KSMT: a Platform for Unified Use of SMT Solvers

V. Sobol

June 14, 2023

Why SMT Solvers?

SMT solvers are used in most software quality assurance tasks

- Verification
- Bugs detection
- Test generation

SMT solvers are used in many program analysis methods

- Symbolic execution
- Bounded model checking
- etc

Why Many SMT Solvers?

There are many SMT solvers: Z3, CVC5, Boolector, Bitwuzla, Yices2, OpenSMT, etc

Why do we need different solvers?

- On different examples, different solvers show different performance (see smt-comp[1])
- Sometimes different solvers produce different results

[1] https://smt-comp.github.io/2022/

Problems when Using Solvers

Most of the SMT solvers are research projects (not production ready)

- Difficulties in production usage
  - Solvers (sometimes) ignore timeouts
  - Solvers (sometimes) suddenly crash causing the entire process interruption, including the user's app
- Solvers native binaries distribution
  - Many solvers don't have prebuilt binaries
  - Usually, binaries are provided for linux platform only

Problems when Using Solvers

Many program analyzers are JVM applications

- Most of solvers provide only native C API or SMT-LIB interface
- C API requires JVM bindings
- SMT-LIB interface is not suitable for use in high-performance applications
  - Requires serialization/parsing of the SMT-LIB
  - Parsing is tricky because the output format of the solvers is actually different
  - Text representation is ineffective
  - Some solvers implement SMT-LIB with errors

Motivating Features

Easy solver delivery

- Solver binaries distribution
- Provide JVM bindings for solver API

Unified SMT solver API

- Easy switching between solvers

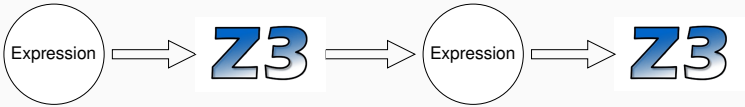
Easy solver management

- Solver portfolio
- Possibility to run solver in separate process

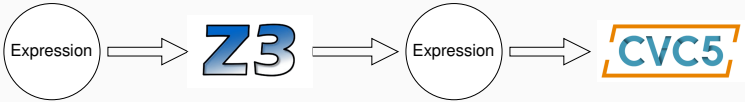
Simple mode

Portfolio mode

Continuous mode



Joint mode



What other features do we need?

Solver agnostic SMT formula representation

- Formula transformations — implemented once, works for any solver
- Required for Joint mode
- Expression can be used after the solver is freed

Various usability features

- Simple and concise DSL for SMT expressions
- Solver configuration API

Why not JavaSMT?

JavaSMT — Unified Java API for SMT solvers

- Provides unified JVM API
- Solves problem with solver binaries distribution

Missing important features

- Handle missed solver timeouts and solver crashes
- Solver agnostic expressions (required for Joint mode)
- Portfolio solver

KSMT

KSMT — universal JVM based platform for SMT solvers with simple configuring and launch management

- Rich expression system
- Unified solver API
- Process runner
- Unified SMT model representation
- Expression simplifier
- Simple and concise DSL for SMT expressions

Unified Solver API

Same API for any supported solver

Provided features

- Check-sat with timeout
- Model generation
- Incremental solving via push/pop
- Incremental solving via assumptions (check-sat-assuming)
- Unsat cores

Currently supported solvers

- Z3, Bitwuzla
- Yices2, CVC5

Rich Expression System

Expressions are solver agnostic

- Expression manipulation without involving a solver

Expression interning (aka hash consing)

- Fast expression comparison — otherwise we need to compare expression trees
- Expression deduplication — equal expressions stored in memory once

Currently supported theories

- Core, Arithmetic, FP
- BV, Arrays, UF
- Quantifiers

Supported theories: Arrays

Multi-indexed arrays: (*Array* $Index_0$ $Index_1$ *Value*)

- Used in program analysis to represent multidimensional arrays
- Natively supported in Z3
- Modeled with uninterpreted functions in other solvers

Array lambdas [2]

- Used in program analysis to represent array copy operations
- Natively supported in Z3
- Modeled with uninterpreted functions in other solvers

[2] https://microsoft.github.io/z3guide/docs/logic/Lambdas

Supported theories: UF

Unintepreted sorts: (declare-sort *S* 0)

- Used in program analysis to represent heap locations
- Natively supported in all solvers

