Z3 Projects
Nikolaj Bjørner
Microsoft Research
nbjorner@microsoft.com

Abstract

The document summarizes individual projects using the SMT solver Z3.

1 A solver for UTVPI (Octagons)

1.1 Objective

Develop a stand-alone theory solver for UTVPI and integrate it with Z3.

1.2 Background

UTVPI generalizes Difference Arithmetic. UTVPI admits constraints of the form $\pm x_i \pm x_j \leq k$, where $x_i, x_j$ are variables ranging over reals, respectively integers. $k$ is a numeric constant (a real or an integer). The class is decidable. There is quite a bit of literature on this domain already. We here list a few pointers that can be of interest.

- [http://dl.acm.org/citation.cfm?id=1614551](http://dl.acm.org/citation.cfm?id=1614551)
- Port to a theory solver: [https://bitbucket.org/michal.moskal/fx7/src/867890d35d41/Utvpith.n](https://bitbucket.org/michal.moskal/fx7/src/867890d35d41/Utvpith.n)
- [http://www.cs.berkeley.edu/~sseshia/pubdir/JSAT_3_2_3_Seshia.pdf](http://www.cs.berkeley.edu/~sseshia/pubdir/JSAT_3_2_3_Seshia.pdf)

There are several other references of relevance. The basic UTVPI algorithms have been improved, and there are extensions of UTVPI to min/max functions and modular constraints (but no efficient algorithms for these are known).

1.3 Plan

- Develop (port) a stand-alone solver for UTVPI.
- Integrate it with Z3 as a user-theory. You can leverage the parser support from the API to load benchmarks, but it will be helpful to replace arithmetical sub-formulas by predicates in a format that is understood only by the theory solver (you declare predicates yourself).
- A first implementation checks for consistency in FinalCheck. It only produces theory conflicts and does not attempt to perform theory propagation. You can use a variant of the Floyd-Warshall algorithm to implement this version.
- Can you mine for strong explanations from UTVPI constraints?
- Is the theory convex? Can you extract implied equalities?
- An alternative implementation uses the Ford-Fulkerson algorithm, which can be used to incrementally update a graph.
2 Checking properties of XACML policies

2.1 Objective

XACML policies describe rule-based systems over expressive domains (including dates, priorities). To the extent that we wish to analyze XACML policies, how can SMT solvers be used?

2.2 Background

2.3 Plan

- Encode XACML policies using multi-sorted first-order logic.
- Define admin queries of interest. Examples could be:
  1. Is a policy determined? In other words, is there a query where a policy produces NotApplicable.
  2. Given two policies, produce a query that produces different answers. More generally, for each type of answer difference (example Allow vs. Deny and Deny vs. Allow), produce a query (or determine that there is none) such that the two policies exhibit this difference.
  3. Determine if a rule is redundant.
  4. Can you simplify rules by using satisfiability checks?

3 F⋆⋆

3.1 Objective

Encode separation and higher-order type theory with SMT.

3.2 Background

The F⋆ programming language uses Z3 to check refinement type checking obligations. It is based on foundational type theory and uses affine types for programming with side-effects. The language F⋆ is based on F# and F7 and FINE. Separation logic is designed for handling updates to heaps. Can F⋆ be made better heap aware using separation logic?

3.3 Plan

- Verify a basic example using F⋆.
- Exemplify what can be done using HTT and separation logic and not with F⋆ (if any).
- Exemplify what can be done using F⋆ and not published HTT with separation logic (if any).
- Develop a case study (by hand) where refinement types and affine types interplay with separation logic types.
- (hand-)code the example into SMT-LIB2.
4 Planning and SMT

4.1 Objective

Encode a robot-arm movement problem using SAT and/or SMT.

4.2 Background

Robot-planning is a multi-dimensional problem. A robot arm has several degrees of freedom. Some robot planning tools use specialized A* based algorithms for searching the space of paths for robot arms that avoid obstacles. Can robot arm movement be encoded into SMT theories?

4.3 Plan

The project requires to

1. Develop a model of a simple, but realistic, robot arm with 3 (or 4) degrees of freedom.

2. Create a two dimensional spatial model with obstacles, a starting point and an end-point.

3. Create a three dimensional spatial model with obstacles, a starting point and an end-point.

4. Encode the reachability problem for the robot arm as a path planning problem with a discrete number of steps. For example, the robot arm may be provided 10 steps. Each step moves one of the joints of the robot arm a bounded distance. The arm should be free of the obstacles before and after each step.

5. The purpose is to extract the robot movements whenever there is a set of steps that gets the arm from the start to the goal. These steps can be read off from the satisfying model.

6. Different paths and obstacles can be defined. Z3 will not scale indefinitely. Can you create family of robot movement problems (more smaller steps/more obstacles/longer distance to goal) that measure what is feasible?

7. There are different encoding techniques for the robot planning problem. One technique is to indicate the robot arm state using coordinates over the reals. The coordinates of a joint is then a three dimensional vector \((x, y, z)\), where \(x, y, z\) range over the reals. A second encoding technique can be to discretize the grid, and at this point the coordinates range over bit-vectors. In either case you have to write constraints that take a configuration space of obstacles and ensure that the arms of the robot don’t intersect with any of the obstacles.

5 Probabilistic Optimization

5.1 Objective

Given a propositional formula whose propositional variables are annotated with propabilities over an independent distribution, determine whether the formula is satisfied with probability beyond a threshold \(k \in [0, 1]\).

The objective of this problem is to investigate a theory solver-based approach for solving. It is a variant of weighted MaxSMT.
5.2 Background

The project on Symbolic Stochastic Automata (Section 6) is highly-related, and this problem could be a component in that project.

