A Picat-based XCSP Solver – from Parsing, Modeling, to SAT Encoding

Neng-Fa Zhou[1] and Håkan Kjellerstrand[2]

[1] CUNY Brooklyn College & Graduate Center
[2] hakank.org

Abstract. This short paper gives an overview of a Picat-based XCSP3 solver, named PicatSAT, submitted to the 2018 XCSP competition. The solver demonstrates the strengths of Picat, a logic-based language, in parsing, modeling, and encoding constraints into SAT.

XCSP3

XCSP3 [1] is an XML-based domain specific language for describing constraint satisfaction and optimization problems (CSP). XCSP3 is positioned as an intermediate language for CSPs. It does not provide high-level constructs as seen in modeling languages. However, XCSP3 is significantly more complex than a canonical-form language, like FlatZinc. A constraint can sometimes be described in either the standard format or simplified format. The advanced format, which is used by group and matrix constraints, allows more compact description of constraints.

Picat

Picat [6] is a simple, and yet powerful, logic-based multi-paradigm programming language. Picat is a Prolog-like rule-based language, in which predicates, functions, and actors are defined with pattern-matching rules. Picat incorporates many declarative language features for better productivity of software development, including explicit non-determinism, explicit unification, functions, list comprehensions, constraints, and tabling. Picat also provides imperative language constructs, such as assignments and loops, for programming everyday things. Picat provides facilities for solving combinatorial search problems, including a common interface with CP, SAT, and MIP solvers, tabling for dynamic programming, and a module for planning. PicatSAT uses the SAT module, which generally performs better than the CP and MIP modules on competition benchmarks.

Parsing

The availability of different formats in XCSP3 makes it a challenge to parse the XCSP3 language. A parser implemented in C++ by the XCSP designers has

more than 10,000 lines of code. The entire Picat implementation of XCSP3 has about 2,000 lines of code, two-thirds of which is devoted to parsing and syntax-directed translation. As illustrated in the following example, Picat is well suited to parsing.

```
% E -> T E'
ex(Si,So) => term(Si,S1), ex_prime(S1,So).

% E' -> + T E' | - T E' | e
ex_prime(['+'|Si],So) =>
    term(Si,S1),
    ex_prime(S1,So).
ex_prime(['-'|Si],So) =>
    term(Si,S1),
    ex_prime(S1,So).
ex_prime(Si,So) => So = Si.
```

The parser follows the framework for translating context-free grammar into Prolog [3]: a non-terminal is encoded as a predicate that takes an input string (`Si`) and an output string (`So`), and when the predicate succeeds, the difference `Si-So` constitutes a string that matches the nonterminal. Unlike in Prolog, pattern-matching rules in Picat are fully indexed, which facilitates selecting right rules based on look-ahead tokens.

Modeling

It is well known that loops and list comprehensions are a necessity for modeling CSPs. The following Picat example illustrates the convenience of these language constructs for modeling.

```
post_constr(allDifferentMatrix(Matrix)) =>
    NRows = len(Matrix),
    NCols = len(Matrix[1]),
    foreach (I in 1..NRows)
        all_different(Matrix[I])
    end,
    foreach (J in 1..NCols)
        all_different([Matrix[I,J] : I in 1..NRows])
    end.
```

The `allDifferentMatrix(Matrix)` constraint takes a matrix that is represented as a two-dimensional array, and posts an `all_different` constraint for each row and each column of the matrix.

SAT Encoding

PicatSAT adopts the log encoding for domain variables. While log encoding had been perceived to be unsuited to arithmetic constraints due to its hindrance

to propagation [2], we have shown that log encoding can be made competitive with optimizations [4]. There are hundreds of optimizations implemented in PicatSAT, and they are described easily as pattern-matching rules in Picat. We have also shown that, with specialization, the binary adder encoding is not only compact, but also generally more efficient than BDD encodings for PB constraints [5]. PicatSAT adopts specialized decomposition algorithms for some of the global constraints. While competitive overall, PicatSAT is not competitive with state-of-the-art CP solvers on benchmarks that use path-finding constraints that require reachability checking during search. The future work is to design efficient encodings for these global constraints.

References

1. Frederic Boussemart, Christophe Lecoutre, and Gilles Audemard. XCSP3 - an integrated format for benchmarking combinatorial constrained problems. Technical report, xcsp.org.
2. Donald Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley, 2015.
3. Fernando C. N. Pereira and David H. D. Warren. Definite clause grammars for language analysis - A survey of the formalism and a comparison with augmented transition networks. *Artif. Intell.*, 13(3):231–278, 1980.
4. Neng-Fa Zhou and Håkan Kjellerstrand. Optimizing SAT encodings for arithmetic constraints. In *CP*, pages 671–686, 2017.
5. Neng-Fa Zhou and Håkan Kjellerstrand. Encoding pb constraints into sat via binary adders and bdds – revisited. In *Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion RCRA*, 2018.
6. Neng-Fa Zhou, Håkan Kjellerstrand, and Jonathan Fruhman. *Constraint Solving and Planning with Picat*. Springer, 2015.