


Evaluating the Impact of Encoding on SAT Based Constraint Solving in Exchequer

Aadil Sattar ✉

Department of Computer Science, University of Reading, United Kingdom

Martin Mariusz Lester ✉ 

Department of Computer Science, University of Reading, United Kingdom

Abstract

Exchequer is a constraint solver that translates XCSP3 problems into C programs, which are then analysed by a bounded model checker. The model checker encodes the C program as a SAT instance, and if the problem is satisfiable, produces a counterexample trace from which a solution can be extracted. The way constraints are encoded in C, particularly the choice between logical (&&, ||) and bitwise (&, |) operators, can significantly affect the performance of the SAT solver. In this work, we benchmark Exchequer using different encoding strategies to determine which approaches yield the most efficient solving.

2012 ACM Subject Classification Software and its engineering → Software verification; Software and its engineering → Constraint and logic languages

Keywords and phrases XCSP3, SAT Solving, Solver Benchmarking, Solver Performance, Bounded Model Checking, Encoding Strategies

1 Motivation

Exchequer is a constraint solver designed to address combinatorial problems expressed in the XCSP3-Core format, a widely used XML-based representation that preserves much of the structure and semantics of constraint satisfaction and optimisation problems [4]. Unlike many traditional solvers that process XCSP3 instances directly, Exchequer employs a novel approach: it translates XCSP3 problems into C programmes, which are then analysed using tools from the software verification community, such as bounded model checkers like CBMC [3].

Each variable in the XCSP3 problem is mapped to a corresponding C variable, and constraints are encoded as assumptions or assertions in the C code. The generated C programme is constructed so that an assertion violation occurs if and only if the variable assignments satisfy all the original constraints. The programme is then passed to a bounded model checker (CBMC), which encodes the C programme as a SAT instance. If the problem is satisfiable, the model checker produces a counterexample trace, from which Exchequer extracts the solution [5].

This methodology leverages the strengths of state-of-the-art verification tools, allowing Exchequer to benefit from advanced SAT-based reasoning and symbolic execution. It also opens the door to evaluating a broad range of verification techniques, developed and benchmarked in venues such as the annual Software Verification Competition (SV-COMP) [1], on combinatorial constraint problems.

A critical insight is that the efficiency of the SAT solver, and thus the overall performance of Exchequer, is not determined solely by the original XCSP3 problem but also by the specifics of the translation to C. In particular, the choice of operators for encoding constraints, such as logical AND/OR (&&, ||) versus bitwise AND/OR (&, |), can significantly affect both the size and complexity of the resulting SAT instance and the speed with which the solver finds a solution. Logical operators in C provide short-circuit evaluation, which can reduce unnecessary computation and lead to more compact SAT encodings, while bitwise operators

always evaluate both operands, potentially increasing the computational burden and the number of SAT clauses generated.

However, there are plausible reasons to consider both logical and bitwise encodings, making the choice non-trivial. Logical operators guarantee results of exactly 0 or 1, and, if the model checker is sufficiently sophisticated, it may optimise chains of logical operations to single-bit computations, potentially yielding efficient SAT encodings. Conversely, the short-circuiting behaviour of logical operators introduces additional control flow paths, which can cause path explosion in tools that enumerate execution paths, such as symbolic execution engines. In these cases, bitwise operators, which do not short-circuit and thus avoid extra branches, may be preferable. Recent work by Verma and Yap [6] demonstrates that bitwise encodings can mitigate path explosion and sometimes lead to better solver performance. Thus, it is not a foregone conclusion which encoding is best for Exchequer; systematic benchmarking is necessary to empirically determine the most effective strategy.

Example

■ **Listing 1** A simple XCSP3 example: find integers $1 \leq x \leq 10$ and $1 \leq y \leq 100$ with $y = x^2$.

```
<instance format="XCSP3" type="CSP">
<variables>
<var id="x"> 1..10 </var>
<var id="y"> 1..100 </var>
</variables>
<constraints>
<intension>
eq(y,mul(x,x))
</intension>
</constraints>
</instance>
```

■ **Listing 2** Exchequer's C translation using logical operators (`&&`, `||`).

```
#include <stdint.h>
uint32_t x = __VERIFIER_nondet_u32();
uint32_t y = __VERIFIER_nondet_u32();

__VERIFIER_assume((x >= 1) && (x <= 10));
__VERIFIER_assume((y >= 1) && (y <= 100));
__VERIFIER_assume(y == x * x);

__VERIFIER_error();
```

■ **Listing 3** Alternative C translation using bitwise operators (`&`, `|`).

```
#include <stdint.h>
uint32_t x = __VERIFIER_nondet_u32();
uint32_t y = __VERIFIER_nondet_u32();

__VERIFIER_assume((x >= 1) & (x <= 10));
__VERIFIER_assume((y >= 1) & (y <= 100));
__VERIFIER_assume(y == x * x);

__VERIFIER_error();
```

