νZ - An Optimizing SMT Solver

Nikolaj Bjørner[1], Anh-Dung Phan[2], and Lars Fleckenstein[3]

[1] Microsoft Research, Redmond, WA, USA
[2] DTU Compute, Technical University of Denmark
[3] Microsoft Dynamics, Vedbæk, Denmark
nbjorner@microsoft.com, padu@dtu.dk, LarsFleckenstein@outlook.com

Abstract. νZ is a part of the SMT solver Z3. It allows users to pose and solve optimization problems modulo theories. Many SMT applications use models to provide satisfying assignments, and a growing number of these build on top of Z3 to get *optimal* assignments with respect to objective functions. νZ provides a portfolio of approaches for solving linear optimization problems over SMT formulas, MaxSMT, and their combinations. Objective functions are combined as either Pareto fronts, lexicographically, or each objective is optimized independently. We describe usage scenarios of νZ, outline the tool architecture that allows dispatching problems to special purpose solvers, and examine use cases.

1 An Invitation to νZ

νZ extends the functionality of Z3 [7] to include optimization objectives. It allows users to solve SMT constraints and at the same time formulate optimality criteria for the solutions. It relieves users of Z3 from writing their own loops around the solver to find optimal values. The solver integrates state-of-the-art algorithms for optimization, and it extends some of these algorithms with its own twists: For example, it includes direct support for difference logic solvers, it uses Simplex over non-standard numbers to find unbounded constraints, and it applies an incremental version of the MaxRes [11] algorithm for MaxSAT solving.

To give a first idea, we can ask to optimize the term $x+y$ under the constraints $y < 5 \wedge x < 2$ and $y - x < 1$ using the SMT query to the right. The optimal answer is given as 2 and νZ returns a model where $x = y = 1$. The example shows the `maximize` command that is added to the SMT-LIB [13] syntax.

```
(declare-fun x () Int)
(declare-fun y () Int)
(assert (and (< y 5) (< x 2)))
(assert (< (- y x) 1))
(maximize (+ x y))
(check-sat)
(get-model)
```

1.1 Optimization Commands

The full set of commands νZ adds to SMT-LIB are:

C. Baier and C. Tinelli (Eds.): TACAS 2015, LNCS 9035, pp. 194–199, 2015.
DOI: 10.1007/978-3-662-46681-0_14

```
(declare-fun x () Int)
(declare-fun y () Int)
(define-fun a1 () Bool (> x 0))
(define-fun a2 () Bool (< x y))
(assert (=> a2 a1))
(assert-soft a2    :dweight 3.1)
(assert-soft (not a1) :weight 5)
(check-sat)
(get-model)
```

Fig. 1. Maximize $3.1 \cdot a2 + 5 \cdot \overline{a1}$. νZ finds a solution where $y \leq x \leq 0$

```
(declare-fun x () Int)
(declare-fun y () Int)
(assert (= (+ x y) 10))
(assert (>= x 0))
(assert (>= y 0))
(maximize x)
(maximize y)
(set-option :opt.priority box)
(check-sat)
```

Fig. 2. νZ produces two independent optima $x = 10$, respectively $y = 10$

- `(maximize` *t*`)` - instruct the solver to maximize *t*. The type of the term *t* can be either Integer, Real or Bit-vector.
- `(minimize` *t*`)` - instruct the solver to minimize *t*.
- `(assert-soft` F [`:weight` *n* | `:dweight` *d*] [`:id` *id*]`)` - assert soft constraint F, optionally with an integral weight *n* or a decimal weight *d*. If no weight is given, the default weight is 1 (1.0). Decimal and integral weights can be mixed freely. Soft constraints can be furthermore tagged with an optional name *id*. This enables combining multiple different soft objectives. Fig. 1 illustrates a use with soft constraints.

1.2 Combining Objectives

Multiple objectives can be combined using lexicographic, Pareto fronts or as independent box objectives.

Lexicographic Combinations: By default, νZ maximizes objectives t_1, t_2 subject to the constraint F using a lexicographic combination. It finds a model M, such that M satisfies F and the pair $\langle M(t_1), M(t_2)\rangle$ is lexicographically maximal. In other words, there is no model M' of F, such that either $M'(t_1) > M(t_1)$ or $M'(t_1) = M(t_1)$, $M'(t_2) > M(t_2)$.

