Concrete 3.5: A CSP solving software & API

Julien Vion
http://github.com/concrete-cp/concrete

1 Features

Concrete is a CSP constraint solver written in Scala 2.12 [15]. We always try to use up-to-date dependencies. Concrete is a pretty standard CP solver, which solves CSP instances using depth-first search and AC or weaker variants for propagation. The two main specific aspects of Concrete are:

- the use of persistent data structures [16] for managing domain states and some constraint states. We use bit vectors copied on-the fly, hash tries, trees with a high branching factor, and red-black trees. For the state of many constraints, semi-persistent data structures (mainly sparse sets [4]) or backtrack-stable data (watched literals [5] or residues [9]) are preferred.
- the use of the companion project CSPOM [19], a solver-independent modeling assistant able to perform automatic reformulation such as constraint aggregation. CSPOM is able to parse problems written in FlatZinc [13], XCSP3 [1], the legacy XCSP2 format or its own Java and Scala DSL (yet to be documented).

Concrete can solve models defined with signed 32-bit integers. Domains are internally represented using either intervals or bit vectors, with a specialization for singleton and boolean domains. CSPOM represents domains with interval trees and supports infinite domains, but they must be fully defined during the compilation phase in order to be processed in Concrete. Set variables are currently not supported.

The main loop of Concrete is a tail-recursive DFS. It allows to enumerate solutions or to search for an optimal solution. If used correctly, it is able to add constraints dynamically between solutions.

Concrete natively supports the following constraints:

- Extension (list of allowed or forbidden tuples). An optimized algorithm should be automatically selected for binary constraints (AC3-bit+rm) [10] or MDD [21].
- Linear ($a \cdot x + b \cdot y + \ldots \ \{=, <, \leq, \neq\} \ k$). Bound consistency (except for $\neq$) [7] or full domain consistency for ternary constraints (using residues).

- Absolute value ($x = |y|$). Bound or domain consistency (using residues).
- Distance ($x = |y - z|$). Bound or domain consistency (using residues).
- All-different with 2-consistency or bound consistency [12].
- Cardinality (AtLeast/AtMost)
- Bin-packing [17]
- Channel ($x(i) = j \iff x(j) = i$)
- Boolean clauses and XOR (using watched literals)
- Cumulative using profile and energetic reasoning
- Rectangle packing (diffN) using quad-trees and energetic reasoning
- Integer division and modulo. Bound or domain consistency (using residues)
- Element / Member (using watched literals and residues)
- Inverse ($x(i) = j \implies y(j) = i$)
- Lex-Leq
- Lex-Neq
- Min/Max
- Quadratic ($x = y \cdot z$, $x = y^2$). Bound or domain consistency (using residues)
- Generic reification (for any constraint C, a boolean variable b can be defined s.t. $b \implies C$)

All other documented MiniZinc constraints are supported via decomposition or reformulation. All other XCSP3 constraints selected for the 2017 competition are supported via decomposition or reformulation. Some XCSP3 constraints are not supported.

2 Search strategies

Concrete solves CSP/COP using a binary depth-first tree search [8]. The default variable ordering heuristic is *dom/wdeg* [3] with incremental computation and random tiebreaking. The default value heuristic chooses the best known value first [22], then the largest value from the domain. Sometimes, a random value is selected to improve diversity in search. Search is restarted periodically (with a geometric growth) to reduce long tails of search time [6].

Propagation queue is managed using a coarse-grained constraint-oriented propagation scheme [2] with dynamic and constraint-specific propagation ordering heuristic [20]. Constraint entailment is managed when it can be detected easily.

3 Present and near-future of Concrete

Feedback from the competition allowed us to improve Concrete in many ways in late 2017. Bugs have been fixed, some heuristics have been improved: it seems that choosing the largest value first for branching was counter-productive for many instances (e.g., bin packing), and the randomization was set too high. We are in the process of performing a large-scale evaluation of diversification/restart strategies in search and the corresponding links with LNS. Since the competition, the way value ordering heuristics work in Concrete has been revamped. Now a portfolio of heuristics can be parameterized, which should allow fancy things such as separate randomization for both variable and value heuristics, tie-breaking and cascading fallback heuristics. A new value ordering heuristic which minimizes the bound of the optimization variable has been implemented.

A new variable ordering heuristic is under development, whose objective is to improve the behavior of *dom/wdeg* when constraints of large arity are involved. In the long term, heuristics inspired from SAT solvers will probably be considered.

