# Effective Encodings of Constraint Programming Models to SMT

Ewan Davidson[1], Özgür Akgün[1], Joan Espasa[1], and Peter Nightingale[2]

[1] School of Computer Science, University of St Andrews, UK
{eld5, ozgur.akgun, jea20}@st-andrews.ac.uk
[2] Department of Computer Science, University of York, York, UK
peter.nightingale@york.ac.uk

**Abstract.** Satisfiability Modulo Theories (SMT) is a well-established methodology that generalises propositional satisfiability (SAT) by adding support for a variety of theories such as integer arithmetic and bit-vector operations. SMT solvers have made rapid progress in recent years. In part, the efficiency of modern SMT solvers derives from the use of specialised decision procedures for each theory. In this paper we explore how the Essence Prime constraint modelling language can be translated to the standard SMT-LIB language. We target four theories: bit-vectors (QF_BV), linear integer arithmetic (QF_LIA), non-linear integer arithmetic (QF_NIA), and integer difference logic (QF_IDL). The encodings are implemented in the constraint modelling tool Savile Row. In an extensive set of experiments, we compare our encodings for the four theories, showing some notable differences and complementary strengths. We also compare our new encodings to the existing work targeting SMT and SAT, and to a well-established learning CP solver. Our two proposed encodings targeting the theory of bit-vectors (QF_BV) both substantially outperform earlier work on encoding to QF_BV on a large and diverse set of problem classes.

**Keywords:** Constraint Modelling · SMT · Automated Reformulation

## 1 Introduction

Constraint programming (CP) is a powerful paradigm for solving constraint satisfaction and optimisation problems, with many diverse applications. Essence Prime [22] is a solver-independent CP modelling language that offers decision variables of Boolean and integer domains as well as arrays of these decision variables, arithmetic and logical operators for expressing constraints, and global constraints. Essence Prime is comparable in its modelling capabilities to MiniZinc [17], OPL [25], Simply [6] and other similar languages. To solve a problem instance described in Essence Prime, a problem class model and a parameter file are translated by Savile Row [21] into input suitable for a backend solver. Savile Row applies automated reformulation steps such as common sub-expression elimination to improve the model [21].

Satisfiability Modulo Theories (SMT) is a problem solving methodology that decides the satisfiability of a propositional formula with respect to a selection of theories in first order logic with equality [20]. SMT has its roots in the field of hardware and software verification, but lately it is being used to solve a wider range of problems. The Satisfiability Modulo Theories Library (SMT-LIB) [4] provides a standard for the specification of benchmarks and theories for SMT. Most SMT solvers are restricted to decidable quantifier free fragments of their logics, which is sufficient for many applications. SMT solvers have made rapid progress in recent years. In part, the efficiency of modern SMT solvers derives from the use of specialised decision procedures for each theory. In this work we will focus on four theories: *QF_LIA*, *QF_NIA*, *QF_IDL*, and *QF_BV*.

*QF_LIA* and *QF_NIA* are the theories of quantifier free linear and non-linear integer arithmetic respectively. The formulas are Boolean combinations of constraints comparing expressions to a constant with respect to $\leq$ and $\geq$. Linear expressions are typically enough to naturally express common problems in formal verification and scheduling problems. SMT solvers typically use variants of the Simplex algorithm to implement integer arithmetic theories [12].

*QF_IDL* is the theory of quantifier free integer difference logic. It supports Boolean combinations of inequalities of the form $x - y < b$ where $x$ and $y$ are integer variables and $b$ is an integer constant. The renewed interest in this fragment came from timed automata, where the verification conditions arising take the form of difference logic formulas [18].

*QF_BV* is the theory of quantifier-free formulas over fixed-size bit-vectors. Bit-vector arithmetic is very commonly used for verification and equivalence checking in the hardware industry. Current solvers [13] typically apply heavy preprocessing techniques that ultimately flatten the formula to SAT, also known as *bit-blasting*.

The main contribution of this paper is a new SMT backend for Savile Row that is able to produce any one of the four theories listed above and at two different levels of flattening, thus providing eight distinct SMT encodings. We exhaustively evaluate the performance of these encodings, showing that our encodings perform significantly better than existing SMT approaches [9] and that they are complementary to the existing SAT and CP backends of Savile Row.

## 2 Related Work

There has been a great deal of work on translating declarative constraint modelling languages to lower-level languages such as SAT, SMT, and MIP. We cannot cover it all here so we cover all the most relevant work (on translation to SMT) and give examples for the rest. One example of SAT encoding is FznTini [14], which translates FlatZinc into SAT. The MiniZinc compiler is able to translate the MiniZinc language into MIP [5]. There are several encodings of a constraint language to SMT: Simply [6,8], fzn2smt [7], and FZN2OMT [9].

Simply [6] is a compiler from a custom constraint modelling language (comparable to Essence Prime) to SMT. It supports translations to QF_LIA and

QF_IDL logics and was later extended to support meta-constraints and weighted CSPs [8]. The same authors presented fzn2smt [7], an approach that translates FlatZinc models to SMT, supporting the standard data types and constraints of FlatZinc. The logic used for solving each instance is determined automatically during the translation, and it handles optimisation problems by means of a dichotomic (binary) search on the domain of the optimisation variable. As in our approach, search annotations are ignored, as they do not make sense in the context of SMT. Only the allDifferent and Cumulative global constraints are supported by decomposing them into SMT.

FZN2OMT [9] can translate standard FlatZinc into suitable input for SMT solvers, and back to FlatZinc. It supports the Z3 [16] and OptiMathSAT [24] solvers, using either the QF_BV or QF_LIRA logics. When using OptiMathSAT, it takes advantage of the fact that OptiMathSAT natively supports a subset of the FlatZinc 1.6 language. Unlike fzn2smt, FZN2OMT is compatible with the current MiniZinc toolchain. We compare our proposed SMT encodings to all four configurations of FZN2OMT in Section 4 below.

## 3 Translating Essence Prime to SMT

The modelling tool Savile Row works by reading a problem class model written in Essence Prime and optionally a parameter file that contains values defining a problem instance. It instantiates the problem class model, unrolls comprehensions and quantified expressions, and performs expression *flattening* where the target solver language does not support nested expressions. Flattening is the task of introducing *auxiliary* variables to represent nested subexpressions. Section 3.3 explains flattening in the SMT backend.

