit}(d',k) = [\![\mathsf{bit}(y,k)]\!]$; and if a bit is constant for all values within the current bounds, we can fix the corresponding variable. A *stable segment* $l..u$ for binary bit k is a range $l..u \subseteq \mathcal{R}(y)$ such that $\mathsf{bit}(d,k)$ is the same for all $d \in l..u$. We can handle both of these cases by posting, for each segment $r_l..r_u \subseteq \mathcal{R}(y)$ which is stable for bit k, the constraint:

$$[\![\mathsf{bit}(y,k)]\!] = \mathsf{bit}(r_l,k) \vee [\![x < r_l]\!] \vee [\![x > r_u]\!]. \tag{7}$$

Example 1. Consider a variable y with $D_i(y) = 0..6$. In the binary representation, the second last bit is fixed for ranges of size 2: $\{\{0,1\},\{2,3\},\{4,5\},\{6\}\}$. The channelling constraints for the second last bit, $k = 1$, are

$$\begin{array}{l} \neg[\![\mathsf{bit}(y,1)]\!] \vee [\![x \geq 2]\!], \\ \quad [\![\mathsf{bit}(y,1)]\!] \vee [\![x \leq 1]\!] \vee [\![x \geq 4]\!], \\ \neg[\![\mathsf{bit}(y,1)]\!] \vee [\![x \leq 3]\!] \vee [\![x \geq 6]\!], \text{and} \\ \quad [\![\mathsf{bit}(y,1)]\!] \vee [\![x \leq 5]\!]. \end{array}$$

The Order-Binary channelling constraints are propagation complete, and require $O(|\mathcal{B}(y)||D_i(x)|)$ ternary clauses. We can omit the outer bounds consistency constraints on the binary encoding since they are enforced by the channel.

Theorem 2. *Unit propagation on the clauses of* **????** *is propagation complete for constraint* $x \in lb(x)..ub(x) \wedge x = y$ *and language* $\mathcal{L} = ORDER(x) \cup BINARY(y)$. *Hence, it enforces the initial bounds on* x. *Note that these equations do not enforce (nor can the encoding variables represent) domain consistency on the domain of* x. □

4.2 Coupling inequality constraints

While the channelling constraints above are propagation complete, introducing a dual representation of a variable and its channelling constraint introduce overhead. When we can avoid this and directly encode a constraint that *couples* two representations we can get more compact models.

Indeed, linear inequality constraints have a simple approach to coupling encodings, which is perhaps folklore. Each encoding can represent the linear term ax as a linear pseudo-Boolean term: direct $a \times d_1 \times [\![x = d_1]\!] + \cdots + a \times d_m \times [\![x = d_m]\!]$, order $a \times d_1 + a \times (d_2 - d_1) \times [\![x \geq d_2]\!] + \cdots + a \times (d_m - d_{m-1}) \times [\![x \geq d_m]\!]$, or binary $a \times [\![\mathsf{bit}(x,0)]\!] + 2a \times [\![\mathsf{bit}(x,1)]\!] + \cdots + 2^k a \times [\![\mathsf{bit}(x,k)]\!]$. Hence, an

arbitrary linear constraint can be mapped to a linear pseudo-Boolean constraint which is then encoded in any number of ways. Note that we can similarly encode arbitrary integer linear objectives into a pseudo-Boolean objective required for MaxSAT.

Diet-Sugar [**?**] improves on this simple approach for order+binary coupling[4] of $ax + b_1y_1 + b_2y_2 + b_ny_n \leq c$ by essentially encoding the constraints $ad_i[\![x \geq d_i]\!] + b_1y_1 + b_2y_2 + b_ny_n \leq c$ for each $d_i \in D_i(x)$ separately, using a BDD to encode each resulting pseudo-Boolean. Although they do not prove it, using the CompletePath encoding (from [**?**]) of the resulting BDDs will guarantee propagation completeness on the language $ORDER(x) \cup BINARY(y_1) \cup \cdots \cup BINARY(y_n)$.

We consider the simpler inequality coupling constraint $x{:}\mathbb{O} \leq y{:}\mathbb{B}$, which we can encode directly as follows.

For each $lb(x) < d \leq ub(x)$ we post the constraint

$$[\![x \geq d]\!] \rightarrow \bigvee_{k \in \mathcal{B}(x), \neg \mathsf{bit}(d-1,k)} [\![\mathsf{bit}(y,k)]\!] \tag{8}$$

The intuition behind this is that the binary representation for d satisfies this because adding any positive amount to $d-1$ will flip at least one bit from false to true. The encoding consists of $O(|ub(x) - lb(x)|)$ clauses of size $O(|\mathcal{B}(y)|)$. If $D_i(x)$ is not a range then many clauses within domain gaps become redundant. To complete the encoding we need to enforce the initial bounds using the lexicographic encoding of $y \geq lb(x)$ and the unit clause encoding of $x \leq ub(y)$. To extend to ranges with negative numbers we treat the sign bit $[\![\mathsf{bit}(x, n-1)]\!]$ as if it were negated, but omit details here.

