

Status Report for Week 3

Phillip Casey
19 April 2009

Accomplishments for the Week

- Completed Non-Null Types and sample programs
- Began Exception Paradigm sample programs
- Continued research and reading surrounding Spec#, Boogie, etc...

Goals for Next Week

- Exception Paradigm sample programs
- Begin Method Specification sample programs

Additions to Annotated Bibliography

Leino, Rustan, and Rummer, Philipp. The Boogie2 Type System: Design and Verification Condition Generator. 2009 Draft. Available from <http://research.microsoft.com/en-us/um/people/leino/papers/krml186.pdf>; accessed 19 April 2009.

Explains the advantages and the need for an Intermediate Language specification. This paper explains the simple translation and the need for richer typed intermediate languages to match more complicated (and more common) object oriented programs. Uses Boogie2 as an example type system, showing how to translate its expressions into first order formulas suitable for theorem provers. Provides benchmarks, samples, and early experimental results. Although still in draft format, this paper begins to outline the theoretical approach for intermediate languages using a concrete example of the Boogie2 language.

Notes from the Week

New Learning's

Intermediate Language

Program Verifier design crosses many domains; which makes the task complicated; to make the task manageable, break the problem down into smaller, more manageable pieces: one such piece applied to program verifiers is an intermediate language.

Translate intermediate language into various targets:

- SMT solvers
- Simplify

- Spec# Program Verifier

Help bridge the gap between the world of programming languages and the world of logics, understood by theorem provers.

Boogie2

Boogie2 is the successor of BoogiePL. Features:

- an actual type system
- polymorphic maps
- higher-rank polymorphism

What is a Boogie2 Program?

A set of declarations; categorized by two types:

- Imperative Declarations: often referred to as implementation; they are transformed into expressions. Some examples:
 - Variable introduction
 - Procedure Specification
 - Partial Command
- Mathematical Declarations: define first-order structures which can be used in expressions and declarations

Type Declarations

“A non-empty set of individuals, and the sets denoted by different types are disjoint”

Basic Boolean types, custom user types, and macros, of sort

Constants and Functions

Constants of any type

Functions over types. Can take multiple parameters. Parameters are type inferred; with coercion expressions to remove ambiguity.

Maps

Maps *are* expressions (unlike functions which can appear in an expression applied to arguments). This means variables can have maps.

Equality Expressions

Comparison of equality amongst two objects. Simple example:

$$(\forall \langle \alpha \rangle o: Ref, f: Field \alpha \bullet Heap[o, f] = \text{old}(Heap[o, f]) \vee (o = p \wedge f = age) \vee (o = p \wedge f = isMarried))$$

Set of rules that maps and assigns types to variables.

Matching Triggers

Common way for SMT Solvers to handle universal quantification is to selectively instantiate the qualifier. Instantiations are based on matching triggers which determine the patterns that give rise to instantiations

Translations

Two ways to translate Boogie types expressions into equivalent simply typed expressions:

- Type (Logical) Guards: a way to capture type information
- Type Arguments (Encoding): encode type parameters as ordinary arguments