The input to Savile Row is a solver-independent model with nested expressions. Savile Row already has existing backends to several CP/SAT/MIP solvers, some of these via FlatZinc. Some of these backends require a lower level output than others. For example, Minion's input language allows a limited form of nesting (for example, constraints may be contained in conjunctions and disjunctions) but still requires a mostly flat structure, whereas FlatZinc-based solvers do not allow this kind of nesting at all. SMT-LIB allows more kinds of nested expressions than any other Savile Row backend. We implement a fully-flat variant as well as a *nested* variant that only flattens when strictly necessary. It is difficult to know which variant will perform better for a given theory, SMT solver and problem class. Flattening introduces potentially useful new variables, and preserving nesting gives solvers access to the high-level structure.

A model in Essence Prime consists of three main components: decision variable declarations, constraint expressions, and an optional objective function. The following sections explain how these components are translated to SMT.

### 3.1 Decision Variables and Their Domains

Essence Prime is a finite-domain language, where every decision variable is associated with a finite domain of discrete values. It supports Booleans and

integers as atomic types, and matrices of arbitrary dimensions of these atomic types. Savile Row expands matrices to lists of atomic types so its output languages do not need to support matrices or arrays. For example, given a matrix $M$ with three rows and columns, it will create 9 declarations $M_{1,1}$, $M_{1,2}$, ... $M_{3,3}$ of its declared type.

When targeting SMT, decision variable declarations are translated to their equivalent variable declarations in SMT-LIB together with unary constraints to define the bounds of the domain. For example, when using the LIA encoding, a `find x : int(1..10)` declaration would be translated as the SMT integer variable `x` and the constraint $1 \leq x \leq 10$. When using the QF_BV theory, numerical variables are represented using fixed-size signed bit-vectors (i.e. binary numbers in two's-complement). We use as few bits as we can, governed by the largest domain value in the union of all decision variables. Savile Row performs domain shaving by enforcing a strong level of consistency (Singleton Arc Consistency on the bounds of the variables) via Minion. This step helps in removing any unnecessarily large values in the domains and reducing the number of variables required after bit-blasting [15].

### 3.2 Constraints and the Objective

Constraint expressions are at the heart of a constraint model. We translate them to SMT-LIB using specialised implementations per expression type and by falling back to the standard SAT encodings available in Savile Row where necessary. Boolean expressions ($\wedge$, $\vee$, $\rightarrow$, ...) and relational operators over integers ($<$, $\leq$, $=$, ...) are translated to use the corresponding operators in SMT-LIB. Arithmetic operators ($+$, $-$, /, mod, abs, ...) are similarly translated depending on the theory we use. Quantified expressions, such as the universally and existentially quantified expression or quantified sums are unrolled and turned into $\wedge$, $\vee$, and weighted sums. AllDifferent and Global Cardinality (GCC) are decomposed: for each value (or each constrained value in GCC) a linear constraint is used to restrict the number of occurrences of the value. Table constraints use the Bacchus encoding [3] and short tables are encoded similarly [1].

SMT solvers are typically implemented by augmenting an existing SAT solver with implementations of theories. Hence, they allow SAT clauses in the input. We implement our SMT encodings when they are specifically supported by a backend theory and fall back to the existing SAT encodings in Savile Row for the rest of the cases.

When targeting the QF_BV logic, we use *bvadd*, *bvor* and *bvneg* as appropriate for arithmetic and logical expressions. These operators are not restricted to linear expressions, they support nested expressions with sums, multiplications, divisions, and modulo operations. Modern QF_BV solvers typically use a technique called bit-blasting (amongst others). Bit-blasting converts the problem to SAT by introducing a SAT variable for each bit in a bit vector. This approach can generate a smaller encoding when compared to the direct or order encodings used by the SAT backend of Savile Row [23], since the direct and order encodings scale linearly in domain size and bit-blasting scales sub-linearly.

The objective function (if present) is represented by a single decision variable and dichotomic search is applied to find the optimal solution. However, when Z3 is the target solver we use its built-in optimisation (regardless of the theory). In our experiments we use Z3 for QF_NIA and other solvers for the other three theories, so solver built-in optimisation is only used with QF_NIA.

### 3.3 Flat vs Nested Encodings

Essence Prime (like MiniZinc, OPL and other comparable constraint modelling languages) allows stating *nested* constraints such as allDiff($[x+1, y+2, z+3]$) directly. This constraint is considered nested because each element of the list inside the allDiff constraint is an expression and not a decision variable.

Some of the backend solvers for Savile Row do not support nested constraints, and therefore constraints like these have to be *flattened*. In this case, the flattening process would create auxiliary variables for each sub-expression ($aux_1$, $aux_2$, $aux_3$), post additional constraints $aux_1 = x + 1$, $aux_2 = y + 2$, $aux_3 = z + 3$, and replace the constraint with allDiff($[aux_1, aux_2, aux_3]$). This transformation will introduce additional variables and constraints.

Since SMT-LIB allows nested expressions we have an opportunity to evaluate the effect of using Savile Row's standard flattening process vs maintaining the nested expressions and letting the SMT solver flatten them if needed. A nested encoding is likely to be smaller in size and may allow the SMT solver to do a better job if and when it chooses to flatten. We evaluate flat and nested encodings for each of the four theories in Section 4.

## 4 Empirical Evaluation

In this section we compare our set of encodings (Savile Row-SMT, or *SR-SMT*) with the state of the art on a wide range of problems, showing the advantages of our approach. We compare SR-SMT to the SAT encoding implemented in Savile Row [21] using the SAT solver CaDiCaL 1.3.0, and also to the well-established learning constraint solver Chuffed 0.10.3. In each experiment we use the same large set of problem instances and the same basic settings of Savile Row. We have not been able to compare our approach to fzn2smt [7] or FznTini [14], as they have become obsolete due to changes in the FlatZinc language. We compare our approach to FZN2OMT [9], described in Section 2.