Example 2. Consider encoding $x{:}\mathbb{O} \leq y{:}\mathbb{B}$ for variables x and y with $D_i(x) = 1..7$, $D_i(y) = 0..6$. This will result in the following clauses from **??**, shown with the relevant bit representation on the left.

$$\begin{array}{lllll}
\texttt{001} & [\![x \geq 2]\!] \rightarrow [\![\mathsf{bit}(y,2)]\!] & \vee [\![\mathsf{bit}(y,1)]\!] & & \\
\texttt{010} & [\![x \geq 3]\!] \rightarrow [\![\mathsf{bit}(y,2)]\!] & \vee & & [\![\mathsf{bit}(y,0)]\!] \\
\texttt{011} & [\![x \geq 4]\!] \rightarrow [\![\mathsf{bit}(y,2)]\!] & & & \\
\texttt{100} & [\![x \geq 5]\!] \rightarrow & & [\![\mathsf{bit}(y,1)]\!] & \vee [\![\mathsf{bit}(y,0)]\!] \\
\texttt{101} & [\![x \geq 6]\!] \rightarrow & & [\![\mathsf{bit}(y,1)]\!] &
\end{array}$$

The encoding also requires $y \geq 1$ encoded as $[\![\mathsf{bit}(y,0)]\!] \vee [\![\mathsf{bit}(y,1)]\!] \vee [\![\mathsf{bit}(y,2)]\!]$ and $x \leq 6$ encoded as $\neg[\![x \geq 7]\!]$. We see that for example $[\![x \geq 4]\!]$, which forces $[\![\mathsf{bit}(y,2)]\!]$ true, means that it never occurs for later literals. This relies on the order encoding, since $[\![x \geq 5]\!]$ implies $[\![x \geq 4]\!]$.

If we suppose instead $D_i(x) = \{1, 2, 6, 7\}$, then we keep only the first, third and last clauses, since the others are subsumed (i.e., $[\![x \geq 3]\!] \equiv [\![x \geq 4]\!] \equiv [\![x \geq 5]\!] \equiv [\![x \geq 6]\!]$). □

This encoding is propagation complete for $x \leq y$.

[4] Their approach handles any number of order encoded variables, we restrict to one for simplicity of explanation.

Theorem 3. *Unit propagation on the clauses of* **????** *and clauses for lexicographic encoding of* $y \geq lb(x)$ *and together with the clause* $\neg[\![x \geq ub(y) + 1]\!]$ *is propagation complete for* $x \leq y$ *for the language* $\mathcal{L} = ORDER(x) \cup BINARY(y)$. □

Interestingly the encoding of Diet-Sugar applied to the constraint $x{:}\mathbb{O} \leq y{:}\mathbb{B}$ produces a strict superset of our encoding clauses, adding many redundant clauses.

We can enforce the reverse $x{:}\mathbb{O} \geq y{:}\mathbb{B}$ similarly. Apart from the initial bounds constraints $x \geq lb(y)$ and $y \leq ub(x)$ we generate the clauses for $lb(y) < d \leq ub(x)$

$$\neg[\![x \geq d]\!] \rightarrow \bigvee_{k \in \mathcal{B}(y), \mathsf{bit}(d,k)} \neg[\![\mathsf{bit}(y,k)]\!] \tag{9}$$

. We can of course use the conjunction of inequalities, $x{:}\mathbb{O} \geq y{:}\mathbb{B} \wedge x{:}\mathbb{O} \leq y{:}\mathbb{B}$, to enforce equality $x{:}\mathbb{O} = y{:}\mathbb{B}$, but this *inequality channel* does not enforce propagation completeness.

4.3 Coupling element constraints

Element constraints are an important component of many CP models. They enable array look-up using a variable index with $y = A[x]$. Our encoding makes use of the fact that the equality constraint $x = d$ for fixed integer d is a conjunction of literals for the direct, order or binary encoding: $[\![x = d]\!]$, $[\![x \geq d]\!] \wedge [\![x \leq d]\!]$, or $\bigwedge_{k \in \mathcal{B}(x)} [\![\mathsf{bit}(x,k)]\!] = \mathsf{bit}(d,k)$, respectively. Consequently, we can define a coupling encoding of constraint for any encoding choices for y, A and x with for all $d \in D_i(x)$, $(x = d) \rightarrow (A[d] = y)$. In other words, the equality constraint $A[d] = y$ is conditional on the conjunction of literals, $x = d$.

