CPMpy_* submissions to the XCSP^3 competition

Tias Guns[1], Wout Vanroose[1], Thomas Sergeys[1], Ignace Bleukx[1], Jo Devriendt[2], Dimos Tsouros[1], and Hélène Verhaeghe[1]

[1]Department of Computer Science, KU Leuven, Belgium
{*tias.guns, wout.vanroose, thomas.sergeys, ignace.bleukx, dimos.tsouros, helene.verhaeghe*}*@kuleuven.be*
[2]Nonfiction Software, Belgium
jo.devriendt@nonfictionsoftware.com

CPMpy [4] is an open-source Python library for modeling combinatorial (optimisation) problems. For this competition, we use CPMpy as a middleware that translates XCSP^3 input to 6 solvers: LCG CP solvers OR-Tools CP-SAT, Chuffed (via MiniZinc); Classic CP solver Gecode (via MiniZinc); SMT solver Z3; ILP solver Gurobi and pseudo-Boolean solver Exact. The purpose is to widen the range of solving technologies evaluated in the competition, while reusing and stress-testing the transformation capabilities of CPMpy. It should be noted that the use of CPMpy as a middleware can add additional overhead and complicates sigterm handling which can impact the performance of the underlying solvers.

Code : CPMpy's code is available on GitHub: https://github.com/CPMpy/cpmpy
Acknowledgement : This research received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant No. 101002802, CHAT-Opt) and Proof of Concept (POC) grant (Grant No.101158376, PO-CPMpy).

Architecture

A simplified overview of the architecture submitted to the competition can be seen in Fig. 1. The grey parts are specific to the XCSP^3 competition, the rest is the standard CPMpy architecture (model in green, transformations in blue, solvers in yellow).

We are very grateful to the PyCSP^3 developers for developing a new and extensively tested Python **parser**[1]. We used the latter's callback mechanism to create a standard CPMpy model on the fly. We did have to extend our set of supported global constraints and decompositions considerably (see below).

CPMpy has a modular collection of **transformations**, that aims to rewrite as little of the original problem formulation as possible. For example, it will not flatten expressions for systems that accept nested expressions (CP languages and SMT solvers), instead decomposing only the non-supported constraints in the expression trees.

We hence call *solve* on the model with the intended solver as argument, which will transform the expressions as needed, post them to the Python API of the underlying solver and call that solver. Our XCSP^3 binary will collect the solver status when finished and print the required XCSP^3 specific **output**.

Solvers

CPMpy can translate to the Python API of many different solvers spanning CP, MIP, SMT, pseudo-Boolean and SAT techniques. These translation algorithms are well-tested using fuzz testing techniques [9]. Our CPMpy submissions bundle the following solvers:

[1]https://github.com/xcsp3team/PyCSP3/tree/master/parser

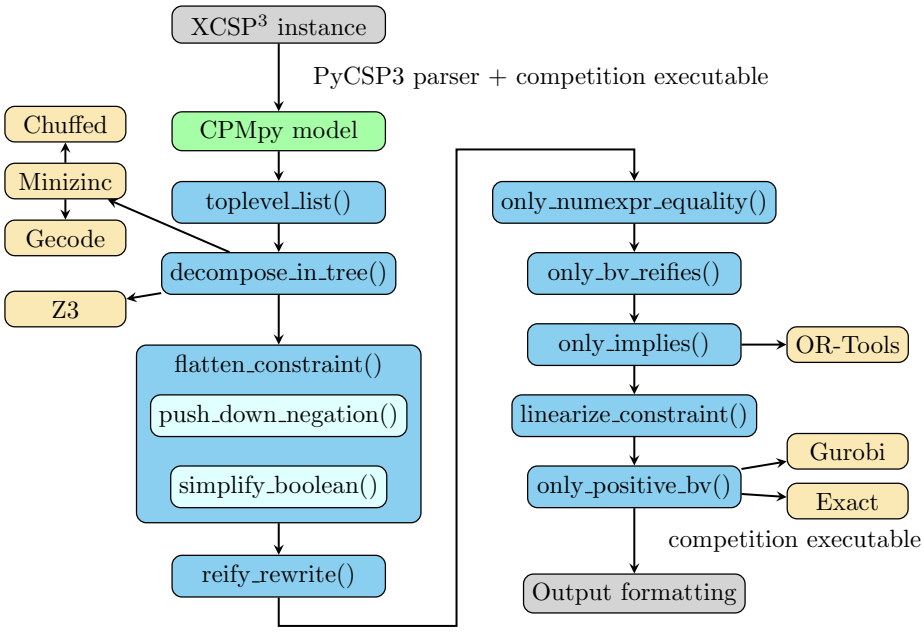

Figure 1: Overview of CPMpy's transformation pipeline

- `cpmpy_ortools` [7] (version: 9.10.4067) The default solver in CPMpy, it is a lazy clause generation CP solver. It supports parallel search too.
- `cpmpy_mnz_chuffed` [1] (version: 0.13.3 (Chuffed) & 2.8.5 (MiniZinc)) Another well-known lazy clause generation solver. It does not have a Python interface, but is bundled with the MiniZinc [6] distribution ('minizinc' Python package).
- `cpmpy_mnz_gecode` [8] (version: 6.3.0 (Gecode) & 2.8.5 (MiniZinc)) A well-known classical CP solver that is also bundled with MiniZinc [6].
- `cpmpy_z3` [2] (version: 4.13.0.0) A versatile SMT theorem prover with an accessible Python API. We also use its optimisation capabilities for optimisation problems.
- `cpmpy_gurobi` [5] (version: 11.0.2) A commercial and high performance mixed integer programming solver.
- `cpmpy_exact` [3] (version: 2.1.0) A pseudo-Boolean solver using cutting-plane conflict analysis. Its API also accepts (reified) integer constraints, internally encoding to pseudo-Boolean constraints.

We did not significantly contribute to the development of any of these solvers, with the exception of the Exact solver, and are grateful to the solver developers for making their technology available.

All submissions have been entered in the following competition tracks: CSP sequential, COP sequential (30'), mini CSP, and mini COP. Only **cpmpy_ortools** has been additionally entered in COP parallel.

Technical Details

Global constraints We added 24 new global constraints from the XCSP^3-core specification that were missing in CPMpy. Some very specific ones will not be added to the main branch.

Overhead We improved the efficiency of our transformations, though there is room for more improvements, especially in *linearize*. We also observed bottlenecks in the PyCSP3 parser for large table constraints, in the OR-Tools type checker when posting large table constraints and in the Z3 Python API.

In-thread execution our XCSP^3 executable runs the solver in the same thread, receiving control back only when the solver stops. This means that our executable cannot report on intermediate solutions during optimisation. It also means we can not gracefully handle a SIGTERM when going over the time limit. We hence call each solver with 5 seconds less than the time limit, to compensate for it timing measurement error.

References

[1] Geoffrey Chu, Peter J Stuckey, Andreas Schutt, Thorsten Ehlers, Graeme Gange, and Kathryn Francis. Chuffed, a lazy clause generation solver. *URL: https://github.com/chuffed/chuffed*, 2018.

[2] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[3] Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-boolean solving. In *IJCAI*, volume 18, pages 1291–1299, 2018. URL: https://gitlab.com/JoD/exact.

[4] Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.

[5] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

[6] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[7] Laurent Perron and Frédéric Didier. Cp-sat. URL: https://developers.google.com/optimization/cp/cp_solver/.

[8] Christian Schulte, Mikael Lagerkvist, and Guido Tack. Gecode. *Software download and online material at the website: http://www. gecode. org*, pages 11–13, 2006.

[9] Wout Vanroose, Ignace Bleukx, Jo Devriendt, Dimos Tsouros, Hélène Verhaeghe, and Tias Guns. Mutational fuzz testing for constraint modeling systems. In *Principles and Practice of Constraint Programming - CP 2024, 30th International Conference, CP 2024, Girona, Spain, September 2-6, 2024, Proceedings*, 2024.