SAWSCRIPT2: OVERVIEW

BRIAN LEDGER

Abstract

SAWScript is a language for the specification and dispatch of proofs involved in circuit verification and testing.

Introduction

- Refine the type system and syntax to express intuitive and clean data structures
- Analyze workflows and propose optimizations in the syntax and standard library functions
- Provide continuity between versions by identifying essential goals and idioms
- Expose greater functionality with an improved standard library

Notes on the original SAWScript

The original SAWScript was developed ad-hoc during the design verification of a suite of Java, LLVM/C, and Cryptol applications. It aided in the orchestrated dispatch of proofs of equivalence between circuits, as well as the simulated testing of invariants.

One major achievement of SAWScript was to provide a framework for lifting stateful imperative methods in Java and C into stateless, well-typed, and functional specifications, suitable for formal methods verification. This is achieved in part by modeling the contexts which affect and are affected by a method's execution, and then restating the method arguments and type to encompass these parameters.

SAWScript2: Core Concepts

Note: Core Concepts are not necessarily final a.t.m., but represent ideas which have received the most interest and attention.

- **Bitfield Primitives**

  Like Cryptol, SAWScript exposes arrays of bits and the functions between them as first class objects. Distinguishing finite sets of bits allows the specification of properties that, in the context of streams or "infinite" bitfields, would be formally undecidable.

- **Atoms**

  Unlike variables, atoms assume their syntactic names as an intrinsic property, allowing programmatic reflection and manipulation. This is suitable for the concise specification of Java/LLVM contexts, wherein the name of a particular atom is indicative of it's role in the Java and LLVM simulators.

  An atom declared without a binding or type-annotation is automatically assigned the "don't care" value (written explicitly as the underscore _) and a bitfield type of **unknown**. In other words,

  ```
  myAtom;
  ```

  is a shorthand for the declaration

  ```
  myAtom = _ : [?];
  ```

- **Bindings**

  A binding is no more than the assignment to a variable, via the prefix

  `x = ...`

  Any well-typed object can be assigned to a binding.

- **Bags**

  A bag is a heterogeneous collection of well-typed objects, separated by semi-colons, and delimited with braces. Bags are a useful generic type, from which objects as various as contexts, proof-strategies, messages, and records can be derived.

  Example

```
methodInputContext = {
  this.xBuf :: [8][8];
  this.W :: [80][64];
  this.H1; this.H2; this.H3; this.H4; this.H5; this.H6; this.H7; this.H8 }

> methodInputContext :: { [8][8]; [80][64]; [?]; [?]; [?]; [?]; [?]; [?]; [?]; [?] }

proofStrategy = { rewrite; yices }

> proofStrategy :: { rule }
```

- **Standard Libraries**
  - makeContext
  - extractAIG, extractBTOR, extractJava, extractCryptol, extractLLVM, ...

SAWScript2: Proposed Concepts

- Compositional Proofs (**JL** suggested)
- Concurrent (Asynchronous?) Dispatch (**JL** suggested)
- Rewrite Rules (complex, increased scope, needs more discussion)
- Recursive Algebraic Datatypes
- Enumerations

SAWScript2: Other Ideas

- Decidable Checking of Bit-Width Arithmetic (bears investigation, potentially very rewarding)
- Type Synonyms
- Module System and Imports (goes well with Standard Libraries)

Functions and Function Composition