Example 3. Consider the coupling for $y{:}\mathbb{O} = A{:}\mathbb{B}[x{:}\mathbb{D}]$. The clauses take the form

$$\neg[\![x = d]\!] \vee ([\![\mathsf{bit}(A[d],k)]\!] = \mathsf{bit}(r_l,k)) \vee [\![y < r_l]\!] \vee [\![y > r_u]\!]$$

for $d \in D_i(x)$ and stable ranges $r_l..r_u \subseteq R(y)$ for bit k. □

When x has the direct encoding and A is fixed, the encoding can be made propagation complete similar to **??** by adding a backwards clauses for every distinct value u in A: $(y = u) \rightarrow \bigvee_{d \in D_i(x), u = A[d]} [\![x = d]\!]$.

5 Views

Views [**?**] are a crucial feature in modern CP solvers. For specific constraints, views can simplify propagation construction using an interface to the variable operations. In SAT, whenever an equivalence or negation arises between different expressions, we can represent both with the same Boolean variable. This allows greater scope for views on SAT encoded integers compared to CP integers.

Affine transformations The most important view constraint is the affine transformation $y = ax + b$ for fixed integers a and b. For the direct encoding, we have for every $d \in D(x) \cup D(y)$ that $[\![y = ad + b]\!] \equiv [\![x = d]\!]$, so we can use the same Boolean variable to represent both. For the order encoding, we have for non-negative a that $[\![y \geq ad + b]\!] \equiv [\![x \geq d]\!]$ (if a is negative, one $\geq$ flips to $\leq$). For the binary encoding, affine views can be applied in some but not all cases [**?**].

Minimum/maximum Boolean encoding also raises the possibility of *partial views* where only some Boolean variables are reused. In $y = \max(x, m)$ for fixed integer m (and similar for min), after enforcing $y \geq m$ we find a partial view for the order encoding $\forall_{d=m}^{ub(y)} [\![y \geq d]\!] \equiv [\![x \geq d]\!]$. For the direct encoding, we have one less equivalence: $\forall_{d=m+1}^{ub(y)} [\![y = d]\!] \equiv [\![x = d]\!]$. We require one new Boolean variable for $[\![y = m]\!]$, which is constrained by $\forall_{d=lb(x)}^{m} [\![x = d]\!] \rightarrow [\![y = m]\!]$.

Element Certain compositions of fixed A allow for views. For $y{:}\mathbb{D} = A[x{:}\mathbb{D}]$, every unique value $A[d]$ has binary backwards clause $[\![y = A[d]]\!] \rightarrow [\![x = d]\!]$, so $[\![x = d]\!] \equiv [\![y = A[d]]\!]$. For $y{:}\mathbb{O} = A[x{:}\mathbb{O}]$, we have for all $d \in D_i(x)$ that $[\![x \geq d]\!] \rightarrow [\![y \geq A[d]]\!]$ if $A[d] \leq A[e]$ holds for all $e > d$. Similarly, $[\![x \leq d]\!] \rightarrow [\![y \leq A[d]]\!]$ if $A[c] \leq A[d]$ holds for all $c < d$. If both conditions hold, then $[\![x \geq d]\!] \equiv [\![y \geq A[d]]\!]$. If A is strictly monotone, both conditions hold for all elements and y becomes a total view of x. When the inequalities are flipped, we have negated views $[\![x \geq d]\!] \equiv [\![y < A[d]]\!]$, which are total if A is strictly antitone.

Views allow us to extend our coupling encodings straightforwardly. For example, we can encode $x{:}\mathbb{O} + d \leq y{:}\mathbb{B}$ by using a view to construct $(x + d){:}\mathbb{O}$ and then using the inequality coupling.

6 Experimental Results

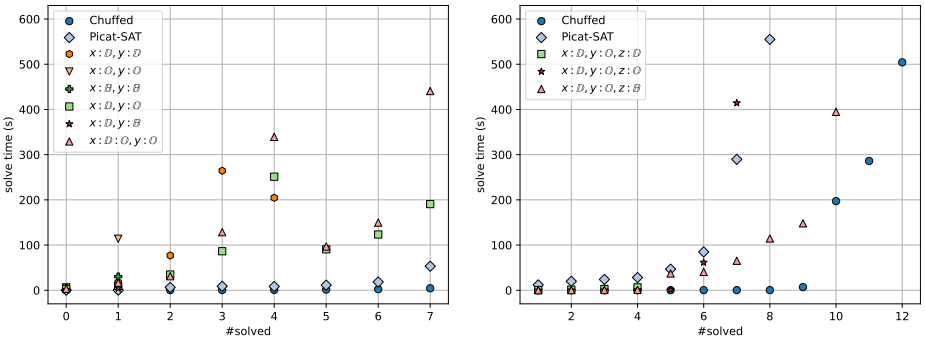
To validate the benefit of coupling different integer encodings, we created the MiniZinc-SAT framework, an extension of MiniZinc. Through MiniZinc's *annotations*, the user can declare the integer variable encodings. For example in the knight's tour model, given in **??**, we can choose direct encoding for each variable in `x` by adding an annotation as follows.

