Pindakaas: A SAT encoding library

Hendrik *Henk* Bierlee @ DTAI, KU Leuven, Belgium

PINDAKAAS
Pseudo-Boolean Spread

SAT-based solvers are...

Efficient Winning competitions (e.g. MiniZinc Challenge, XCSP3 Competition) and applied in industry (e.g. scheduling problems, hardware verification)

Versatile Incremental solving, assumptions, model counting, optimization, explainability, certified solving via proof logging, user propagaters, ...

Hard to use The input problem almost always requires a translation (i.e. ***encoding***) into SAT

The quality of the encoding is **critical** for solver performance

Minimal encode-and-solve example

A XOR-constraint over two Boolean variables p, q is encoded by a ***conjunction*** of ***clauses*** (i.e. disjunctions) of ***literals*** (e.g. p or its negation $\neg p$):

$p \oplus q$	**Input problem with $p, q \in \mathbb{B}$**
$(p \vee q) \wedge (\neg p \vee \neg q)$	**Encode (using Pindakaas)**
$(p \vee q) \wedge (\neg p \vee \neg q)$	**Search decision ($p \mapsto$ true)**
$(p \vee q) \wedge (\neg p \vee \neg q)$	**Unit propagation ($q \mapsto$ false)**
$\{(p \mapsto \text{true}), (q \mapsto \text{false})\}$	**Solution**

Pindakaas: usage and features

Pindakaas is a SAT encoding library which can model non-SAT problems and solve them with SAT solvers:

```python
import pindakaas as pdk
cadical = pdk.CaDiCaL()  # SAT solver
p, q = cadical.new_vars(2)
cadical += p ^ q  # XOR constraint
with cadical.solve() as result:
    assert result.value(p) != result.value(q)
```

Features

- Encode high-level propositional, pseudo-Boolean, and integer linear constraints into SAT
- Supports incremental solving with assumptions
- Rust implementation + Python interface
- Supports various SOTA and novel encoding methods

Integrations

CPMpy Constraint Programming and Modelling library
- Developed @ KU Leuven
- Supports a wider collection of high-level constraints
- Transformation to various solvers

Huub Lazy Clause Generation solver
- Developed @ Monash University, Australia

Single Constant Multiplication [1]

Often, input problems contain linear constraints over finite-domain ***integer*** variables, e.g.:

$$117 \cdot x + 256 \cdot y + 42 \cdot z + \ldots \leq 1000$$

If x has at most 8 domain values, an encoding of x requires 4 Boolean variables using a ***binary bit-pattern***. However, to encode the constraint, we also require an encoding of $117 \cdot x$.

We can compute a ***decomposition*** of cheap additions, subtractions, and multiplications by powers-of-two:

$$\begin{aligned} & 2^1 x && + x && = 3x \\ \wedge\ & 2^4 \cdot (3x) && + x && = 49x \\ \wedge\ & x && + 2^4 \cdot x && = 17x \\ \wedge\ & 2^1 \cdot (17x) && + (49x) && = 117x \end{aligned}$$

Encoding the decompositions requires few clauses, because:

- All multiplications require zero clauses as they are bit-shifts
- The 2nd and 3rd addition requires zero clauses if x is encoded by exactly 4 bits

Computing these decompositions is NP-hard, so we ***pre-compute*** a database for a range of constants and bit-widths.

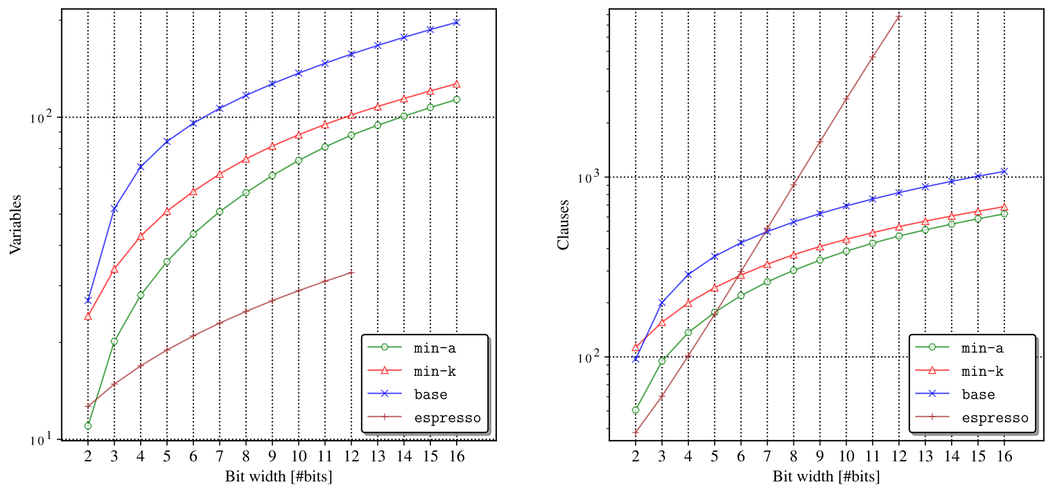

Fig. 1: Avg. #variables/#clauses of the pre-computed databases for bit-widths of x. The `espresso` method minimizes #variables, while `min-k` and `min-a` minimize #clauses.

Future research directions

- Releasing integer linear related work
- Proof-logging to certify the correctness of the encoding
- Inclusion in next CPMpy version

[1] H. Bierlee, J. J. Dekker, V. Lagoon, P. J. Stuckey, and G. Tack, "Single Constant Multiplication for SAT," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28-31, 2024, Proceedings, Part I*, B. Dilkina, Ed., in Lecture Notes in Computer Science, vol. 14742. Springer, 2024, pp. 84–98. doi: 10.1007/978-3-031-60597-0_6.

[1] Paper Repo

henk.bierlee@kuleuven.be
pindakaashq/pindakaas

Ack. ERC grant CHAT-Opt (ID: 101002802)

KU LEUVEN