Concrete's main loop, element and min/max constraints have been recently rewritten to improve performance (both in memory and time), as well as code readability.

A new domain representation using red-black trees is now available, which should improve the representation of sparse domains. A new domain type based on set difference is being studied.

We recently reimplemented nogood recording from restarts [11], which was available in older versions of Concrete for binary nogoods only, but was dropped off a few years ago. A full nogood-managing global constraint is now available with innovating tricks.

We are currently finishing the implementation of common subexpression elimination for CSPOM [14].

References

[1] G. Audemard, F. Boussemart, C. Lecoutre, C. Piette, et al. *XCSP3*. `http://www.xcsp.org`. 2016.

[2] F. Boussemart, F. Hemery, and C. Lecoutre. "Revision Ordering Heuristics for the CSP". In: *Proc. CPAI'04 workshop held with CP'04*. Toronto, Canada, 2004, pp. 29–43.

[3] F. Boussemart, F. Hemery, C. Lecoutre, and L. Saïs. "Boosting Systematic Search by Weighting Constraints". In: *Proc. of the 16th European Conference on Artificial Intelligence (ECAI)*. 2004, pp. 146–150.

[4] P. Briggs and L. Torczon. “An Efficient Representation for Sparse Sets”. In: *ACM Letters on Programming Languages and Systems* 2.1–4 (1993), pp. 59–69.

[5] I. P. Gent, C. Jefferson, and I. Miguel. “Watched literals for constraint propagation in Minion”. In: *Proceedings of CP'06*. 2006, pp. 182–197.

[6] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. “Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems”. In: *Journal of Automated Reasoning* 24.1 (2000), pp. 67–100.

[7] W. Harvey and J. Schimpf. “Bounds Consistency Techniques for Long Linear Constraints”. In: *In Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems*. 2002, pp. 39–46.

[8] J. Hwang and D.G. Mitchell. “2-way vs d-way branching for CSP”. In: *Proc. of the 11th Itl. Conf. on Principles and Practice of Constraint Programming (CP)*. 2005, pp. 343–357.

[9] C. Lecoutre and F. Hemery. “A Study of Residual Supports in Arc Consistency”. In: *Proceedings of IJCAI'2007*. 2007, pp. 125–130.

[10] C. Lecoutre and J. Vion. “Enforcing AC using Bitwise Operations”. In: *Constraint Programming Letters* 2 (2008), pp. 21–35.

[11] C. Lecoutre, L. Saïs, S. Tabary, and V. Vidal. “Nogood Recording from Restarts”. In: *Proceedings of IJCAI'07*. 2007.

[12] A. López-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek. “A fast and simple algorithm for bounds consistency of the alldifferent constraint”. In: *Proc. IJCAI'03*. 2003, pp. 245–250.

[13] N. Nethercote, P.J. Stuckey, R. Becket, S. Brand, G.J. Duck, and G. Tack. “Minizinc: Towards a standard CP modelling language”. In: *Proc. CP'2007*. Ed. by C. Bessière. 2007, pp. 529–543.

[14] P. Nightingale, Ö. Akgün, I. P. Gent, C. Jefferson, I. Miguel, and P. Spracklen. “Automatically Improving Constraint Models in Savile Row”. In: *Artificial Intelligence* 251.Supplement C (2017), pp. 35–61.

[15] M. Odersky et al. *The Scala Programming Language*. `http://www.scala-lang.org/`. 2001.

[16] C. Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998.

[17] P. Shaw. “A Constraint for Bin Packing”. In: *Proc. CP'2004*. Ed. by M. Wallace. 2004, pp. 648–662.

[18] R.M. Stallman. *GNU Lesser General Public License*. GNU Project–Free Software Foundation, http://gnu.org/licenses. 1999.

[19] J. Vion. *CSP Object Model*. `http://github.com/concrete-cp/cspom`. 2008–2016.

[20] J. Vion and S. Piechowiak. “Handling Heterogeneous Constraints in Revision Ordering Heuristics”. In: *Proc. of the TRICS'2010 workshop held in conjunction with CP'2010*. 2010.

[21] J. Vion and S. Piechowiak. “Maintenir des MDD persistants pour établir la consistance d'arc”. French. In: *Revue d'Intelligence Artificielle* 28.5 (2014), pp. 547–569.

[22] J. Vion and S. Piechowiak. “Une simple heuristique pour rapprocher DFS et LNS pour les COP”. In: *Actes des 13^e JFPC*. 2017, pp. 39–44.