Since annotations are first class objects, the user can specify more complex encoding schemes based on, for example, the variable's domain size, its existing encodings, or any other contextual rules. If no encodings are specified, default choices are made. During compilation, MiniZinc-SAT resolves mixed encoding constraints by coupling if possible, or by channelling if necessary. Additionally, views are used whenever the constraint and variable encodings allow it.

Using MiniZinc-SAT, we create various encodings of realistic problems to see if we can improve solver performance. We also compare with Picat-SAT 3.1 [**?**], a solver that maps MiniZinc to SAT using binary encoding, and Chuffed 0.10.4 [**?**] a lazy clause generation solver, using lazy direct-order encoding.

Results are given for SAT models and MaxSAT models running Open-WBO 2.1 [**?**] on a single-core *Intel Xeon 8260* CPU (non-hyperthreaded) with a 10-minute timeout and 10 GB of RAM. The results are visualised in cactus plots, in which for each configuration the solved instances are sorted by solve time (measured from when the SAT or LCG solver is started). Instances which are not solved (SAT) or proved optimal (MaxSAT) within the time limit are omitted.

Source code, models, instances and run logs are available online[5].



(a) (b)

Fig. 1: Cactus plots for solved (**??**) `knights` instances, and (**??**) `orienteering` instances.

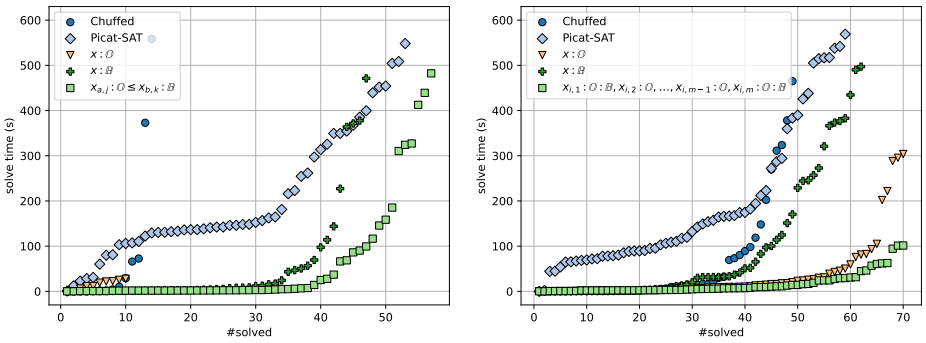
Knight's tour The first example is the `knights` model shown in **??** for instances $n = 8, 10, ..., 22$. The model contains n^2 variables x with relatively sparse domains of up to size 8, but to prevent sub-cycles the decomposition introduces another n^2 variables y with contiguous domains of $1..n^2$ that represent the order in which positions are visited. The encodings of x and y are coupled through constraints, extended by a view on the result variable (e.g., $y{:}\mathbb{O}[x_i{:}\mathbb{D}] = (y_{i-1} + 1){:}\mathbb{O}, \forall 1 < i \leq n$). The results for the three uniform and three sensible mixed encoding choices for x and y are shown in **??**.

Uniform order and binary encodings do not succeed beyond the two easiest instances. This is unsurprising since the `circuit` constraint prescribes $x{:}\mathbb{D}$. However, since the y variables reason about the order of visits, we see that $y{:}\mathbb{O}$ is clearly preferred over the uniform approach, $y{:}\mathbb{D}$. The worst choice is $y{:}\mathbb{B}$, which has the additional disadvantage that it cannot use the affine transformation view. Creating a redundant order encoding for x variables ($x{:}\mathbb{D}{:}\mathbb{O}$) does not seem to make much difference. Chuffed and Picat-SAT both far outperform MiniZinc-SAT, since they have native propagator and encoding [**?**] respectively.

[5] https://github.com/hbierlee/cpaior-2022-coupling-sat

Orienteering The `orienteering` problem is a COP concerning a complete graph with edge distances $d_{i,j}$ and node rewards. The aim is to find a path from start to finish node that maximises the sum of the rewards of the visited nodes, but which is limited by a linear inequality on the distances of the traversed edges. The principal constraint is encoded similarly to in `knights` by two sets of variables x and y, coupled through `element` constraints. Given the results from `knights`, we will use $x{:}\mathbb{D}$ and $y{:}\mathbb{O}$. However, we will experiment with all possible (non-redundant) encodings on a new set of coupled variables $z_i = d_i[x_i]$ which appear in the linear inequality on the maximum distance. The linear objective can be directly encoded using the $x{:}\mathbb{D}$ variables.

