


Solving XCSP3 constraint problems using tools from software verification

Martin Mariusz Lester ✉ 

Department of Computer Science, University of Reading, United Kingdom

Abstract

Since 2012, the Software Verification Competition (SV-COMP) has evaluated the performance of a wide range of software verification tools, using a wide range of automated reasoning techniques and representations. These include SAT/SMT solving, abstract interpretation, Constrained Horn Clause solving, numerical interval analysis and finite state automata.

The XCSP3 constraint format serves as an intermediate language for constraint problems. When trying to solve a particular problem, one often wishes to trial a range of techniques to see which one is initially most promising. Unfortunately, there might not always be an implementation of the desired technique for one's formulation.

We propose to address this problem by encoding XCSP3 problems as C program software verification problems. This grants the ability to trial all techniques implemented in SV-COMP entries. We should not expect the encoding to be efficient, but it may be sufficient to identify the most promising techniques.

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

Keywords and phrases XCSP3, software verification, automatic reformulation

1 Motivation

A popular constraint programming language is MiniZinc [6], which can be compiled down to the lower level format FlatZinc. It is easier for someone wanting to develop a new solver to target FlatZinc rather than MiniZinc. However, some of the structure of a constraint problem is usually lost in the translation, which is disadvantageous for some solver techniques. The XML-based intermediate representation for constraint problems XCSP3 [1] aims to fill the gap between these two extremes, providing a format that is easier than MiniZinc for a solver developer to target, while retaining more of a problem's structure than FlatZinc.

It is often the case that certain families of constraint problems are more amenable to solution using a particular method, so a user wishing to solve a particular problem may want to test several different solvers that implement complementary techniques.

Sometimes, one may wish to test a technique that comes from another research community, but which is not implemented in any constraint solver. The software verification community has developed a wide range of varied techniques for reasoning about execution of programs; these are evaluated in solvers submitted to the yearly Software Verification Competition (SV-COMP) [2]. In order to allow easy testing of these techniques on constraint problems, we propose to translate XCSP3 instances into C program verification problems.

A common form of program verification tool aims to prove that no execution of a program (with any variable values, not just those specified in tests) can lead to an assertion violation. If an assertion violation is possible, it gives a trace of execution of the program, showing how the assertion violation occurs and the values of variables along that trace. We can repurpose this to solve constraint problems by constructing a program in which an assertion violation occurs only when its variables' values satisfy a constraint problem. Then the counterexample trace produced by the verification tool will give the solution to the constraint problem.

2 Example

To demonstrate our idea, we consider a simple example from the XCSP3-core specification [3]:

■ **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** A reformulation in C; functions beginning `__VERIFIER` are standard in SV-COMP.

```
int main() {
  uint32_t x;
  uint32_t y;

  x = __VERIFIER_nondet_u32();
  __VERIFIER_assume(x >= 1 && x <= 10);
  y = __VERIFIER_nondet_u32();
  __VERIFIER_assume(y >= 1 && y <= 100);

  __VERIFIER_assume(y == x*x);

  __VERIFIER_error();
}
```

In our C translation of the problem, the XCSP3 variables `x` and `y` are mapped directly onto C variables with the same names. In order to express that the values of these variables are not fixed, we use `__VERIFIER_nondet_u32()` to assign a value nondeterministically. To express that the variables have a limited range, we use `__VERIFIER_assume()` to tell the verifier to *ignore* paths through the program that *do not* meet the specified condition.

The `intension` constraint is also translated into an `assume` statement, with the content of the constraint converted to C syntax.

Finally, the line `__VERIFIER_error()`, similar to the C idiom `assert(0)`, indicates that it is an error for execution of the program to complete (if all assumptions are true).

3 Status

We have developed a prototype of our transformation tool, `xcsp2c`. In combination with the bounded model-checking program verification tool CBMC [4], which uses a translation to SAT, this yields a solver called *Exchequer*, which we have entered into the XCSP Competition 2022 mini solver tracks. Our hypothesis is that this combination will perform acceptably, but worse than the direct approach used by PicatSAT [7], which also uses a translation to SAT, and ranked highly in the the XCSP Competition 2018 [5] and 2019. We are investigating whether we can improve *Exchequer* by using other verification tools from SV-COMP.

References

- 1 Gilles Audemard, Frédéric Boussemart, Christophe Lecoutre, Cédric Piette, and Olivier Roussel. Xcsp³ and its ecosystem. *Constraints An Int. J.*, 25(1-2):47–69, 2020. URL: <https://doi.org/10.1007/s10601-019-09307-9>, doi:10.1007/s10601-019-09307-9.
- 2 Dirk Beyer. Progress on software verification: SV-COMP 2022. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part II*, volume 13244 of *Lecture Notes in Computer Science*, pages 375–402. Springer, 2022. URL: https://doi.org/10.1007/978-3-030-99527-0_20, doi:10.1007/978-3-030-99527-0_20.
- 3 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.
- 4 Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ANSI-C programs. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004. URL: https://doi.org/10.1007/978-3-540-24730-2_15, doi:10.1007/978-3-540-24730-2_15.
- 5 Christophe Lecoutre and Olivier Roussel. Proceedings of the 2018 XCSP3 competition. *CoRR*, abs/1901.01830, 2019. URL: <http://arxiv.org/abs/1901.01830>, arXiv:1901.01830.
- 6 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007. URL: https://doi.org/10.1007/978-3-540-74970-7_38, doi:10.1007/978-3-540-74970-7_38.
- 7 Neng-Fa Zhou and Håkan Kjellerstrand. The picat-sat compiler. In Marco Gavanelli and John H. Reppy, editors, *Practical Aspects of Declarative Languages - 18th International Symposium, PADL 2016, St. Petersburg, FL, USA, January 18-19, 2016. Proceedings*, volume 9585 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2016. URL: https://doi.org/10.1007/978-3-319-28228-2_4, doi:10.1007/978-3-319-28228-2_4.