PATTINT: Symbolic Numeric Planning with Patterns Over Integers

Itay Elyashiv[1], Matteo Cardellini[2], Enrico Giunchiglia[2], Alexander Shleyfman[1], Yoni Zohar[1]

[1]Bar-Ilan University, Ramat Gan, Israel
[2]Universita degli Studi di Genova, Genova, Italy
firstAuthor@affiliation1.com, {matteo.cardellini, enrico.giunchiglia}@unige.it, {alexash, yoni.zohar}@biu.ac.il

Abstract

PATTINT is a planner for cost-optimal numeric planning built on top of the PATTY framework. The planner targets Symbolic Pattern Planning over integer-valued numeric tasks. Given a planning task, a cost bound, and a pattern represented as a sequence of actions, PATTINT encodes the existence of a valid plan satisfying the pattern and the bound as a Satisfiability Modulo Theories (SMT) formula. This formula is then passed to an SMT solver, which either returns a satisfying assignment corresponding to a valid plan, or reports unsatisfiability.

The main contribution of PATTINT is a transformation of numeric planning tasks into an integer form, allowing the planner to exploit SMT solvers with efficient support for integer arithmetic. In many cases, this integer encoding leads to substantially faster solving than the original encoding used by PATTY. PATTINT therefore provides a practical extension of symbolic pattern-based planning to integer numeric domains, improving the efficiency of cost-optimal numeric planning while preserving the declarative SMT-based structure of the original framework.

Introduction

Numeric planning extends classical planning with state variables ranging over numerical domains, enabling compact models of resources, capacities, counters, and accumulated costs. This additional expressiveness is essential for many realistic planning tasks, but it also makes solving harder: even relatively restricted numeric fragments usually turn out to be undecidable (Helmert 2002; Gnad et al. 2023). The PDDL2.1 language made numeric fluents a central part of the standard planning formalism (Fox and Long 2003), and subsequent work has explored both heuristic-search and symbolic methods for such tasks.

Symbolic planning methods reduce bounded plan existence to the satisfiability of a logical formula, following the classical planning-as-satisfiability paradigm (Kautz and Selman 1992). For numeric planning, Satisfiability Modulo Theories (SMT) based encodings are particularly natural, since they allow the transition relation, action applicability conditions, and numeric effects to be represented directly in arithmetic theories. Existing SMT encodings for numeric planning include rolled-up encodings and relaxed exists-step formulations, which aggregate or partially order action applications in order to reduce the required bound and formula size (Scala et al. 2016; Bofill, Espasa, and Villaret 2017).

Recently, Symbolic Pattern Planning introduced a different way to structure such encodings (Cardellini, Giunchiglia, and Maratea 2026). Instead of encoding arbitrary action choices at each bounded step, the method receives a pattern, i.e., a finite sequence of actions, and encodes the existence of a valid plan among executions induced by that pattern. The PATTY planner instantiated this idea for numeric planning and showed that pattern-based encodings can dominate previous symbolic encodings while producing substantially smaller formulas in important cases (Cardellini, Giunchiglia, and Maratea 2024). This suggests that the performance of SMT-based numeric planning depends not only on the underlying solver, but also on the arithmetic theory and representation exposed to it.

This paper presents PATTYINT, a planner for satisficing numeric planning built on top of the PATTY framework. The main idea is to exploit the integer structure that arises in many numeric planning tasks, specifically in simple numeric planning (SNP), a restricted numeric fragment related to the monotonic numeric setting of Metric-FF (Hoffmann 2003) and later formalized through simple numeric conditions (Scala, Haslum, and Thiébaux 2016). For some numeric planning fragments, integer-valued formulations can be obtained by compilation; however, planning over integers remains undecidable already under severe syntactic restrictions, including tasks with only three numeric variables (Helmert 2002; Gnad et al. 2023). Nevertheless, when a bounded symbolic-pattern encoding is considered, integer arithmetic gives a precise and solver-friendly representation of numeric fluents, action applications along the pattern, and cost bounds. PATTYINT therefore transforms suitable numeric tasks into an integer form and submits the resulting SMT formulas to an integer-arithmetic SMT solver, Bitwuzla (Niemetz, Preiner, and Zohar 2024) in our implementation (Barbosa et al. 2022).

The contribution of this tool is a planner-level integration of integer compilation with symbolic pattern planning, preserving the declarative bounded-plan-existence view of PATTY while changing the arithmetic theory used by the solver. In this way, PATTYINT connects the theory of planning over integers with recent progress in symbolic pattern-based numeric planning, and provides a concrete experimen-

tal test of whether integer encodings can improve optimal numeric planning in practice.

Implementation

PATTYINT is implemented in Python, and takes as input a PDDL domain and a problem file together with a user-specified bound, solver to use and pattern computation strategy.

Grounding. The domain and problem are parsed and grounded using the same infrastructure as PATTY. Grounding instantiates all action schemas and produces a *grounded domain* containing the set of ground actions, propositional atoms, and numeric functions. An Asymptotic relaxed planning graph (ARPG) is computed over the grounded domain; it is used both to order actions in the pattern and to derive reachability intervals that prune the search space.

Pattern Construction. A pattern is an ordered sequence of ground actions. The pattern construction is implemented as part of the grounding process. PATTYINT supports two strategies for constructing the pattern: an ARPG-based ordering that sequences actions according to their first appearance in the relaxed planning graph, and a random ordering. A dummy action is appended to the end of the pattern to serve as the step anchor.(Cardellini, Giunchiglia, and Maratea 2024).

