CPMpy_* submissions to the XCSP^3 competition

Tias Guns[1], Thomas Sergeys[1], Ignace Bleukx[1], Dimos Tsouros[1], and Hendrik Bierlee[1]

[1]Department of Computer Science, KU Leuven, Belgium
tias.guns, thomas.sergeys, ignace.bleukx, dimos.tsouros, henk.bierlee @kuleuven.be

CPMpy [4] is an open-source Python library for modeling combinatorial (optimisation) problems. For this competition, we use CPMpy as a middleware that can translate XCSP^3 input to 7 solvers of varying technologies: LCG CP solvers OR-Tools CP-SAT, Chuffed (via MiniZinc); Classic CP solvers Gecode (via MiniZinc), CP Optimizer; SMT solver Z3; ILP solver Gurobi; and pseudo-Boolean solver Exact. The purpose of our submission is to widen the range of solving technologies evaluated in the competition, while reusing and stress-testing the transformation capabilities of CPMpy.

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 gray node and its outgoing arc represent elements specific to XCSP^3, like the input format, the parser and the accompanying CPMpy tooling. We are very grateful to the PyCSP^3 developers for providing an extensively tested Python **parser**[1]. We used the latter's callback mechanism to create a standard CPMpy model on the fly. All CPMpy **tooling** for the XCSP^3 format, including a competition-compatible CLI, have become part of the library since the recent 0.9.26 release. The red, green, and blue components signify a subset of the standard CPMpy "*waterfall*" transformation architecture. 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. At last, the orange boxes hold the different **solver backends** we've submitted to this year's competition.

After parsing to a CPMpy model, 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. We then 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 technologies. These translation algorithms are well-tested using fuzz testing techniques [10]. Our CPMpy submissions bundle the following solvers:

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

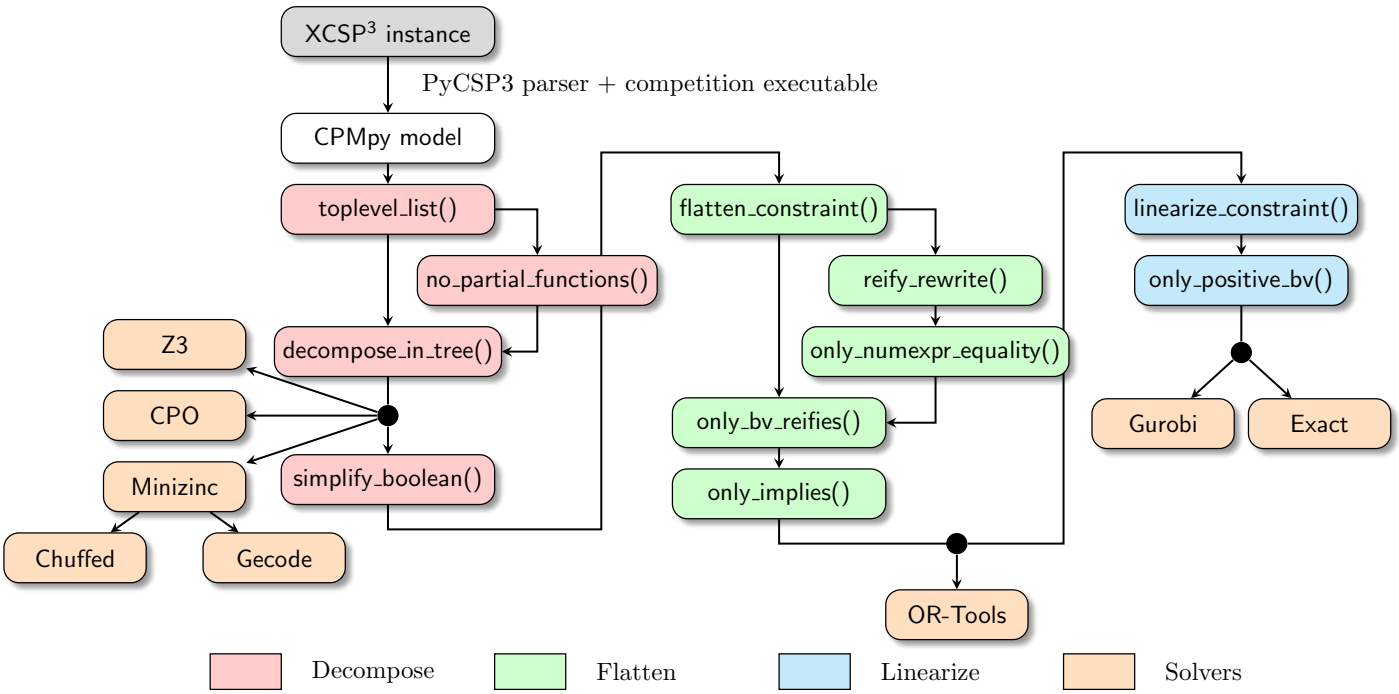

Figure 1: Subset of CPMpy's transformation pipeline as submitted to the XCSP^3 2025 competition

- `cpmpy_ortools` [8] (version: 9.12.4544) The default solver in CPMpy, it is a lazy clause generation CP solver. It supports parallel search too.
- `cpmpy_cpo` [6] (version: 22.1.1.0) A well-known commercial CP solver, particularly suitable for scheduling problems.
- `cpmpy_mzn_chuffed` [1] (version: 0.13.2 (Chuffed) & 2.9.0 (MiniZinc)) Another well-known lazy clause generation solver. It does not have a Python interface, but is bundled with the MiniZinc [7] distribution ('minizinc' Python package).
- `cpmpy_mzn_gecode` [9] (version: 6.3.0 (Gecode) & 2.9.0 (MiniZinc)) A well-known classical CP solver that is also bundled with MiniZinc [7].
- `cpmpy_z3` [2] (version: 4.15.0.0) A versatile SMT theorem prover with an accessible Python API. We also use its optimisation capabilities for optimisation problems.
- `cpmpy_gurobi` [5] (version: 12.0.2) 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') and COP sequential (3'). Only **cpmpy_ortools**, **cpmpy_cpo**, and **cpmpy_gurobi** have additionally been entered in COP parallel.

Technical Details

Tooling We created new tooling for the XCSP^3 format, including improved PyCSP^3 callbacks, an easy to use model loader, a competition XML solution printer, a pytorch-compatible dataset loader and a benchmarking script. All have been added to the library in the recent 0.9.26 release.

Globals Besides the globals from last year, additional support for *notin*, *atleast*, and *element_matrix* have been added.

Transformations All decompositions have been revisited, adding a set of ILP-friendly and SMT-friendly decompositions, alternative decompositions when in numerical context and a dynamic decomposition for *Cumulative* depending on problem dimensions (time- vs task-resource). The submission additionaly includes a very limited implementation of context-aware flattening.

CSE The use of Common Subexpression Elimination has been expanded and improved upon, eliminating duplicates of expressions as to only post a single copy to the backend solver. CSE is now part of the general release of CPMpy since version 0.9.26.

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] IBM. Cp optimizer. URL: https://www.ibm.com/products/ilog-cplex-optimization-studio/cp-optimizer.

[7] 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.

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

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

[10] 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.