A cactus plot comparing encodings is shown in **??**. Of the mixed encodings, $z{:}\mathbb{B}$ is clearly the best, since it makes the path length constraint compact. While both Chuffed and Picat-SAT again have native propagator and encoding, Picat-SAT is now outperformed by the best MiniZinc-SAT encoding. It seems that its encoding is less effective than the MiniZinc standard decomposition using element coupling.



(a) (b)

Fig. 2: Cactus plots for solved `jsswet` instances of (**??**) variant where 40% of jobs are high priority, and of (**??**) variant with limited average job length.

Job-shop scheduling with weighted earliness/lateness Here, we examine a job-shop scheduling problem `jsswet` over n jobs, each with m tasks, with start time variables x. Every task j of job i runs on a different machine for its full duration $d_{i,j}$. Tasks of the same job must finish in sequence ($x_{i,j} + d_{i,j} \leq x_{i,j+1}$), and must not overlap (`disjunctive`) with the tasks of other jobs that run on the same machine. The objective is to minimise the sum of the penalties over all jobs. A job's penalty is its final task's earliness or lateness to its deadline t_i, weighted by its earliness e_i or lateness l_i penalty coefficients, respectively. The order encoding combines affine transformations and max views for the objective: $e_i \times \max(0, t_i - x_{i,m} + d_i){:}\mathbb{O} + l_i \times \max(0, x_{i,m} + d_i - t_i){:}\mathbb{O}$.

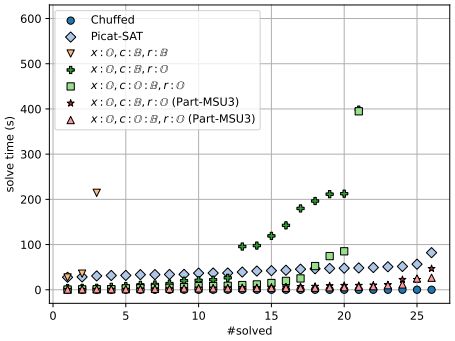


Fig. 3: Cactus plots for solved `table-layout` instances.

In preliminary results not shown here we found that if the schedule's horizon is small, the order encoding works best thanks to its propagation strength. However, when the horizon becomes large this approach will run out of memory due to the sheer number of order encoding variables, and only the binary encoding remains viable. So each encoding has benefits and drawbacks. We now consider two variants of the problem.

In the first variant, we consider a large horizon, but designate the first 40% of jobs as *high priority* jobs which must finish within 500 time steps of their deadline, and have much higher penalties. Effectively, this splits the variables into those with small and large domains. We consider encoding all variables with order, or binary, or a mix which uses order for small and binary for large domains. For high priority job-task $x_{a,j}$ and low priority job-task $x_{b,k}$ assigned to the same machine, the `disjunctive` constraint couples the (potentially) mixed encodings via $((x_{a,j} + d_{a,j}){:}\mathbb{O} \leq x_{b,k}{:}\mathbb{B}) \vee (x_{b,k}{:}\mathbb{B} \leq (x_{a,j} - d_{b,k}){:}\mathbb{O})$. The results in **??** show that the mixed encoding outperforms the other solvers and uniform encodings.

For the other variant we consider a smaller horizon, which allows a full order encoding of all tasks. However, now the average run time of all jobs is limited by parameter M (2.5 times the sum of all minimal job durations). To effectively constrain this linear inequality $\sum_{i=1}^{n} x_{i,m} - x_{i,1} \leq M$, we add a redundant binary encoding to the first and last task of each job, but *not* on the in-between tasks: $x_{i,1}{:}\mathbb{O}{:}\mathbb{B}, x_{i,2}{:}\mathbb{O}, \ldots, x_{i,m-1}{:}\mathbb{O}, x_{i,m}{:}\mathbb{O}{:}\mathbb{B}$.

The results in **??** show the mixed encoding convincingly outperforms the other solvers and uniform encodings. The redundant encodings are coupled with $x_{i,1}{:}\mathbb{O} = x_{i,1}{:}\mathbb{B}$, but coupling with $x_{i,1}{:}\mathbb{O} \leq x_{i,1}{:}\mathbb{B} \wedge x_{i,1}{:}\mathbb{O} \geq x_{i,1}{:}\mathbb{B}$ or just $x_{i,1}{:}\mathbb{O} \leq x_{i,1}{:}\mathbb{B}$ produces similar results.