Integer Compilation. As stated in the introduction, the key contribution of PATTYINT over PATTY is a compilation of numeric planning tasks into the quantifier-free bit-vector theory (QF_BV). In attempt to compile floating point problems, a scale factors is computed from the planning task. The *scale factor* σ is the least common multiple of the denominators of all rational constants appearing in action preconditions, effects, the initial state and the goal formula. Multiplying every numeric constant by the scale factor produces an equivalent task with integer-valued constants. An important remark is that in complex linear planning problems, the scale factor needs to be calculated recursively and is possibly infinite, hence the best current solution is falling back to The original PATTY and solve using the (QF_BV) theory. A minimum bit-width w is then determined by finding the largest scaled constant, adding one bit for the sign, one bit as a safety margin, and one bit for intermediate sums in goal comparisons. Numeric state variables are encoded as w-bit signed bit-vectors. Action-count variables use a narrower width w_a, computed from an estimate of the maximum useful repetition count, keeping the action sub-formula compact.

SMT Encoding. For a given bound b, the encoding introduces $b + 1$ layers of *transition variables*. Each layer i contains:

- *Value variables* $v_i(f)$: a $\mathrm{BV}(w)$ variable for each numeric function f and a Boolean variable for each propositional atom, representing the state after step i;
- *Delta variables* $\delta_i(a, f)$: a $\mathrm{BV}(w)$ variable for each function f and action a in the pattern, representing the value of f after executing all pattern actions up to and including a in step i;
- *Action-count variables* $n_i(a)$: a $\mathrm{BV}(w_a)$ variable for each action a, representing the number of times a fires in step i;
- *Auxiliary variables* $u_i(a, f)$: used to handle assignment effects and non-simple linear increments.

The formula for step i comprises five groups of constraints. *Delta rules* propagate state through the pattern: the delta of the first action copies the previous value layer, and each subsequent action in the pattern accumulates the effects of its predecessor via BV multiplication of the action count and the scaled constant. *Action-count rules* constrain each $n_i(a)$ to lie in $[0, 1]$ for non-repeatable actions and $[0, M]$ otherwise, where M is derived from the initial state. ARPG intervals provide additional tightening. *Precondition rules* assert, for each action a, that $n_i(a) > 0$ implies the preconditions hold at $\delta_i(a, \cdot)$; for repeatable actions an additional implication handles the multi-application case by substituting the transformed fluent values. *Effect rules* handle assignment effects and non-simple increments through auxiliary variables. *Frame rules* connect the end of each pattern repetition to the next value layer.

The initial-state formula fixes value variables at layer 0 to their scaled initial values; negative Boolean atoms are explicitly set to false. The goal formula constrains the value variables at layer b using the scaled goal constants.

Solving and Plan Extraction. The SMT formula is submitted to an SMT solver via the pySMT interface (Gario and Micheli 2015). PATTYINT supports (mainly but not exclusively) Bitwuzla (Niemetz, Preiner, and Zohar 2024) for QF_BV. If the formula is satisfiable the solver returns a model; action-count variable values are read from the model, divided by any linearization factor, and assembled into a `NumericPlan`. The plan is then validated against the original unscaled problem. If the formula is unsatisfiable or the plan is invalid, the bound is incremented by one and the process repeats up to a configurable maximum bound.

Configuration As stated, PATTYINT accepts several arguments. The configured arguments are Bitwuzla as the solver, ARPG pattern mining strategy, and it doesn't specify a maximum bound (In order to allow solving complicated problems).

Acknowledgments

A big thank you to the organizers of the numeric tracks of the 2026 International Planning Competition, whose work made this benchmark effort possible and much more enjoyable.

References

Barbosa, H.; Barrett, C. W.; Brain, M.; Kremer, G.; Lachnitt, H.; Mann, M.; Mohamed, A.; Mohamed, M.; Niemetz, A.; Nötzli, A.; Ozdemir, A.; Preiner, M.; Reynolds, A.; Sheng, Y.; Tinelli, C.; and Zohar, Y. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In Fisman, D.;

and Rosu, G., eds., *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, 415–442. Springer.

Bofill, M.; Espasa, J.; and Villaret, M. 2017. Relaxed Exists-Step Plans in Planning as SMT. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, 563–570.

Cardellini, M.; Giunchiglia, E.; and Maratea, M. 2024. Symbolic Numeric Planning with Patterns. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024*, 20070–20077. AAAI Press.

Cardellini, M.; Giunchiglia, E.; and Maratea, M. 2026. Symbolic pattern planning. *Artificial Intelligence*, 352: 104482.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20: 61–124.

Gario, M.; and Micheli, A. 2015. PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In *SMT Workshop 2015*.

Gnad, D.; Helmert, M.; Jonsson, P.; and Shleyfman, A. 2023. Planning over Integers: Compilations and Undecidability. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, Prague, Czech Republic, July 8-13, 2023*, 148–152. AAAI Press.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, 44–53. AAAI.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research*, 20: 291–341.

Kautz, H. A.; and Selman, B. 1992. Planning as Satisfiability. In Neumann, B., ed., *10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings*, 359–363. John Wiley and Sons.

Niemetz, A.; Preiner, M.; and Zohar, Y. 2024. Scalable Bit-Blasting with Abstractions. In Gurfinkel, A.; and Ganesh, V., eds., *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, 178–200. Springer.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for Numeric Planning via Subgoaling. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, 3228–3234. IJCAI/AAAI Press.

Scala, E.; Ramírez, M.; Haslum, P.; and Thiébaux, S. 2016. Numeric Planning with Disjunctive Global Constraints via SMT. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016*, 276–284. AAAI Press.