Smten: Automatic Translation of High-level Symbolic Computations into SMT Queries

Richard Uhler (MIT-CSAIL) and Nirav Dave (SRI International)

CAV 2013
Saint Petersburg, Russia

This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237 and supported by National Science Foundation under Grant No. CCF-1217498.

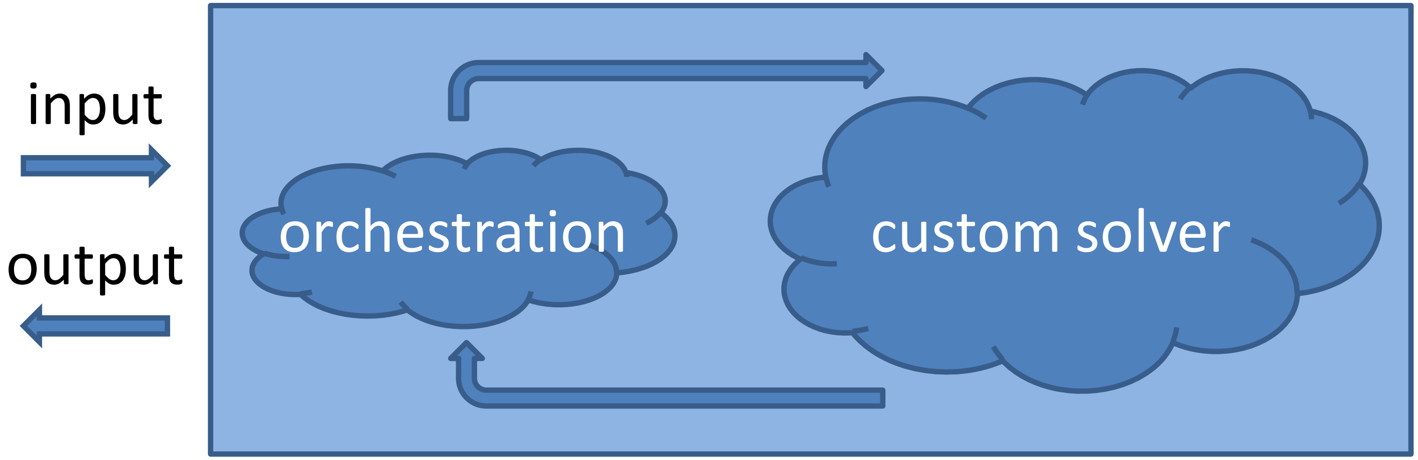
The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views of policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

Motivation: SMT-Based Tools

Satisfiability Modulo Theories (SMT) solvers are well suited for computer aided verification tasks

Uses include:

- model checking
- program synthesis
- automated theorem proving
- automatic test generation
- software verification



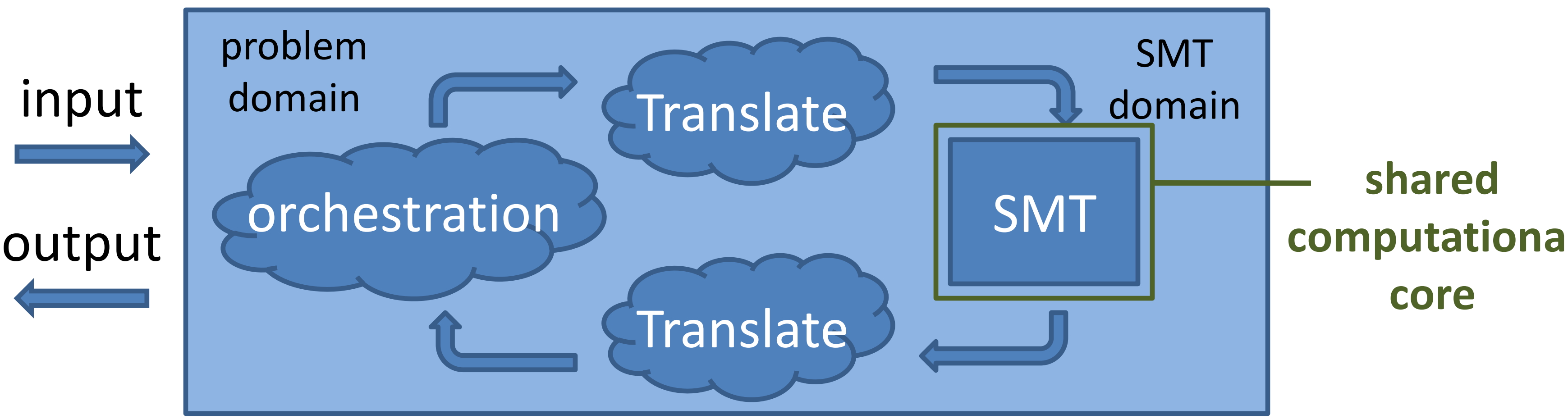
Verification Tool – Before SMT

Motivation: SMT-Based Tools

Satisfiability Modulo Theories (SMT) solvers are well suited for computer aided verification tasks