Table Layout Finally, we consider the `table-layout` problem. For a table composed of $n \times m$ cells, our task is to assign a width-height configuration variable $x_{i,j}$ for each cell at row i, column j. The configuration will determine for cell i, j its cell-width $w_{i,j} = W[x_{i,j}]$ and cell-height $h_{i,j} = H[x_{i,j}]$ through `element` constraints. These variables are combined in n row-height variables r_i, where

$\bigwedge_{j=1}^{m} r_i \geq h_{i,j}$, and m column-width variables c_j, where $\bigwedge_{i=1}^{m} c_j \geq w_{i,j}$. The objective is then to minimise the table's height $\sum_{i=1}^{n} r_i$, without the table's width $\sum_{j=1}^{m} c_j$ exceeding a given maximum width.

We chose this problem to test two unexplored properties. First, $H_{i,j}$ and $W_{i,j}$ are guaranteed to be respectively monotone and antitone, since the width-height configurations represent sorted, optimal text layouts. Consequently, the element constraint establishes a view between the order encoding variables $[\![x_{i,j} \geq k]\!]$, $[\![h_{i,j} \geq H_{i,j,k}]\!]$ and $[\![w_{i,j} \leq W_{i,j,k}]\!]$. Secondly, $D(h_{i,j})$ is sparse whereas $D(r_i)$ is contiguous. Thus, the coupling $h_{i,j}{:}\mathbb{O} \leq r_i{:}\mathbb{B}$ requires far fewer clauses for the domain gaps in $h_{i,j}$, while also skipping a large order encoding of r_i. We compare this very compact encoding against an order encoded objective (using simple binary clauses for $h_{i,j}{:}\mathbb{O} \leq r_i{:}\mathbb{O}$). A third approach creates an order encoding for the width ($w_{i,j}{:}\mathbb{O} \leq c_j{:}\mathbb{O}$) as well, but still redundantly channels $c_j{:}\mathbb{O} = c_j{:}\mathbb{B}$ for the linear inequality.

The results in **??** show that solver performance suffers greatly if we couple $x{:}\mathbb{O}$ to a binary encoded objective $r{:}\mathbb{B}$ rather than using straightforward order encoded objective $r{:}\mathbb{O}$. The coupling perhaps overcomplicates the objective compared to using binary clauses. Furthermore, channelling rather than coupling to $c{:}\mathbb{B}$ seems to be marginally better as well. The domain sizes of the `table-layout` instances are too large for the more compact coupling to pay off (in contrast to `jsswet`). For $r{:}\mathbb{O}$, the pseudo-Boolean objective is unweighted, which means Open-WBO can use its core-guided Part-MSU3 algorithm. This makes the models solve almost instantly. Chuffed does equally well, while Picat-SAT solves all instances but requires more time.

7 Related Work

The choice of variable encoding is a critical decision when encoding CSPs to SAT, and unsurprisingly has seen considerable attention. Most CSP-to-SAT converters fix one of the encodings described in **??** as their core representation, and convert all variables/constraints based on that encoding. Binary encoding is a popular choice (in two's complement [**?**] or sign-magnitude [**?**] variants) despite its relatively weak propagation strength, as it can reliably cope with very large domains. Order encoding, conversely, is adopted by some encoders [**?**,**?**] despite the risk of blow-up due to its effectiveness on primitive arithmetic constraints [**?**]. This is sometimes paired with a channelled direct encoding for flexibility [**?**,**?**]. Not to be confused with our order-binary channelling, some works propose new integer encodings which mix features of the order and binary encoding [**?**,**?**].

Some works are concerned with picking the right encoding for the right variable. `Proteus` [**?**] attempts to predict, for a given instance, which encoding of variables (and constraints) will be most effective, then commits to that encoding for the whole instance. `Satune` [**?**] selects encodings on a per-variable basis, and optimises the domain representation, based on a training phase for a class of instances. However, it does not resolve the coupling problem, instead requiring connected variables to share a common representation. A more radical approach,

adopted by current lazy clause generation solvers [?,?,?] is to implicitly maintain a partial representation of the direct-order encoding, introducing new literals and channelling as necessary.

`Diet-Sugar` [?] is the only other work that we are aware of that considers encoding non-uniform constraints. It directly introduces coupled order and binary encodings of linear constraints. The SAT translation chooses a single encoding for each variable using a heuristic that leads to an overall small and effective encoding of the constraints. The resulting mixed encodings are shown to yield significant improvement.