Uninterpreted sort values

- Uninterpeted sort distinct values
- Used in SMT models to represent sort universe
- Natively supported in Yices
- Modeled with uninterpreted constants in other solvers

Rich SMT Model Representation

Solver agnostic

- Same model representation for any solver

Solver independent

- Native SMT model may be erased
  - Some solvers invalidate native model after next check-sat
  - Sometimes solver may crash
  - Sometimes one want to release solver resources
- Model manipulation without solver
  - Model completion — provide default values for free variables
  - Expression evaluation

Expression Simplifier

Features

- Reduce expression size and complexity
- Evaluate (reduce to a constant) expression
- Allows evaluation of expressions with free variables

Simplifier in KSMT

- Implements more than 200 rules
- Used for expression evaluation in solver agnostic model

Process Runner

Safe run solver in separate process

- Strict timeouts by process interruption
- Fail safety — solver crash handling

High performance

- Reusable process pool
- Fast expression binary serialization

Transparent for user

- Provides the same API as any other solver

Portfolio solver

SMT solvers may differ in performance across different formulas

Portfolio solver idea

- Run multiple solvers in parallel on the same formula
- Get the first (the fastest) result

Implemented on top of the process runner

Transparent for user

- Provides the same API as any other solver

KSMT: Current State

Expression system

- Expression interning
- Expression simplification / evaluation
- Supported theories:
  - Core, Arithmetic, FP, BV, Quantifiers
  - Arrays, multi-indexed arrays, array lambdas
  - UF, uninterpreted sorts, uninterpreted sort values

SMT solvers support

- Unified SMT solver interface
- Solver agnostic models
- Solvers: Z3, Bitwuzla, Yices2, CVC5

KSMT: Current State

Concise & elegant Kotlin/Java DSL

```
val expr = (array.select(index - 1.expr) lt value) and
        (array.select(index + 1.expr) gt value)
```

KSMT platform distribution

- Distributed as JVM library
- Solver binaries are provided
- State: production ready
- SMT-LIB benchmarks based test environment
- Documentation & examples are available

Feature Comparison: Theories

	BV	FP	Arith	Array			UF			Quant	Str	DT
				Array	MI	Lambdas	UF	Sort	Values			
KSMT	+	+	+	+	+	+	+	+	+	+	-	-
JavaSMT	+	+	+	+	-	-	+	-	-	+	+	-
pySMT	+	-	+	+	-	-	+	+	-	+	+	-
SMT Kit	+	-	+	+	-	-	+	-	-	+	-	-
Smt-Switch	+	-	+	+	-	-	+	+	-	+	-	+
metaSMT	+	-	-	+	-	-	+	-	-	-	-	-
jSMTLIB	+	-	+	+	-	-	+	+	-	+	-	-
rsmt2	+	-	+	+	-	-	+	+	-	+	-	+
SBV	+	+	+	+	-	+	+	+	-	+	-	+
Scala SMT-LIB	+	-	+	+	-	+	+	+	-	+	+	-
What4	+	+	+	+	-	+	+	+	-	+	+	+

Feature Comparison: Platform Features

	Language	Solver API		Platform features				
				Solver independent		Simplification	Process runner	Portfolio
		Native	SMT-LIB	Expression	Model			
KSMT	Kotlin/Java	+	-	+	+	+	+	+
JavaSMT	Java	+	-	-	-	-	-	-
pySMT	Python	+	+	+	+	+	+	+
SMT Kit	C/C++	+	-	+	-	-	-	-
Smt-Switch	C/C++	+	-	-	-	-	-	+
metaSMT	C/C++	+	+	-	-	-	-	+
jSMTLIB	Java	-	+	-	-	-	+	-
rsmt2	Rust	-	+	-	-	-	+	-
SBV	Haskell	-	+	+	-	-	+	+
Scala SMT-LIB	Scala	-	+	-	-	+	+	-
What4	Haskell	-	+	+	-	+	+	-

KSMT: Future Plans

Expressions

- Quantifier elimination
- Solver proofs
- Support more theories (Strings, Datatypes)
- SymFPU layer (in progress)

SMT solvers support

- Support more solvers: SMTInterpol, Princess, etc
- More effective encoding of complex expressions (lambdas, MI arrays)

KSMT

ksmt.io

Valentyn Sobol

conyashka
vo.sobol @ *mail.ru*