Uses include:

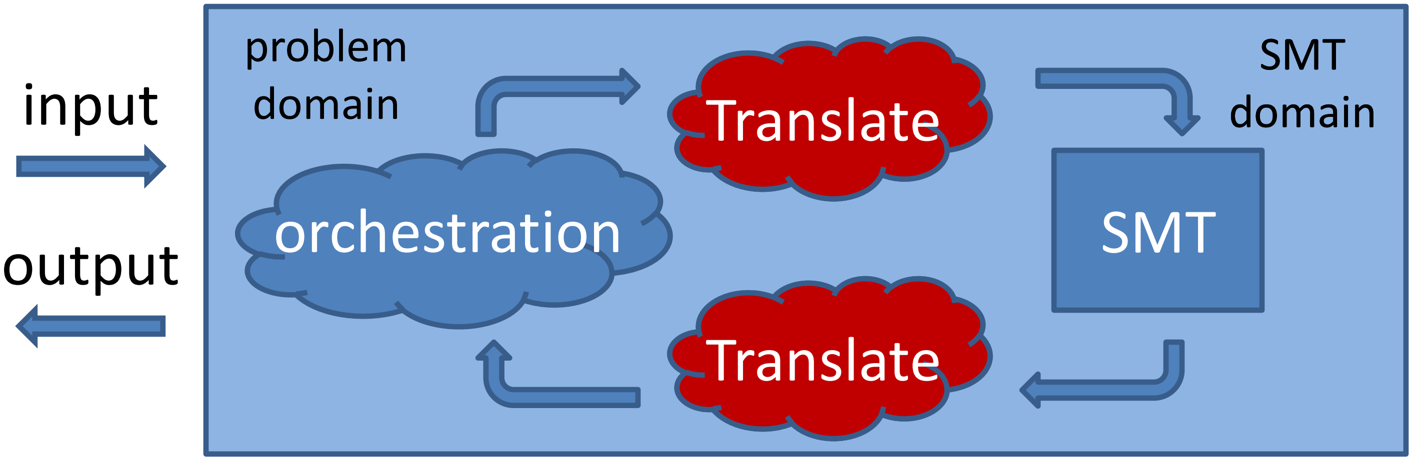
- model checking
- program synthesis
- automated theorem proving
- automatic test generation
- software verification



Verification Tool – With SMT

Efficient Translation with Smten

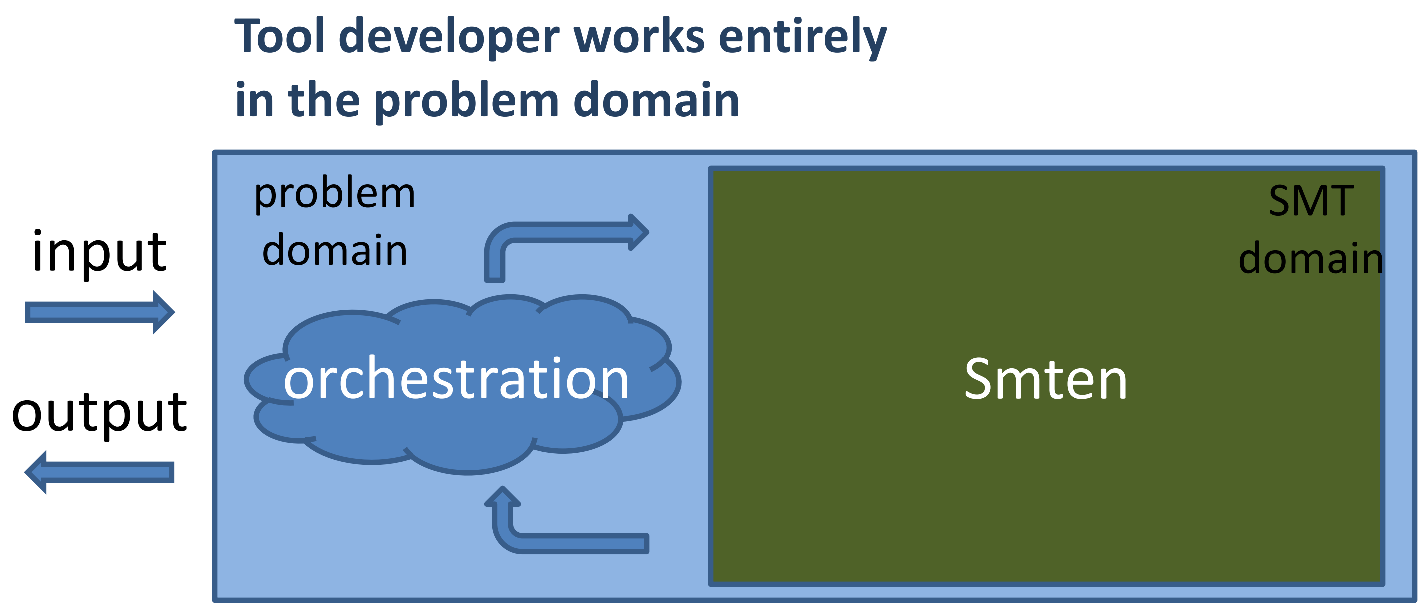
- Implementing efficient translation is tedious and time consuming
- Smten automates the task of translation