SR-SMT has eight configurations and FZN2OMT has four, which presents a challenge when attempting to compare the two systems. We resolve this by using the *virtual best solver* (VBS) approach for each system; i.e. for each problem instance the best configuration is selected. We also look into which configurations are contributing most to the VBS for each system.

### 4.1 Experimental Setup

As SR-SMT outputs standard SMT-LIB2 files, we are able to target many solvers easily. For the QF_LIA and QF_IDL encodings we use Yices 2.6.2 [11], for the

QF_BV encoding we target Boolector 3.2.1 [19] and finally for QF_NIA we use Z3 4.8.8 [16]. Experience of using SMT solvers for planning and scheduling problems (expressed in QF_LIA or QF_IDL) led us to use Yices for those theories; Boolector is known to perform well on QF_BV; and Z3 was chosen for QF_NIA because of its advanced preprocessing and heuristics (for example, as noted below, Z3 converts the QF_NIA Discrete Tomography problem into QF_BV). Regarding optimisation, we handle it by means of a binary search on the domain of the variable to optimise. The only exception is when the Z3 solver is used, as it has native support for optimisation.

The set of problem classes used is an expanded version of the one used in [21], with 63 problem classes and 757 instances in total[3]. Both satisfaction and optimization problems are included, and optimization problems are reported as solved when optimality of the last solution has been proven.

We used a time limit of 1 hour total time (i.e. Savile Row time plus solver time). We use PAR2 to summarise the performance of each configuration. PAR2 is the mean of total time, where the instances that timed out are assumed to have taken two times the time limit (i.e. 2 hours). Configurations with a lower PAR2 score are considered to be better.

The default set of optimisations in Savile Row are used for all solvers and encodings: domain filtering, variable unification, aggregation and active CSE [21]. A standard FlatZinc backend was added to Savile Row to be used for experiments with FZN2OMT. Decompositions of global constraints closely follow those in the MiniZinc `std` library. For example, allDifferent is decomposed into a clique of pairwise not-equal constraints, and global cardinality into one sum for each constrained value. The existing Chuffed FlatZinc backend is very similar but does not decompose allDifferent or lexicographic ordering constraints.

### 4.2 Comparison to FZN2OMT

Figure 1 gives an overview of the results comparing the SR-SMT virtual best solver (SR-SMT-VBS) to the FZN2OMT virtual best solver (FZN2OMT-VBS). For SR-SMT-VBS, 685 instances were solved and for FZN2OMT-VBS 643 were solved. In Table 1 we report the number of instances solved by the two virtual best solvers as well as each configuration of both systems. The results show a clear advantage for SR-SMT-VBS on the bulk of the instances. However, there are confounding factors. First, SR-SMT-VBS is constructed from 8 configurations rather than 4, potentially giving it an advantage. Second, different SMT solvers are used. For example, Boolector was used to solve QF_BV encodings in SR-SMT-VBS, whereas Z3 and OptiMathSAT were used in FZN2OMT.

Many problem classes contain the allDifferent constraint. For these, SR-SMT uses a decomposition where each value is constrained to have at most one occurrence using a linear constraint (in common with Savile Row's SAT backend), whereas the standard FlatZinc backend used with FZN2OMT decomposes to pairwise not-equal constraints. We compare the two decompositions below.

[3]Experiment scripts, model and parameter files and raw results can be found at: https://github.com/stacs-cp/CP2020-SRSMT [10]

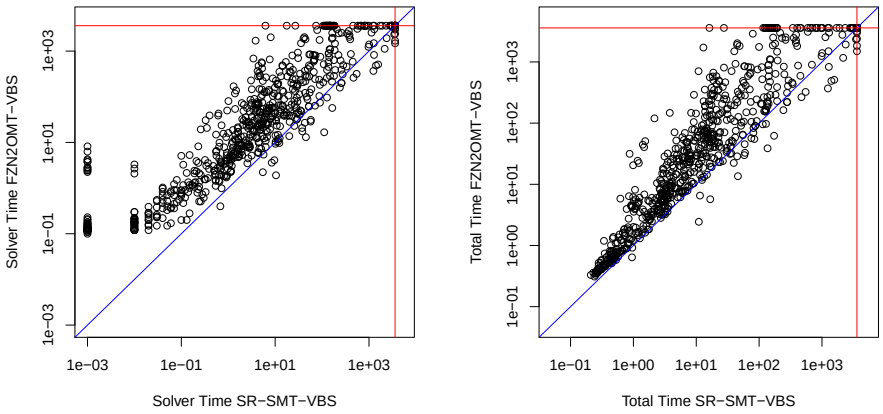


Fig. 1: SR-SMT-VBS vs FZN2OMT-VBS, solver time (left) and total time (right).

| Encoding | Instances solved | Mean PAR2 |
|---|---|---|
| **SR-SMT-VBS** | **685** | **803** |
| SR-SMT-Nested-VBS | 682 | 840 |
| SR-SMT-Flat-VBS | 682 | 839 |
| FZN2OMT-VBS | 643 | 1283 |
| SR-SMT-BV-Nested | 663 | 1092 |
| SR-SMT-BV-Flat | 661 | 1106 |
| SR-SMT-NIA-Nested | 499 | 2620 |
| SR-SMT-NIA-Flat | 511 | 2522 |
| SR-SMT-LIA-Nested | 590 | 1695 |
| SR-SMT-LIA-Flat | 586 | 1729 |
| SR-SMT-IDL-Nested | 592 | 1714 |
| SR-SMT-IDL-Flat | 591 | 1716 |
| SR-SMT-BV-Nested-Z3 | 657 | 1138 |
| SR-SMT-BV-Nested-Z3-PA | 643 | 1228 |
| FZN2OMT-LIA-Z3 | 508 | 2638 |
| FZN2OMT-LIA-OptiMathSAT | 517 | 2609 |
| FZN2OMT-BV-Z3 | 587 | 1921 |
| FZN2OMT-BV-OptiMathSAT | 533 | 2454 |
| SAT | 671 | 931 |
| Chuffed | 637 | 1248 |

Table 1: Number of instances solved and PAR2 for each configuration. Bold indicates the overall best configuration (by instances solved), red indicates the best configuration of one system (SR-SMT or FZN2OMT).