Pareto Fronts: Again, given two maximization objectives t_1, t_2, the set of Pareto fronts under F are the set of models $M_1, \ldots, M_i, \ldots, M_j, \ldots$, such that either $M_i(t_1) > M_j(t_1)$ or $M_i(t_2) > M_j(t_2)$, and at the same time either $M_i(t_1) < M_j(t_1)$ or $M_i(t_2) < M_j(t_2)$; and for each M_i, there is no M' that dominates M_i. νZ uses the Guided Improvement Algorithm [14] to produce multiple objectives. Fig. 3 illustrates a use where Pareto combination is specified.

Boxes: Box objectives, illustrated in Fig.2 are used to specify independent optima subject to a formula F. They are used in the Symba tool [9]. The box combination of objectives t_1, t_2 requires up to two models M_1, M_2 of F, such that $M_1(t_1)$ is the maximal value of t_1 and $M_2(t_2)$ is the maximal value for t_2.

1.3 Programming Optimization

The optimization features are available over Z3's programmatic APIs for C, C++, Java, .NET, and Python. There is furthermore a library available as an example that plugs into the Microsoft Solver Foundation (MSF). Fig. 3 shows an example using the Python API to generate Pareto optimal solutions. Fig. 4 shows an OML model used by MSF.

```
x, y = Ints('x y')
opt = Optimize()
opt.set(priority='pareto')
opt.add(x + y == 10, x >= 0, y >= 0)
mx = opt.maximize(x)
my = opt.maximize(y)
while opt.check() == sat:
    print mx.value(), my.value()
```

Fig. 3. Pareto optimization in Python. νZ produces all 11 Pareto fronts.

```
Model[
 Decisions[
  Reals[-Infinity, Infinity], xs, xl],
 Constraints[
  limits -> 0 <= xs & 0 <= xl,
  BoxWood -> xs + 3 * xl <= 200,
  Lathe -> 3 * xs + 2 * xl <= 160],
 Goals[
  Maximize[$ -> 5 * xs + 20 * xl]]]
```

Fig. 4. OML model used by MSF

1.4 MILP, MaxSAT, CP and SMT

Efficient mixed integer linear solvers are backbones of several highly tuned tools, such as CPLEX and Gurobi, used in operations research contexts. Being able to state and solve optimization objectives in the context of logical constraints has also been well recognized in the SMT community [12,5,15,8] and it is a recurring feature request for Z3 as well. We briefly outline a use case in Section 4, and through this experience we observed a need for more abstract and flexible ways of modeling problems than exposed by OML used by the Microsoft Solver Foundation (MSF), where flexible Boolean combinations of constraints, which empower end-users to refine models, are afterthoughts. By making νZ generally available, we hope to make it easier for existing users to use Z3, for instance [2], and to fuel further applications that benefit from the flexibility and expressive power of Z3's SMT engines, including theory support and quantifiers, with the convenience of built-in support for (reasonably tuned) optimization algorithms. In return, we anticipate that new applications from SMT users can inspire advances in areas such as non-linear arithmetic, mixed symbolic/numerical algorithms, and combinations with Horn clauses.

1.5 Resources

The full source code of νZ is available with Z3 from `http://z3.codeplex.com`, the sources compile on all main platforms, there is an online tutorial on `http://rise4fun.com/z3opt/tutorial/`, and a companion paper [3] describes details of algorithms used in νZ.

2 Architecture

Fig. 5 gives an architectural overview of νZ. The input SMT formulas and objectives are rewritten and simplified using a custom strategy that detects 0-1 integer variables and rewrites these into Pseudo-Boolean Optimization (PBO) constraints. Objective functions over 0-1 variables are rewritten as MaxSAT problems[1]. If there are multiple objectives, then νZ orchestrates calls into the SMT or SAT cores. For box constraints over reals, νZ combines all linear arithmetic objectives and invokes a single instance of the OptSMT engine; for lexicographic combinations of soft constraints, νZ invokes the MaxSAT engine using multiple calls.