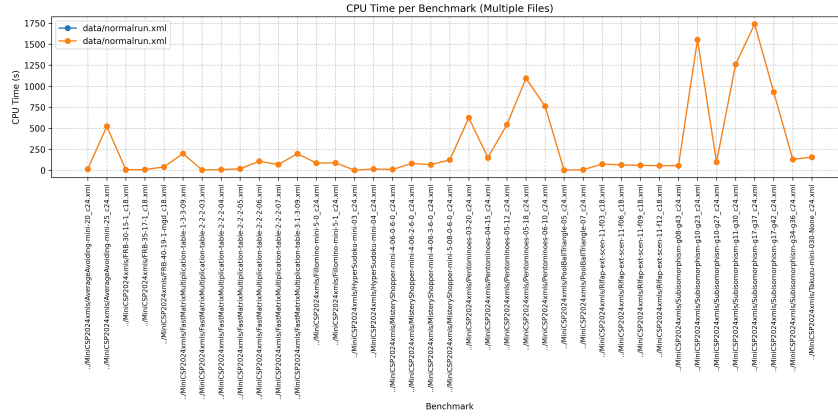
The choice between logical ($\&\&$, \parallel) and bitwise ($\&$, \mid) operators impacts both correctness and performance:

- Logical operators short-circuit evaluation (e.g., stop at the first false condition in $\&\&$), which can reduce SAT instance size but introduces additional control-flow branches. This may cause *path explosion* in symbolic execution tools [6].
- Bitwise operators evaluate all operands and avoid branching, simplifying control flow but potentially generating larger SAT encodings.

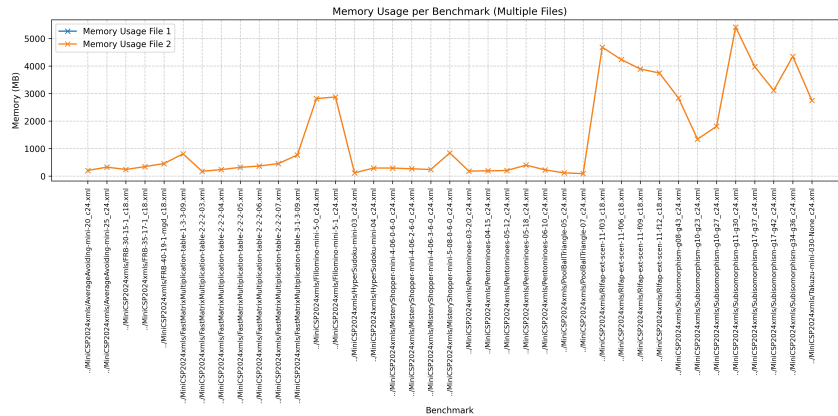
Systematic benchmarking in Exchequer is required to determine which encoding yields better performance for specific problem classes.

2 Results

We benchmarked Exchequer using a suite of representative XCSP3 problems, comparing the impact of logical ($\&\&$, \parallel) versus bitwise ($\&$, \mid) operator encodings on solver performance. The results are summarised in Figures 1 and 2.



■ **Figure 1** Comparison of CPU time for logical vs. bitwise operator encodings across benchmark problems. Logical operators consistently result in lower CPU time.



■ **Figure 2** Comparison of memory usage for logical vs. bitwise operator encodings. Logical encodings tend to use less memory, especially on larger instances.

As shown in Figure 1, encoding constraints using logical operators leads to a substantial reduction in CPU time compared to bitwise encodings. Similarly, Figure 2 demonstrates that logical encodings typically consume less memory. These results confirm that the choice of encoding strategy has a significant impact on the efficiency of SAT-based solving in Exchequer, with logical operators offering clear advantages in both runtime and memory usage.

References

- 1 D. Beyer. Competition on software verification and witness validation: SV-COMP 2023. In *Proc. TACAS (2)*, LNCS 13994, pages 495–522. Springer, 2023. URL: <https://sv-comp.sosy-lab.org/2022/publications.php>, doi:10.1007/978-3-031-30820-8_29.
- 2 Frédéric Boussemart, Christophe Lecoutre, Gilles Audemard, and Cédric Piette. Xcsp3-core: A format for representing constraint satisfaction/optimization problems. *CoRR*, abs/2009.00514, 2020. URL: <https://arxiv.org/abs/2009.00514>, arXiv:2009.00514.
- 3 Edmund Clarke, Daniel Kroening, and Flavio Lerda. CBMC: A bounded model checker for c programs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, volume 2988 of LNCS, pages 436–450. Springer, 2004. URL: <https://dblp.org/rec/conf/tacas/ClarkeKL04.html>, doi:10.1007/978-3-540-24730-2_29.
- 4 Christophe Lecoutre et al. Xcsp3-core: a format for representing core constraint networks. *arXiv preprint arXiv:2009.00514*, 2020. URL: <https://arxiv.org/abs/2009.00514>.
- 5 Martin Mariusz Lester. Solving xcsp3 constraint problems using tools from software verification. In *ModRef 2022: The 21st workshop on Constraint Modeling and Reformulation*, 2022. URL: https://modref.github.io/papers/ModRef2022_SolvingXCSP3ConstraintProblemsUsingToolsFromSoftwareVerification.pdf.
- 6 Sahil Verma and Roland H. C. Yap. Benchmarking symbolic execution using constraint problems – initial results. *arXiv preprint arXiv:2001.07914*, 2020. Presented at ICTAI 2019. URL: <https://arxiv.org/abs/2001.07914>.