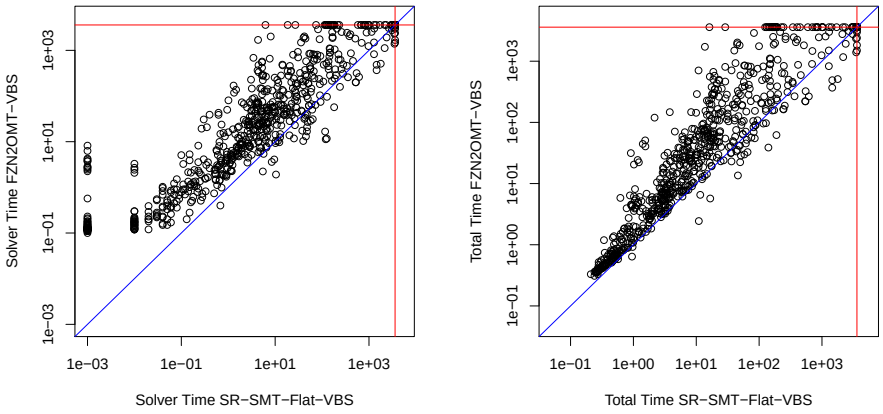


Fig. 2: SR-SMT-Flat-VBS (i.e. virtual best solver of the 4 Flat configurations) vs FZN2OMT-VBS, solver time (left) and total time (right).

To provide a fairer comparison between two portfolios of the same size, we constructed two further virtual best solvers for SR-SMT, one using the four Flat configurations (SR-SMT-Flat-VBS) and the other using the four Nested configurations (SR-SMT-Nested-VBS). As Table 1 shows, the Flat configuration solves more instances for one theory (QF_NIA) and is quite close to Nested in performance for the other theories. On the other hand, Nested solves more instances with the most promising theory (QF_BV) and also QF_LIA and QF_IDL. It turns out that SR-SMT-Flat-VBS is very slightly stronger (Table 1). Figure 2 compares SR-SMT-Flat-VBS to FZN2OMT-VBS, and Table 1 contains the number of instances solved and PAR2 score for both. Even with a smaller portfolio it is clear that SR-SMT is performing better than FZN2OMT on these benchmarks.

Also, there is the issue that the solvers do not match for any theory. SR-SMT uses Yices with QF_LIA and Boolector with QF_BV, whereas FZN2OMT uses Z3 and OptiMathSAT. To be able to compare just the encodings, we ran SR-SMT-BV-Nested (the strongest configuration of SR-SMT measured by instances solved and PAR2) with Z3 instead of Boolector, creating a configuration called SR-SMT-BV-Nested-Z3. Table 1 shows that SR-SMT-BV-Nested-Z3 is somewhat weaker than SR-SMT-BV-Nested on these benchmarks, solving 6 fewer instances within 1 hour and having a higher PAR2 score. Figure 3 compares SR-SMT-BV-Nested-Z3 to FZN2OMT-BV-Z3. The results are mixed, with some instances solving much faster with FZN2OMT-BV-Z3, and 15 solved only by FZN2OMT-BV-Z3. However, the overall trend is that SR-SMT-BV-Nested-Z3 is stronger, it solves 70 more instances and has a lower PAR2 score.

Finally, there is the issue that the decompositions of AllDifferent do not match. We created another configuration SR-SMT-BV-Nested-Z3-PA, which is SR-SMT-BV-Nested-Z3 with the pairwise decomposition of AllDifferent (matching FZN2OMT). As Table 1 shows, the new configuration is weaker than SR-

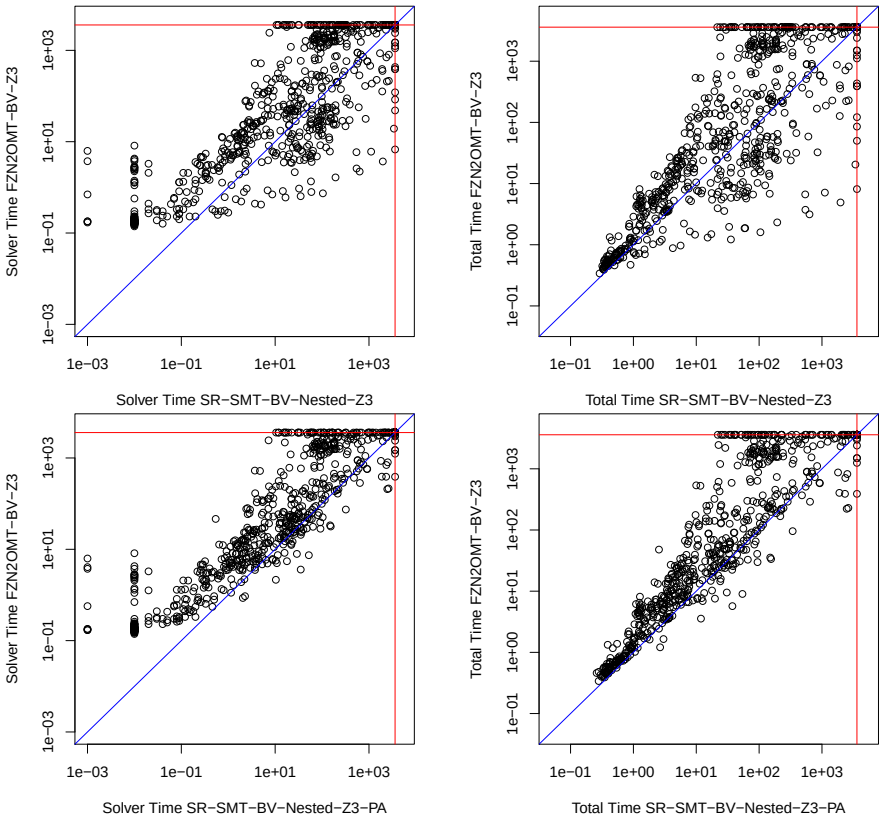


Fig. 3: SR-SMT-BV-Nested-Z3 (i.e. SR-SMT-BV-Nested with Z3 instead of Boolector) vs FZN2OMT-BV-Z3, solver time (upper left) and total time (upper right). The lower plots are the same but with the pairwise AllDifferent decomposition (PA).