Verification Tool – With SMT

Efficient Translation with Smten

- Implementing efficient translation is tedious and time consuming
- Smten automates the task of translation



Verification Tool – With **Smten**

Case Study: Hampi String Solver

Solve for a bounded length variable string which satisfies constraints:

- string contains the given substring
- string belongs to a given regular language

Hampi input:

```
var v : 14 .. 16;
```

bounded length variable string

```
cfg SqlSmall := "SELECT " (Letter)+ " FROM " (Letter)+ " WHERE " Cond;
cfg Cond := Val "=" Val | Cond " OR " Cond;
cfg Val := (Letter)+ | "'" (Ascii)* "'" | (Digit)+;
cfg Digit := ['0'-'9'];
cfg Letter := ['a'-'z'] | ['A'-'Z'];
cfg Ascii := Letter | Digit | " " | "'";
```

regular language specification

```
val q := concat("SELECT msg FROM messages WHERE topicid='", v, "'");
```

```
assert v contains "OR '1'='1'";
```

"contains" constraint

```
assert q in SqlSmall;
```

"membership" constraint

Hampi output: **{VAR(v)=80' OR '1'='1'}**

[Kiezun, et. all ISSTA '09]

Challenges in Translation

Application-level design decisions effect the translation:

- Choice of SMT solver and background theories
  - Determines the target API of translation
- Representation of application-level data structures in SMT domain
  - Represent Hampi symbolic characters using Integers? Bit-vectors? Booleans?
- Decomposition of problem into SMT queries
  - Use single SMT query for an entire Hampi problem?
  - Use a different query for each possible string length?

Making these decisions empirically is tedious and time consuming:

- Translation must be re-implemented for each choice

⇒Implementing efficient translation is a non-trivial amount of work

Optimization in Translation

Even if you know the right design decisions to make, the translation must be optimized

- Hampi example:

  assert **"zb?????d"** **in** **/a(b*cd)*/**

  - This assertion obviously doesn't hold
  - Direct translation to SMT would construct a full-sized SMT query for the assertion
  - Even if the solver can solve this quickly, there is still a non-trivial translation cost
- Another example: preservation of sharing

=> Implementing efficient translation is *hard*

Smten will do these low-level optimizations for you

The Smten Language

High-level, purely functional language, with syntax and features borrowed heavily from Haskell:

- User defined algebraic data types, pattern matching
- User defined functions: higher-order and recursive
- Polymorphism: parametric and ad-hoc (type classes)
- General purpose input/output

Provides an API (based on monads) for orchestrating symbolic computations

No distinction between concrete and symbolic functions

The Hampi Membership Constraint in Smten

```
data RegEx = Epsilon | Empty | Atom Char | Range Char Char
           | Star RegEx | Concat RegEx RegEx | Or RegEx RegEx

match :: RegEx -> [SChar] -> Bool
match Epsilon s          = null s
match Empty _            = False
match (Atom x)  [c]      = toSChar x == c
match (Range l h) [c]    = toSChar l <= c && c <= toSChar h
match r@(Star x) []      = True
match r@(Star x) s       = any (match2 x r) (splits [1..length s] s)
match (Concat a b) s     = any (match2 a b) (splits [0..length s] s)
match (Or a b) s         = match a s || match b s
match _          s       = False

match2 a b (sa, sb)      = match a sa && match b sb

splits ns x              = map (\n -> splitAt n x) ns
```