General compilers that support MiniZinc have been developed using binary encoding, namely `FznTini` [?,?] (two's complement) and `Picat-SAT` [?] (sign-and-magnitude). Notably, the binary encoding despite theoretically lower propagation strength has continued to prove itself in the Picat-SAT compiler in the yearly MiniZinc competition. It is clear that the black box compilers such as Picat-SAT essentially introduce other forms of integer encoding in some global constraint decompositions [?]. Savile Row [?] similarly converts a high-level model (specified in Essence') into SAT. Savile Row commits to a channelled direct-order encoding for variables, allowing each constraint to choose its preferred uniform encoding, though they apply some transformations [?] to improve the generated SAT encoding.

BEE [?] is an approach to compiling integer models to SAT using only order encoding. It goes beyond our view approach by searching for all Boolean variable equivalences (using a SAT solver) in order to simplify the resulting SAT model. BEE would automatically discover the (partial) views we define on order encoded variables, but none of them are covered by its ad hoc methods.

8 Conclusion

In conclusion, while compilation to SAT is a competitive approach to tackling discrete optimisation problems, efficient encoding of discrete optimisation problems into SAT is challenging. To create the best encodings possible we must allow different representations of integers to be used in the encoding, which means the problem of coupling encodings arises. In this paper we show how we can create coupled models by using channelling equalities, or directly by using coupled encodings of element or inequality constraints. Total and partial views extend the coupling constraints we can encode. We show that coupling encodings can be required for getting the best resulting SAT encoding.

Acknowledgements This research was partially funded by the Australian Government through the Australian Research Council Industrial Transformation Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Project ID IC200100009.

A Proofs

Theorem 1. *Unit propagation on the clauses of* **????** *is propagation complete for constraint* $x \in D(x) \wedge x = y \wedge y \in D(y)$ *and* $\mathcal{L} = DIRECT(x) \cup BINARY(y)$. *Hence, it enforces domain consistency on* x.

Proof. First we show that unit propagation enforces the exactly-one constraint. If we have $[\![x = d]\!]$ *and* $[\![x = d']\!]$ *true simultaneously, then* **??** *will force one binary variable in* y *to take two values, thus failing. If all variables* $[\![x = d']\!]$ *are set false except one* $[\![x = d]\!]$, *then* **??** *will set the correct bits of* y *in the remaining solution (via setting the negation to false). The forward direction will then propagate* $[\![x = d]\!]$.

Next we show that the unit propagation is propagation complete with respect to the binary variables and missing values in the original domain. Suppose $S \subseteq D(x)$ *are the remaining values in the domain. Then* $\neg[\![x = d]\!]$ *is set for* $d \in D_i(x) - S$. *Suppose that there is no support for* $[\![\mathsf{bit}(y, k)]\!]$ *in* S. *Then unit propagation of* **??** *will set* $\neg[\![\mathsf{bit}(y, k)]\!]$.

Finally, we show that unit propagation is propagation complete on the encoding variables. Given the subset of the current assignment literals $L \subseteq \phi$ *restricted to the direct encoding variables for* x *and binary encoding variables of* y. *Let* $S \subset D(x)$ *be the set of possible domain values remaining, i.e.* $S = \{d \mid d \in D(x), \neg[\![x = d]\!] \notin L\}$. *Clearly the direct encoding variables are consistent with this by definition, and if* S *is a singleton, by the first argument* $[\![x = d]\!]$ *is set true. We now consider each bit variable* $[\![\mathsf{bit}(y, k)]\!]$*: if* $[\![\mathsf{bit}(y, k)]\!] \in L$ *then by* **??** *each value in* $d \in S$ *has* $\mathsf{bit}(d, k)$ *is true, hence it is supported; similarly if* $\neg[\![\mathsf{bit}(y, k)]\!] \in L$. *Finally, if variable* $[\![\mathsf{bit}(y, k)]\!]$ *does not appear in* L, *then there must be values in* S *with both truth values, otherwise one of the equations in* **??** *would have propagated.*

Theorem 2. *Unit propagation on the clauses of* **????** *is propagation complete for constraint* $x \in lb(x)..ub(x) \wedge x = y$ *and language* $\mathcal{L} = ORDER(x) \cup BINARY(y)$. *Hence, it enforces the initial bounds on* x. *Note that these equations do not enforce (nor can the encoding variables represent) domain consistency on the domain of* x.