SMT-BV-Nested-Z3, solving 14 fewer instances in total. Figure 3 compares SR-SMT-BV-Nested-Z3-PA to FZN2OMT-BV-Z3. The two are quite strongly correlated but SR-SMT is stronger, solving 56 more instances in total.

### 4.3 Analysis of SR-SMT

In this section we look at which configurations of SR-SMT are most effective, and how each configuration contributes to the virtual best solver. First we compare Nested to Flat configurations, then we compare the four theories. Also, Table 2 shows how each configuration contributed to the virtual best solver SR-SMT-VBS, and how many additional instances were solved within multiples of 2 and 5 of the VBS time. This gives an overview of how strong the configurations are relative to the VBS.

| Configuration | Multiple of SR-SMT-VBS Total Time | | | |
|---|---|---|---|---|
| | 1 | 2 | 5 | Any |
| SR-SMT-BV-Nested | 76 | 350 | 465 | **663** |
| SR-SMT-BV-Flat | 62 | 319 | 473 | 661 |
| SR-SMT-NIA-Nested | 25 | 244 | 351 | 499 |
| SR-SMT-NIA-Flat | 60 | 294 | 365 | 511 |
| SR-SMT-LIA-Nested | 146 | **434** | 530 | 590 |
| SR-SMT-LIA-Flat | 86 | 433 | **534** | 586 |
| SR-SMT-IDL-Nested | 120 | 401 | 488 | 592 |
| SR-SMT-IDL-Flat | **149** | 403 | 486 | 591 |

| Configuration | Multiple of FZN2OMT-VBS Total Time | | | |
|---|---|---|---|---|
| | 1 | 2 | 5 | Any |
| FZN2OMT-LIA-Z3 | 148 | 318 | 362 | 508 |
| FZN2OMT-LIA-OptiMathSAT | 114 | 222 | 308 | 517 |
| FZN2OMT-BV-Z3 | **239** | **439** | **498** | **587** |
| FZN2OMT-BV-OptiMathSAT | 150 | 282 | 378 | 533 |

Table 2: Number of instances solved by each configuration within multiples of SR-SMT-VBS or FZN2OMT-VBS total time. An instance is counted for configuration X and multiple $f$ if total time for X is within $f\times$ total time for the VBS. The highest value in each column is highlighted in bold.

**Comparing Nested to Flat Translation** Figure 4 compares Nested to Flat configurations for each theory. With the theory of bit-vectors the two configurations are remarkably similar. Total number of instances solved and PAR2 score from Table 1 suggest that Nested is slightly better, and Nested is also selected more often in SR-SMT-VBS (Table 2).

With the QF_NIA theory, Flat translation performs better overall, solving 12 more instances and with a lower PAR2 score. Some problem classes were solved much better with Flat, such as BIBD (in green). For some instances of EFPA (highlighted in blue), the two encodings are similar, and for others Flat is significantly more efficient. BIBD exhibits increasing gains for Flat as the instances become more difficult. However, Nested is more efficient for Langford's Problem, solving several instances that Flat cannot.

For the QF_LIA theory, we found that almost all problem classes were very similar. There were three exceptions: Killer Sudoku (where the Flat encoding is somewhat better), Car Sequencing, and Peg Solitaire (where the instances are scattered and neither Nested nor Flat seem to have an advantage).

Finally, with QF_IDL for almost all problem classes the Nested and Flat encodings were very similar in performance. There are no problem classes where one encoding consistently outperforms the other by a substantial margin.

**Comparing Theories** The four theories have quite different characteristics and the ideal choice of theory might vary by problem class. To compare the

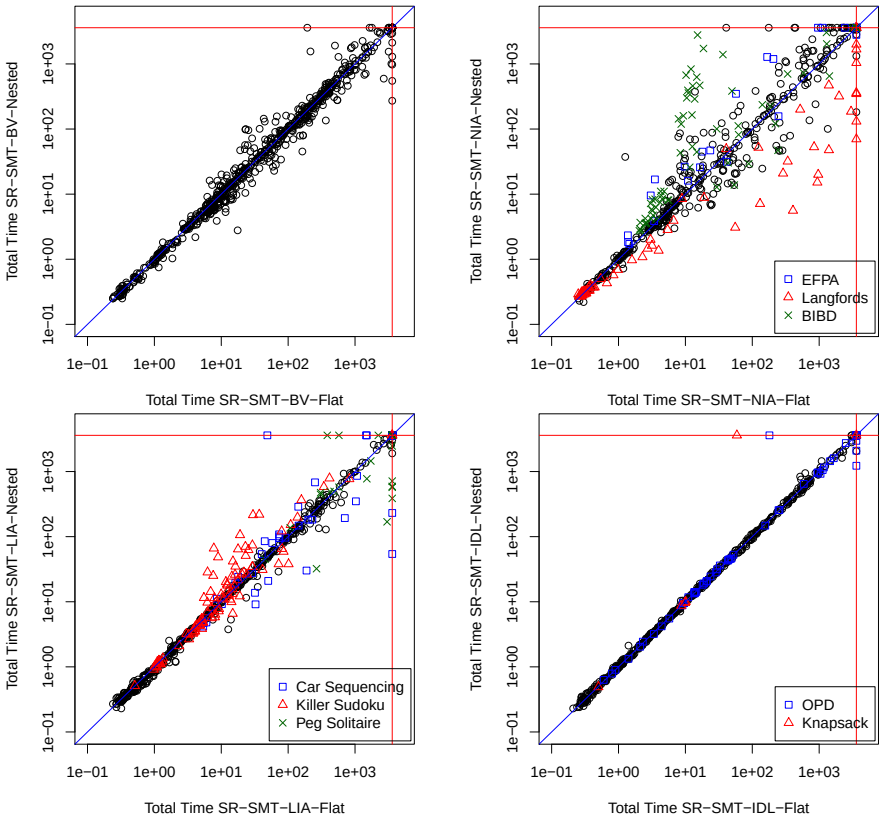


Fig. 4: Nested vs Flat translation for each theory, total time.

four theories, for each one we take the better configuration (of Nested or Flat), so we compare SR-SMT-BV-Nested, SR-SMT-NIA-Flat, SR-SMT-LIA-Nested, and SR-SMT-IDL-Nested. BV is the strongest theory in terms of instances solved within 1 hour and PAR2 score, therefore we use BV as the gold standard and compare the other theories to it.