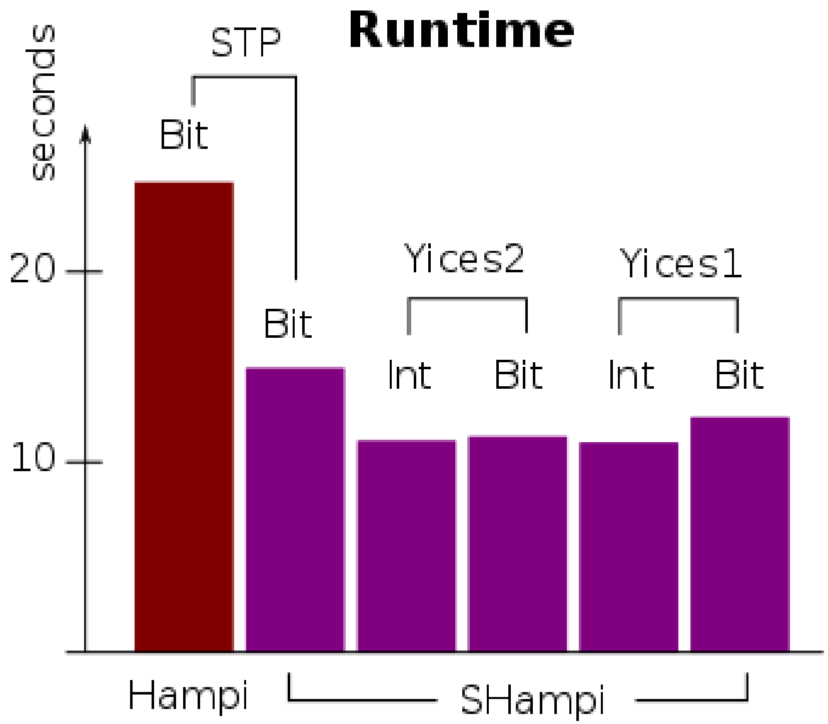
No knowledge of SChar implementation needed

No mention of SMT solver or theories

Can memoize using memo library

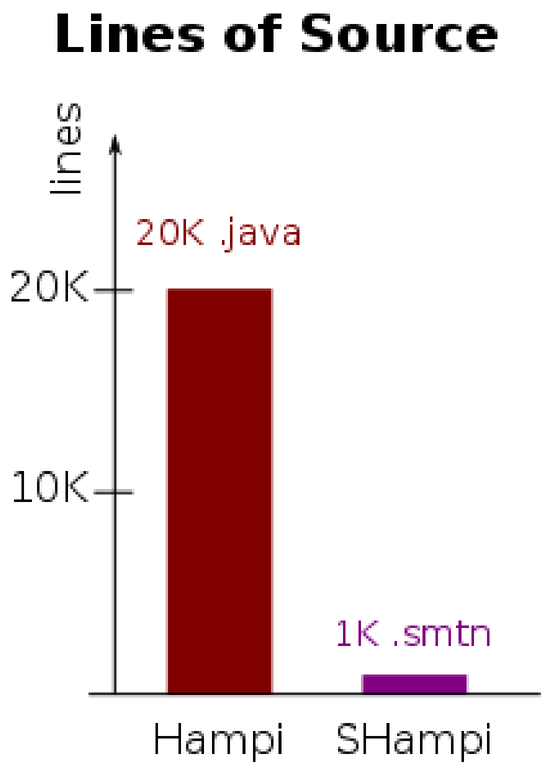
No mention of what is concrete and what symbolic

SHampi: Hampi Implemented with Smten



- SHampi's automatic translation performs just as well as Hampi's manually implemented translation
- Smten allowed to easily explore different SMT solvers and character representations in SHampi

SHampi Development Effort



Component	Lines of Source
SHampi Parser	325
CFG Fix Sizing	100
RegEx Match	56
Query	89
SChar	54

Implemented in just 3 weeks

- Including time to understand Hampi problem
- Including time spent in maturing Smten tool

Conclusion: The Promise of Smten

Smten enables SMT-based tools to share efficient translation

- lowers the barrier to entry in the development of SMT-based tools

Future work:

- Many more optimizations in translation possible:
  - exploit theory of functions
  - implied value concretization
- Supporting infinite symbolic computations
- Libraries of techniques for SMT-based tools
- Generalize portfolio approach of SMT solvers to include background theories

Come see us at the poster session

Richard Uhler

ruhler@csail.mit.edu

Nirav Dave

ndave@csl.sri.com