Proof. Given the subset of the current assignment literals $L \subseteq \phi$ *restricted to the order encoding variables for* x *and binary encoding variables for* x, *we show there is support for each possible value defined by* L. *Since each bound or bit is supported or propagated, propagation completeness follows. The order literals in* L *define a range* $l..u$ *given by* $l = \max\{i \mid [\![x \geq i]\!] \in L\}$ *and* $u = \min\{i - 1 \mid \neg[\![x \geq i]\!] \in L\}$. *The order consistency* **??** *enforces that* $\neg[\![x \geq i]\!] \in L$ *for* $i > u$ *and* $[\![x \geq i]\!] \in L$ *for* $i < l$. *We show* l *and* u *are supported. Suppose that* $[\![\mathsf{bit}(x, b)]\!] \neq \mathsf{bit}(l, b) \in L$ *for some bit* b. *Then there is some stable segment for this bit including* l, *say* $r_l..r_u$. *Then the clause* $[\![\mathsf{bit}(x, b)]\!] \vee [\![x < r_l]\!] \vee [\![x > r_u]\!]$ *will fire enforcing* $[\![x > r_u]\!]$ *and hence* l *cannot be the lower bound. A similar argument applies to the upper bound.*

We now show each possible value for each binary encoding variable $[\![\mathsf{bit}(x,b)]\!]$ *is supported in the range* $l..u$*. Suppose to the contrary that w.l.o.g.* $[\![\mathsf{bit}(x,b)]\!]$ *is not true for any* $x \in l..u$*. Then there is a stable segment* $r_l..r_u$ *at least as large as* $l..u$ *for* $\neg[\![\mathsf{bit}(x,b)]\!]$*. Then the clause* $\neg[\![\mathsf{bit}(x,b)]\!] \vee [\![x < r_l]\!] \vee [\![x > r_u]\!]$ *will propagate* $\neg[\![\mathsf{bit}(x,b)]\!]$*, which then must be in* L*. The same reasoning holds for the negation.*

The channelling also enforces the outer bounds on the binary encoding variables. Clearly $d_1 \leq l \leq u \leq d_m$ *so the arguments for support of binary encoding variables automatically take into account the initial bounds.*

Theorem 3. *Unit propagation on the clauses of* **????** *and clauses for lexicographic encoding of* $y \geq lb(x)$ *and together with the clause* $\neg[\![x \geq ub(y) + 1]\!]$ *is propagation complete for* $x \leq y$ *for the language* $\mathcal{L} = ORDER(x) \cup BINARY(y)$*.*

Proof. Suppose $\mathcal{D} \wedge x \leq y \rightarrow [\![x \leq d]\!]$ *we show that unit propagation will enforce this. The initial case given by* $y \leq ub(x)$ *is enforced by the last clause. Let* d *be the maximum value* y *can take given* $\mathcal{D}$*, then we know that* $\neg[\![\mathsf{bit}(y,k)]\!] \in \mathcal{D}$ *for all* $k \in \mathcal{B}(y), \mathsf{bit}(d,k) = 0$ *otherwise* y *could take a larger value. The clause* **??** *for* $[\![x \geq d+1]\!]$ *has right-hand side* $\bigvee_{k \in \mathcal{B}(y), \neg\mathsf{bit}(d,k)} [\![\mathsf{bit}(y,k)]\!]$ *and hence propagates* $\neg[\![x \geq d+1]\!]$ *as required.*

Let Y *be the set of all possibly values that* y *can take given the* $\mathcal{D}$*. Suppose* $\mathcal{D} \wedge x \leq y \rightarrow [\![\mathsf{bit}(y,k)]\!]$ *for some* $k \in \mathcal{B}(y)$*. We show that unit propagation will enforce this. If* x *still sits at its initial lower bound this will be forced by the encoding of* $y \geq lb(x)$*. Otherwise, this propagation was caused by* $[\![x \geq d]\!]$ *in the current domain, where* $d > \min(Y)$

So clearly (A) $\mathsf{bit}(v,k) = 1$ *for all* $v \in Y \cap d..\max(y)$*. Let (B)* $d' = \max\{d'' \mid d'' \in Y, \mathsf{bit}(d'',k) = 0\}$*. Clearly such a* d' *exists otherwise we would already have propagated* $[\![\mathsf{bit}(y,k)]\!]$*, and, since* $d' < d$*,* $[\![x \geq d'+1]\!]$ *is propagated by* **??***. We claim the clause* **??** *for* $[\![x \geq d'+1]\!]$ *will propagate* $[\![\mathsf{bit}(y,k)]\!]$*. Suppose to the contrary then there is another bit* k' *where* $\mathsf{bit}(d',k') = 0$ *where* $\neg[\![\mathsf{bit}(y,k')]\!] \notin \mathcal{D}$*. Then* $d' + 2^{k'} \in Y$ *and either* $d' + 2^{k'} \geq d$ *contradicting (A), or* $d' + 2^{k'} < d$ *contradicting (B).*

While it is not possible for $\mathcal{D} \wedge x \leq y \rightarrow \neg[\![\mathsf{bit}(y,k)]\!]$ *since a lower bound can never force a negative bit (although this can happen subsequently from the clauses for constraint* $y \leq ub(y)$*).*