Figure 5 (top left) plots NIA-Flat against BV-Nested. Many instances time out with NIA-Flat, for example the majority of Aces-Up (in red), and all instances of the block party metacube problem (BPMP, in blue). Langford's Problem is an example where the two theories are correlated, but BV is more efficient on the bulk of the instances. In contrast, Discrete Tomography is solved more efficiently by NIA-Flat. In this case, Z3 internal heuristics decide to convert the QF_NIA formula into a QF_BV formula before solving, and this is more efficient than Boolector directly applied to the BV-Nested formula. Also, both NIA encodings contribute a relatively small number of instances to the VBS (Table 2).

LIA-Nested is plotted against BV-Nested in Figure 5 (top right). Langford's Problem and Discrete Tomography are examples where LIA-Nested performs substantially better than BV-Nested. The constraints in these problems are well suited to the QF_LIA theory (in particular Discrete Tomography, where all constraints are linear). Car Sequencing is split, some instances are solved more quickly by LIA-Nested while others time out. All instances of JPEncoding are solved more efficiently by LIA-Nested. Overall BV-Nested has a substantial edge: it is able to solve 73 more instances and its PAR2 score is much lower.

Finally, we compare IDL-Nested to BV-Nested in Figure 5 (lower). MRCPSP (the multi-mode resource-constrained project scheduling problem) has precedence constraints that are naturally expressed in IDL. The most difficult instances of MRCPSP are solved more efficiently by IDL-Nested. Langford's Problem is also solved more efficiently by IDL-Nested. OPD is mixed but shows large speed-ups for BV for some of the most difficult instances. Many instances time out for IDL-Nested and are solved with BV-Nested, e.g. all instances of the JPEncoding problem. In total, 71 more instances are solved by BV-Nested.

In summary, we have seen several cases where NIA, LIA, or IDL performs well on a problem class and in these cases the problem class has constraints that are naturally expressed in the theory. For example, Discrete Tomography (with linear constraints) is solved very well by LIA-Nested (and also by NIA-Flat, where the solver converts it into a QF_BV formula). In contrast, BV seems the most robust. It performs well on a wide range of problem classes and in each comparison solves many instances that the other configuration did not.

### 4.4 Analysis of FZN2OMT

Table 2 gives an overview of how each configuration of FZN2OMT contributes to the virtual best solver, FZN2OMT-VBS. The BV encoding with Z3 is clearly the strongest combination on these benchmarks, contributing the most instances to the VBS and also solving the most within 1 hour. Z3 was more effective than OptiMathSAT on the BV formulas, however the picture is not so simple with LIA. OptiMathSAT seems to be stronger overall on LIA formulas, solving more instances within 1 hour, however Z3 contributes more instances to the VBS.

### 4.5 Comparison to SAT

As described in Section 3, all SR-SMT encodings are built on the default SAT encoding of Savile Row, so comparison between SR-SMT and SAT is particularly relevant. We use CaDiCaL as the SAT solver and use the default options for the SAT backend. Figure 6 (top left) plots SAT against the strongest SMT configuration, SR-SMT-BV-Nested. The two are quite similar in overall performance, with similar PAR2 scores. SAT solves 8 more of the benchmark instances than SR-SMT-BV-Nested (Table 1). Some problem classes are solved more efficiently by SAT, such as Discrete Tomography, Langford, and MRCPSP. In contrast, SR-SMT-BV-Nested is able to solve 6 out of 10 instances of JPEncoding whereas SAT solves none. Car Sequencing is mixed but the majority of

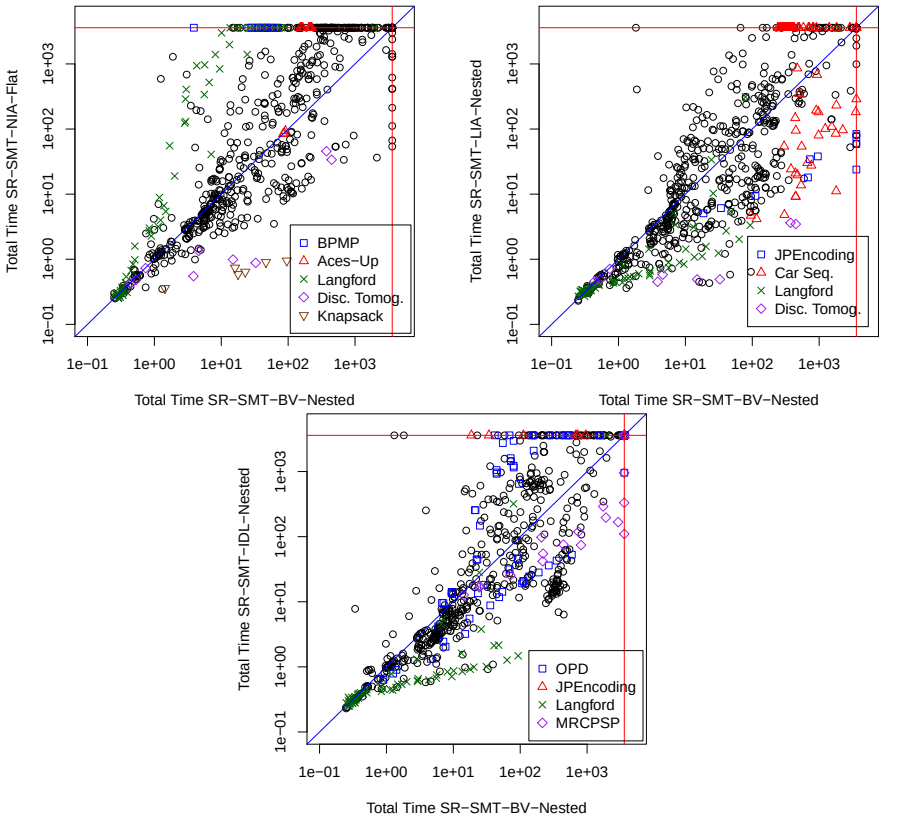


Fig. 5: Comparison of the four theories using the better configuration for each one (of Nested or Flat).

instances are solved more efficiently by SAT. 644 instances are solved by both SAT and BV-Nested, and of those the majority (510) are solved faster by SAT.