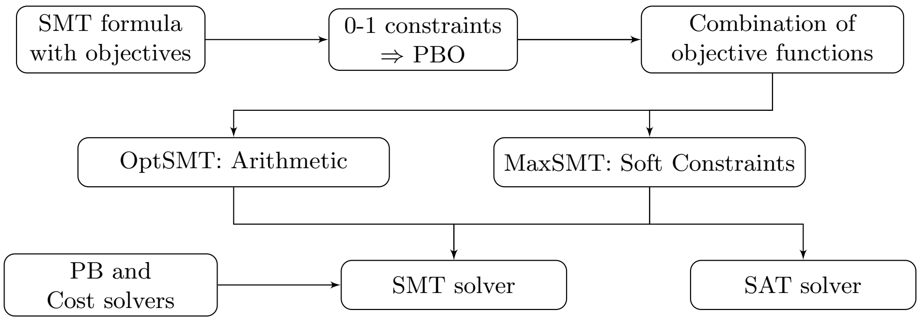

Fig. 5. νZ system architecture

3 Internals

OptSMT: We have augmented Z3's dual Simplex core with a primal phase that finds maximal assignments for reals. It also improves bounds on integers as long as the improvements are integral. It is used, similarly to [15,9], to improve values of objective functions. A similar primal Simplex solver is also accessible to Z3's difference logic engines. νZ discovers unbounded objectives by using non-standard arithmetic: It checks if $t \geq \infty$ is feasible, over the extension field $R \cup \{\epsilon, \infty := 1/\epsilon\}$. This contrasts the approach proposed in [9] that uses a search through hyper-planes extracted from inequalities.

νZ also contains a Pseudo-Boolean theory solver. It borrows from [4,1] for simplification, generating conflict clauses, and incrementally compiling into small sorting circuits. It also adds an option to prune branches using dual simplex.

MaxSMT: νZ implements several engines for MaxSAT. These include WMax [12], MaxRes [11], BCD2 [10], MaxHS [6]. WMax uses a specialized theory solver of *costs*, also explored in [5]. The solver associates penalties with a set of tracked propositional variables. It then monitors the truth assignments to these variables, as given

[1] Using the correspondence: *maximize* $c_1 \cdot x_1 + c_2 \cdot x_2 \equiv$ *(assert-soft* x_1 *:weight* c_1*), (assert-soft* x_2 *:weight* c_2*).*

by the SAT solver. The cost is incremented when a tracked variable is assigned to *false*. The solver creates a conflict clause when the cost exceeds the current optimal value. WMax can be interrupted at any point with a current upper bound. Our implementation of MaxRes generally performs much better than WMax. MaxRes increments a lower bound when there is an unsatisfiable core of the soft constraints. It then replaces the core $F_1, \ldots, F_k$ with new soft constraints $F'_1, F'_2, \ldots, F'_{k-1}$ using the equations:

$$F'_1 = F_2 \vee F_1,\; F'_2 = F_3 \vee (F_1 \wedge F_2),\; \ldots,\; F'_{k-1} = F_k \vee ((F_1 \wedge F_2) \wedge \ldots \wedge F_{k-1}) \;.$$

SAT: νZ reduces Pseudo-Boolean formulas to propositional SAT by converting cardinality constraints using sorting circuits, using a Shannon decomposition (BDDs) of simple PB inequalities and falling back to bit-vector constraints on inequalities where the BDD conversion is too expensive. This transformation is available by ensuring that the option `:opt.enable_sat` is `true`. For benchmarks that can be fully reduced to propositional SAT, MaxRes uses Z3's SAT solver.

4 A Use for νZ

As a driving scenario for νZ we used an experimental warehouse manager in the context of Microsoft Dynamics AX. The objective is to reduce cost by optimizing how shipments are distributed on trucks, reducing the number of trucks, the distance traveled by the truck while maximizing the amount of goods delivered. AX can deliver the standard constraints and cost functions, e.g., weight and volume of a truck, but users often want to be more specific. For example, frozen foods need to be in a cooled truck and cannot be packed together with chemicals. The expressive power and convenience of SMT is useful: these constraints can be formulated as a Boolean combination of linear constraints over 0-1 variables, while the objective functions we considered could be expressed as lexicographic combinations of a couple of cost functions. Such cost functions are expected to evolve when users learn more about their usages. The abstraction layer of the models provides this flexibility.