5.3 Plan

The project involves writing a theory solver that accumulates the sum of probabilities of satisfying assignments. This can be modeled fairly directly after the weighted MaxSMT solver. To prune the search space effectively an additional component, forward pruning, can be helpful and the project is to develop this.

A challenge is to investigate optimizations on top of this scheme. For example you can consider if it is feasible to cut off (or rather postpone) search branches opportunistically when you can determine that the search space covered by the branch cannot be instrumental in reaching the sum. For example, if the current assignment comes with probability $2^{-100}$ and there are 5 propositional variables that have not been assigned, then the contribution of a branch is at most $2^{-95}$. If the objective is to check if the formula has probability at least $\frac{1}{2}$ the search along other assignments will have to contribute much more than the current branch.

6 Symbolic Stochastic Automata

6.1 Objective

Check bounded properties of Markov chains with infinite state.

6.2 Background

Properties of stochastic processes can in some formulations be checked using linear programming techniques. SMT solvers include linear programming solvers for feasibility. Can we leverage SMT solvers to both check the linear programming problems as well as path feasibility for Markov chains?

6.3 Plan

This project does not necessarily require hands on solving, but it would be a definite plus. There are several challenges in just formulating what a reasonable solution is, or whether there are reasonable solutions at all. The first challenge is to define a reasonable notion of symbolic Markov processes, where the set of states are infinite so an encoding of state transformers using SMT techniques is of relevance in the first end. The second challenge is developing the principles of checking bounded properties of Markov processes. The choice of which properties are of relevance is open. You may choose to focus on simulation relations, such as alternating simulation, or on probabilistic guarantees for a bounded number of steps. The common trait is that checking these properties map in a straight-forward way into a bounded unfolding of the Markov transition relation and into an SMT formula. A question is how to formulate the probabilistic transitions in this context. In one variant you may want to take the sum of the feasible paths; what are the options for using SMT solving techniques (eg., would you want to start enumerating paths as models?).
7 From cryptographic primitives to belief logic

7.1 Objective
Formulate a contract library for cryptographic primitives in the context of belief logics.

7.2 Background
Cryptographic primitives can be used to ensure trust that can be formulated in belief logics. Belief logics do not directly use cryptographic primitives. What connections can be made and how can implementations using cryptographic primitives then be checked.

7.3 Plan
1. Define a language of cryptographic library functions for a high-level language with primitives for trust.
2. Axiomatize the connections between cryptographic primitives and belief logics.
3. Define the contracts for the library.
4. Check correctness of the library based on the axiomatization.
5. Check composed belief logic properties of the high-level language.

The formalization can take advantage of encodings in SMT-LIB2.

8 Checking Horn clauses for security: finite models and refutations

8.1 Objective
Evaluate using Z3 for checking Horn clause formulations of security protocols.

8.2 Background
The tool by Sebastian uses back-end theorem provers. Try to evaluate Z3 (and other theorem provers) for proofs and model finding.

8.3 Plan
Use benchmarks from Sebastian’s project. Examine encoding techniques for finite model finding using Z3 and evaluate when model-based quantifier elimination strategies apply.

9 Symbolic Büchi Automata for Workflows

9.1 Objective
Develop automata operations for Symbolic Büchi automata.
9.2 Background

Tijs’s research on workflows relies on a reduction to Büchi automata. We would like to evaluate finding feasible paths in workflows with symbolic constraints.

9.3 Plan

- Define a notion of workflows with symbolic constraints. For example, define workflows that update variables that range over integers or reals.
- Develop an encoding of Symbolic Büchi automata as a theory. This can be achieved using an axiomatic characterization of the transition relation. The alphabet of these automata may be arbitrary formulas as [http://research.microsoft.com/projects/automata](http://research.microsoft.com/projects/automata).
- Check bounded properties using the encoding, meaning synthesize finite counter-examples with a loop.
- You can extend the work quite a bit. For example, symbolic automata may require infinite traces and you will need to incorporate reasoning about well-founded measures (but this is likely out of scope of a shorter project).

10 Answer set programming and SMT

10.1 Objective

Investigate three encodings of ASP into SMT.

10.2 Background

Two encodings of ASP into SMT have been tried. The first uses difference logic, the second uses bit-vectors (or SAT in disguise). I am curious if you can leverage the theory of recursive data-types as well.

10.3 Plan

The idea would be to encode ASP problems similar to the encodings into difference logic, but use the theory of recursive data-types to ensure well-foundedness. The theory solver for recursive data-types performs occurs check and produces only conflict clauses when the occurs check fails.

The project requires to

- Formalize the encoding using recursive data-types and check if it is indeed sound.
- Identify a set of grounded ASP benchmarks.
- Parse and pretty print the benchmarks into SMT-LIB using the three encodings.
- Evaluate the benchmarks on the tree encodings.

11 Symbolic Execution

11.1 Objective

Write a symbolic execution engine for C0.
11.2 Background

The programming language C0 is used at CMU for freshman imperative programming. The last project is to write a virtual machine for C0. Testing and contracts are promoted as part of the course.

11.3 Plan

- Write a virtual machine for C0 and extend it to produce a symbolic trace of the executed program.
- Handle first numeric programs (without heap).
- Once you can generate traces for numeric programs try to handle heap objects. You can take advantage of the strong typing in C0.
- You can alternatively translate C0 into C# and try it with Pex. (C0 uses only type safe constructs).

12

12.1 Objective

12.2 Background

12.3 Plan