Figure 6 (top right) plots SAT against SR-SMT-VBS. The VBS solves a further 22 instances compared to BV-Nested (14 more than SAT), and improves on various problem classes including Langford, Discrete Tomography, MRCPSP, and Car Sequencing. For the 657 instances that are solved by both the VBS and SAT, the majority (337) are solved more efficiently by SAT, however the VBS has a better PAR2 score (by over 100 seconds). The VBS is of course a theoretical solver, but these results show the value of selecting an appropriate theory for a given instance. As part of our future work we intend to investigate algorithm selection methods to construct a portfolio of the 8 SR-SMT configurations.

We found that the QF_IDL encodings were the largest, with a median clause ratio of 104.5% for both IDL-Nested and IDL-Flat compared to SAT. The QF_LIA encodings were more compact, with median clause ratios of 15.7% for

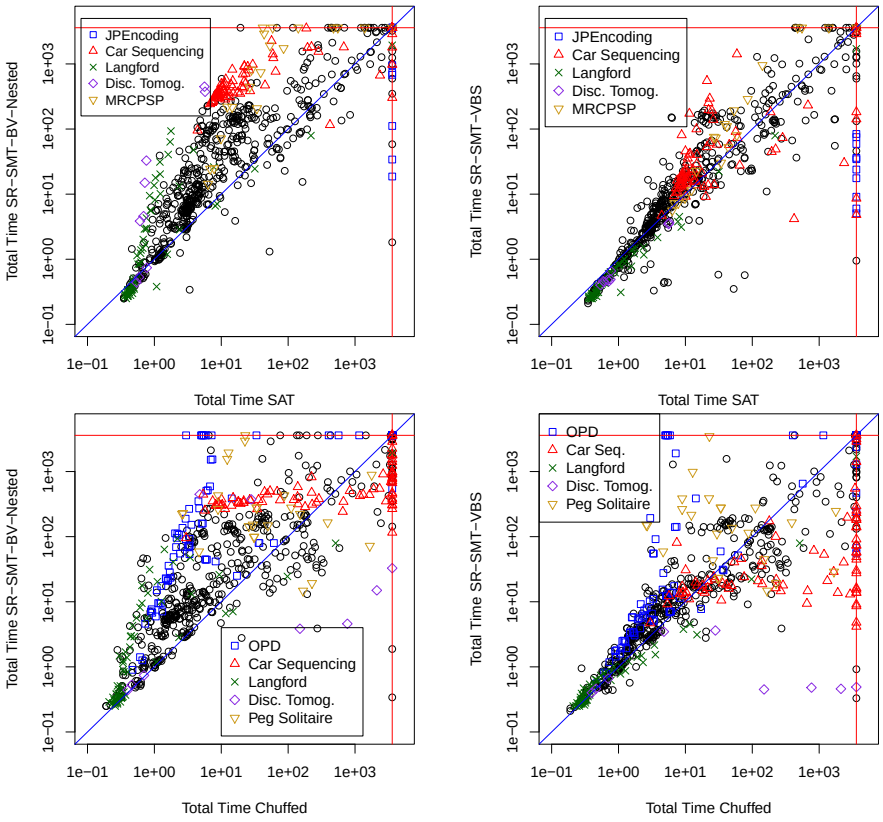


Fig. 6: Comparison of SAT to SR-SMT-BV-Nested (top left) and SR-SMT-VBS (top right). Comparison of Chuffed to SR-SMT-BV-Nested (lower left) and SR-SMT-VBS (lower right).

LIA-Flat and 5.2% for LIA-Nested. The more expressive theories of QF_BV and QF_NIA allowed for the smallest encodings, with median clause ratios of 7.5% for BV-Flat, 2.8% for BV-Nested, 7.3% for NIA-Flat, and 2.8% for NIA-Nested. It is notable that one of the two smallest encodings (BV-Nested) has the highest performance. Also, Nested encodings are smaller than Flat with the exception of IDL where they are very similar in size.

### 4.6 Comparison to Chuffed

Chuffed is a well-established learning CP solver that uses a similar learning scheme as CDCL SAT and SMT solvers. The Chuffed backend of Savile Row produces FlatZinc that is specific to Chuffed, i.e. it uses the global constraints implemented in Chuffed. We use Chuffed's free search option and also provide a

search annotation given in the model or a default search annotation (variable declaration order). Figure 6 (lower left) plots Chuffed against SR-SMT-BV-Nested.

Results are mixed, with several problem classes solved much more efficiently by Chuffed (such as OPD and Langford for a large majority of their instances). Others are split, several Car Sequencing instances are solved substantially faster by Chuffed but others time out for Chuffed and are only solved by BV-Nested. The BV-Nested encoding performs well on Discrete Tomography (with linear constraints) and some instances of Peg Solitaire. In terms of instances solved, SR-SMT-BV-Nested performs better than Chuffed, solving 26 more instances within 1 hour. However Chuffed is more efficient for 510 of the 615 instances that they both solve. The timeouts cause Chuffed to have a relatively high PAR2 score of 1248, compared to 1092 for BV-Nested.

Figure 6 (lower right) plots Chuffed against SR-SMT-VBS. The VBS solves 48 more instances than Chuffed, but Chuffed remains more efficient for the majority of instances (380 out of 627) that they both solve. Comparing the two plots, the VBS is more efficient than BV-Nested on many instances (including large numbers of OPD, Langford and Car Sequencing instances) and has narrowed the gap between SMT and Chuffed on instances where Chuffed is faster.