Table 1. Evaluation of νZ on selected examples

Source	Category	Solved instances	Time
MaxSAT 2014 wpms industrial track	MaxSAT	361/410	0.5-1800s
MaxSAT 2014 pms industrial track	MaxSAT	406/568	0.5-1800s
Longest Paths	MaxSAT	bb 8/8	<0.05s
Longest Paths	MaxSAT	chat 34/34	1-36s
DAL Allocation challenge	PBO	SampleA&B 96/96	0.02-6s
Symba [9]	LRA	2435/2435	0.2s-36s
OptiMathSAT [15]	LRA	9 non-random	0.5-20s

4.1 Experience

We evaluated νZ on a cross-section of benchmarks used in MaxSAT competitions, from Z3 users, and from recent publications. Table 1 summarizes a selected evaluation. Motivating examples from users included strategy scheduling for Vampire (MaxSAT) that are easy with the new MaxSAT engine, but used to be hard for the bisection search used by Vampire. Likewise, Cezary Kaliszyk has used Z3 to tune his portfolio solver using linear arithmetic constraints. His systems are significantly more challenging (take days to run). In this case WMax offers partial solutions during search. Elvira Albert tried using Z3 for finding longest paths, her benchmarks are called bb ($\approx$300 clauses), chat ($\approx$3K clauses) and p2p ($\approx$30K clauses), and we summarize timing for bb and chat below; the p2p category times out.

References

1. Abío, I., Nieuwenhuis, R., Oliveras, A., Carbonell, E.R.: A parametric approach for smaller and better encodings of cardinality constraints. In: CP (2013)
2. Becker, K., Schätz, B., Armbruster, M., Buckl, C.: A formal model for constraint-based deployment calculation and analysis for fault-tolerant systems. In: SEFM, pp. 205–219 (2014)
3. Bjørner, N., Phan, A.-D.: νZ - Maximal Satisfaction with Z3. In: SCSS (2014)
4. Chai, D., Kuehlmann, A.: A fast pseudo-boolean constraint solver. IEEE Trans. on CAD of Integrated Circuits and Systems 24(3), 305–317 (2005)
5. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability modulo the theory of costs: Foundations and applications. In: TACAS (2010)
6. Davies, J., Bacchus, F.: Postponing optimization to speed up MAXSAT solving. In: CP, pp. 247–262 (2013)
7. de Moura, L.M., Bjørner, N.: Z3: An Efficient SMT Solver. In: TACAS (2008)
8. Larraz, D., Nimkar, K., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving Non-termination Using Max-SMT. In: CAV, pp. 779–796 (2014)
9. Li, Y., Albarghouthi, A., Kincaid, Z., Gurfinkel, A., Chechik, M.: Symbolic optimization with SMT solvers. In: POPL, pp. 607–618 (2014)
10. Morgado, A., Heras, F., Marques-Silva, J.: Improvements to Core-Guided Binary Search for MaxSAT. In: SAT, pp. 284–297 (2012)
11. Narodytska, N., Bacchus, F.: Maximum Satisfiability Using Core-Guided MaxSAT Resolution. In: AAAI, pp. 2717–2723 (2014)
12. Nieuwenhuis, R., Oliveras, A.: On SAT Modulo Theories and Optimization Problems. In: SAT, pp. 156–169 (2006)
13. Ranise, S., Tinelli, C.: The SMT Library, SMT-LIB (2006), http://www.SMT-LIB.org
14. Rayside, D., Estler, H.-C., Jackson, D.: The Guided Improvement Algorithm. Technical Report MIT-CSAIL-TR-2009-033. MIT (2009)
15. Sebastiani, R., Tomasi, S.: Optimization in SMT with $\mathcal{LA}(\mathbf{Q})$ Cost Functions. In: IJCAR, pp. 484–498 (2012)