Concrete 3.9.2: 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 [6] or residues [9]) are preferred.
- the use of the companion project CSPOM 2.25 [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, bit vectors, or red-black trees depending on domain density. Singleton and boolean domains are handled specifically. 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. This allows to infer some variable domains, e.g. for auxiliary variables or variables defined by a constraint. 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) [11] or MDD [20].
- 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
- Quadratic ($x = y \cdot z$, $x = y^2$). Bound or domain consistency (using residues)
- 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
- Subcircuit with defined starting point (uses Dijkstra shortest path algorithm)
- Regular and Sliding-Sum *via* MDD decomposition
- Generic reification (for any constraint C, a boolean variable b can be defined s.t. $b \implies C$)

All FlatZinc constraints are supported, and other documented MiniZinc constraints are supported *via* provided decomposition or reformulation. All XCSP3 constraints selected for the 2018 competition are supported *via* (trivial) decomposition or reformulation. Some XCSP3 constraints may not be supported.

2 Search strategies

Concrete solves CSP/COP using a binary depth-first tree search [8]. The default variable ordering heuristic is *dom/wdeg* [2] with incremental computation and random tiebreaking. The default value heuristic chooses the best known value first [22], then applies BBS, an heuristic that uses singleton assignments to find the value that maximizes potential solution quality [5]. Ties are broken randomly, with prority given to domain bounds. 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 [**Gomes2000**].

Propagation queue is managed using a coarse-grained constraint-oriented propagation scheme [3] with dynamic and constraint-specific propagation ordering heuristic [21]. 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. For 2018, we implemented new constraints selected for the competition (mainly Subcircuit), BBS, and tuned some parameters.

We recently reimplemented nogood recording from restarts [10], 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.

Common subexpression elimination (ACCSE) [14] for CSPOM is now fast and stable.

References

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

[2] F. Boussemart, F. Hemery, C. Lecoutre, and L. Saïs. "Boosting Systematic Search by Weighting Constraints". In: *Proc. 16th ECAI*. 2004, pp. 146–150.

[3] 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.

[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] J.-G. Fages and C. Prud'Homme. “Making the first solution good!” In: *Proc. 29th ICTAI*. Boston, MA, United States, Nov. 2017.

[6] I. P. Gent, C. Jefferson, and I. Miguel. “Watched literals for constraint propagation in Minion”. In: *Proc. 12th CP*. 2006, pp. 182–197.

[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. 11th CP*. 2005, pp. 343–357.

[9] C. Lecoutre and F. Hemery. “A Study of Residual Supports in Arc Consistency”. In: *Proc. 20th IJCAI*. 2007, pp. 125–130.

[10] C. Lecoutre, L. Saïs, S. Tabary, and V. Vidal. “Nogood Recording from Restarts”. In: *Proc. 20th IJCAI*. 2007.

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

[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. 18th IJCAI*. 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. 13th CP*. 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. 10th CP*. 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. “From MDD to BDD and Arc consistency”. In: *Constraints* (July 2018). DOI: `10.1007/s10601-018-9286-5`.

[21] 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, pp. 62–82.

[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.