## 5 Conclusions and Future Work

We have presented SR-SMT, an SMT backend for Savile Row that is able to target four SMT theories, each with two levels of flattening. We have performed an extensive set of experiments comparing our encodings to each other and also to FZN2OMT, SAT, and Chuffed. We found that SR-SMT with the QF_BV theory is a very robust approach: it solves more instances than other theories, Chuffed, and FZN2OMT within the time limit. However, SR-SMT with QF_BV is not always the fastest approach, suggesting that it would be a useful component of a portfolio of solvers.

While we found QF_BV to be particularly robust, other theories (QF_NIA, QF_LIA, and QF_IDL) performed strongly when problem constraints are naturally expressed in the theory, for example the LIA theory applied to the Discrete Tomography problem (which is linear). Consequently, the virtual best solver composed of all 8 SR-SMT configurations is significantly stronger than any one configuration. As part of future work, we will look at algorithm selection methods (such as the SUNNY algorithm used in the SUNNY-CP portfolio solver [2]) to construct a portfolio of SMT encodings, and investigate whether such a portfolio has similar performance to the VBS.

In summary, encoding constraint problems to SMT via SR-SMT is a successful approach, solving more instances of our benchmark set than the mature learning CP solver Chuffed and the existing FZN2OMT system.

**Acknowledgements** We thank Marc Roig Vilamala who worked on an early version of the SMT backend of Savile Row. This work is supported by EPSRC grant EP/P015638/1.

## References


1. Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Exploiting short supports for improved encoding of arbitrary constraints into SAT. In: Principles and Practice of Constraint Programming - 22nd International Conference, CP, Toulouse, France. pp. 3–12 (2016)
2. Amadini, R., Gabbrielli, M., Mauro, J.: SUNNY-CP and the MiniZinc challenge. Theory and Practice of Logic Programming **18**(1), 81–96 (2018). https://doi.org/10.1017/S1471068417000205
3. Bacchus, F.: GAC via unit propagation. In: Principles and Practice of Constraint Programming, 13th International Conference, CP Providence, RI, USA. pp. 133–147 (2007)
4. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Tech. rep., Department of Computer Science, The University of Iowa (2017), available at `www.SMT-LIB.org`
5. Belov, G., Stuckey, P.J., Tack, G., Wallace, M.: Improved linearization of constraint programming models. In: International Conference on Principles and Practice of Constraint Programming. pp. 49–65. Springer (2016)
6. Bofill, M., Palahı, M., Suy, J., Villaret, M.: SIMPLY: a Compiler from a CSP Modeling Language to the SMT-LIB Format. In: Proceedings of the 8th international workshop on constraint modelling and reformulation. pp. 30–44 (2009)
7. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving constraint satisfaction problems with SAT modulo theories. Constraints **17**(3), 273–303 (2012). https://doi.org/10.1007/s10601-012-9123-1
8. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving intensional weighted CSPs by incremental optimization with BDDs. In: International Conference on Principles and Practice of Constraint Programming. pp. 207–223. Springer (2014)
9. Contaldo, F., Trentin, P., Sebastiani, R.: From minizinc to optimization modulo theories, and back (extended version). CoRR **abs/1912.01476** (2019), http://arxiv.org/abs/1912.01476
10. Davidson, E., Akgün, Ö., Espasa, J., Nightingale, P.: Experiments for CP2020 SR-SMT paper. (2020), https://doi.org/10.5281/zenodo.3953600
11. Dutertre, B.: Yices 2.2. In: Computer Aided Verification - 26th International Conference, CAV Vienna, Austria. pp. 737–744 (2014)
12. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: Computer Aided Verification, 18th International Conference, CAV Seattle, WA, USA. pp. 81–94 (2006)
13. Hadarean, L., Bansal, K., Jovanovic, D., Barrett, C.W., Tinelli, C.: A Tale of Two Solvers: Eager and Lazy Approaches to Bit-Vectors. In: Computer Aided Verification - 26th International Conference, CAV Vienna, Austria. pp. 680–695 (2014)
14. Huang, J.: Universal Booleanization of Constraint Models. In: Principles and Practice of Constraint Programming, 14th International Conference, CP, Sydney, Australia. pp. 144–158 (2008)
15. Kroening, D., Strichman, O.: Decision Procedures - An Algorithmic Point of View, Second Edition. Texts in Theoretical Computer Science. An EATCS Series, Springer (2016). https://doi.org/10.1007/978-3-662-50497-0
16. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS, Budapest, Hungary. pp. 337–340 (2008)

17. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a Standard CP Modelling Language. In: Bessiere, C. (ed.) Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, Providence, RI, USA. vol. 4741, pp. 529–543 (2007). https://doi.org/10.1007/978-3-540-74970-7_38
18. Niebert, P., Mahfoudh, M., Asarin, E., Bozga, M., Maler, O., Jain, N.: Verification of timed automata via satisfiability checking. In: Formal Techniques in Real-Time and Fault-Tolerant Systems, 7th International Symposium, FTRTFT Oldenburg, Germany. pp. 225–244 (2002)
19. Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2 , BtorMC and Boolector 3.0. In: Computer Aided Verification - 30th International Conference, CAV Oxford, UK. pp. 587–595 (2018)
20. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL($T$). Journal of the ACM **53**(6), 937–977 (2006). https://doi.org/10.1145/1217856.1217859
21. Nightingale, P., Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I., Spracklen, P.: Automatically improving constraint models in Savile Row. Artificial Intelligence **251**, 35–61 (2017). https://doi.org/10.1016/j.artint.2017.07.001
22. Nightingale, P., Rendl, A.: Essence' description. Computing Research Repository (CoRR) **abs/1601.02865** (2016), http://arxiv.org/abs/1601.02865
23. Nightingale, P., Spracklen, P., Miguel, I.: Automatically improving SAT encoding of constraint problems through common subexpression elimination in Savile Row. In: International Conference on Principles and Practice of Constraint Programming. pp. 330–340. Springer (2015)
24. Sebastiani, R., Trentin, P.: OptiMathSAT: A tool for optimization modulo theories. In: Computer Aided Verification - 27th International Conference, CAV, San Francisco, CA, USA. pp. 447–454 (2015)
25. Van Hentenryck, P.: The OPL optimization programming language. MIT press (1999)