

Verified Software Initiative Benchmarks in Dafny

Rosemary Monahan, NUIM
Joint work with Rustan Leino, MSR

JML Spec-a-thon, 19 November 2009



Dafny

- ▶ experimental language
- ▶ sequential, object based (no subclassing)
- ▶ specifications in the style of *dynamic frames*
- ▶ coarse-grained frames (at the level of whole objects, not individual memory locations)

- ▶ available as open source:
<http://boogie.codeplex.com>

Motivation from VSTTE 2008:

Incremental Benchmarks for Software Verification Tools and Techniques

Bruce W. Weide, Murali Sitaraman, Heather K. Harton, Bruce Adcock, Paolo Bucci, Derek Bronis, Wayne D. Heym, Jason Kirschenbaum, David Frazier

{weide,adcockb,bucci,bronish,heym,kirschen}@cs.e.ohio-state.edu

{murali,hkeown,dfrazie}@cs.clemson.edu



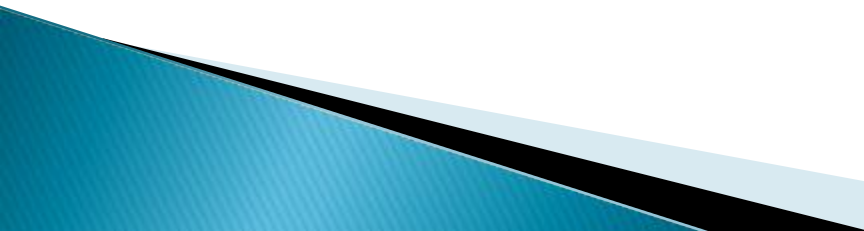
The Benchmarks

“An initial catalogue of easy-to-state, relatively simple, and incrementally more and more challenging benchmark problems for the Verified Software Initiative.”


Their objectives:

- ▶ support assessment of verification tools
- ▶ support assessment of techniques to prove total correctness of functionality software.
- ▶ evaluate the state-of-the art and the pace of progress toward verified software in the near term
- ▶ allow researchers to illustrate and explain how proposed tools and techniques deal with known pitfalls and well-understood issues, as well as how they can be used to discover and attack new ones.

Proposed solutions should include:

- ▶ all formal specifications relevant to the benchmark problem requirements
 - including mathematical definitions, theories, and similar artefacts developed for and/or used in the specifications
 - ▶ all code subjected to the verification process
 - ▶ all verification conditions involved in the verification process
 - ▶ descriptions of the verification system proof rules employed, tools used, and techniques applied.
- 

The proposed solution should:


- ▶ involve both an automatic proof of total correctness of a correct solution, and evidence that the tools and techniques can automatically detect that a “slightly” incorrect solution is incorrect
 - ▶ be modular
 - ▶ be submitted formally to the VSI repository.
- 

Our attempts in Dafny

- ▶ [See http://boogie.codeplex.com](http://boogie.codeplex.com)
- ▶ Go to the source code tab
- ▶ Browse the Boogie source code in the Test/VSI Benchmarks folder

Benchmark #1: Adding and Multiplying Numbers

Problem Requirements:

- ▶ Verify an operation that adds two numbers by repeated incrementing.
 - ▶ Verify an operation that multiplies two numbers by repeated addition, using the first operation to do the addition. Make one algorithm iterative, the other recursive.
- 

Comments:

- ▶ We don't consider overflow.
- ▶ We don't verify that the recursion terminates
– not supported in Dafny.

Benchmark #2: Binary Search in an Array


Problem Requirements:

- ▶ Verify an operation that uses binary search to find a given entry in an array of entries that are in sorted order.

In Dafny:


- ▶ Needed to implement arrays as Dafny does not support them

Comments:

- ▶ Overflow: could have overflow issues.
`var mid := low + (high - low) / 2;`
 - ▶ Fixed a bug in the well-formedness of functions. In particular, it didn't look at the requires clause (in the proper way).
 - ▶ Needed to Implement arrays as Dafny does not provide them
- 

Benchmark #3: Sorting a Queue

Problem Requirements:

- ▶ Specify a user-defined FIFO queue ADT that is generic (i.e., parameterized by the type of entries in a queue).
 - ▶ Verify an operation that uses this component to sort the entries in a queue into some client-defined order.
- 

Benchmark #3: Sorting a Queue


In Dafny:

- ▶ We used integers instead of a generic Comparable type
 - because Dafny has no way of saying that the Comparable type's AtMost function is total and transitive.
- ▶ To prove properties of sequences in Dafny we needed
 - to supply two lemmas to assist the verifier
 - a complicated assignment to pperm
 - to write invariants over p & perm rather than pperm
 - couldn't use “x in p”

Comments

- ▶ We used integers instead of a generic Comparable type, because Dafny has no way of saying that the Comparable type's AtMost function is total and transitive.
- ▶ Tried changing the queue to be generic i.e. `Queue<T>` . This won't verify as when we instantiate the queue `<int>` the translation process generates errors.
- ▶ Would need to pass in the type, the comparison operator and specify the transitive and reflective properties if we were to make this method more generic

Comments:


- ▶ Notation: we couldn't use "x in p".
 - ▶ We couldn't get things to work out if we used the Get method. Instead, we used .contents.
 - ▶ Due to infelicities of the Dafny sequence treatment, we needed to supply two lemmas, do a complicated assignment of pperm, had to write invariants over p and perm rather than pperm
 - ▶ Ghost variables would be nice e.g. pperm is a spec only variable but we cant mark it so.
- 

Benchmark #4: Layered Implementation of a Map ADT

Problem Requirements:

- ▶ Verify an implementation of a generic map ADT, where the data representation is layered on other built-in types and/or ADTs.

Comments:

- ▶ Used sequences of Keys and Values using indices into these sequences to define the mapping
 - ▶ Used built-in equality to compare keys
 - ▶ Can we make this more efficient?
- 

Benchmark #5: Linked-List Implementation of a Queue ADT

Problem Requirements:

Verify an implementation of the queue type specified for benchmark #3, using a linked data structure for the representation.

In Dafny:

Implemented as a set of `Node<T>`

Benchmark #6: Iterators


Problem Requirements:

- ▶ Verify a client program that uses an iterator for some collection type, as well as an implementation of the iterator.

In Dafny:


- ▶ Wrote a collection class as a `seq<int>`
- ▶ Wrote an iterator class
- ▶ Used the iterator to iterate over the collection, storing the elements in a new sequence and verified that the iterator returns the correct things

Comments:

- ▶ Does the iterator destroy the structure that it iterates over? Not in our case.
 - ▶ Could make this harder by requiring the specification and implementation to catch errors if we
 - iterate with one iterator, change the collection and iterate again
 - have two iterators on the same collection
- 

Benchmark #7: Input/Output Streams


Problem Requirements:

- ▶ Specify simple input and output capabilities such as character input streams and output streams
 - ▶ Verify an application program that uses them in conjunction with one of the components from the earlier benchmarks.
- 

Comments:

- ▶ Implemented a stream as a `seq<int>`
- ▶ Methods to :
 - Create a stream from writing
 - Open a stream for reading
 - `PutChar` to write a “Char” // int
 - `GetChar` to read a “Char” // int
 - Check if `AtEndOfStream`
 - Close a stream
- ▶ Client program reads in characters, stores them on Queue (from BM3), sorts them and writes them to a stream

Comments:

- We assume finite streams.
 - If we are required to prove termination then we would need some way of signalling the end of stream
 - What else can we specify? We use the input sequence, sorting and the output sequence correctly but we say nothing about the output that we produce.
- 

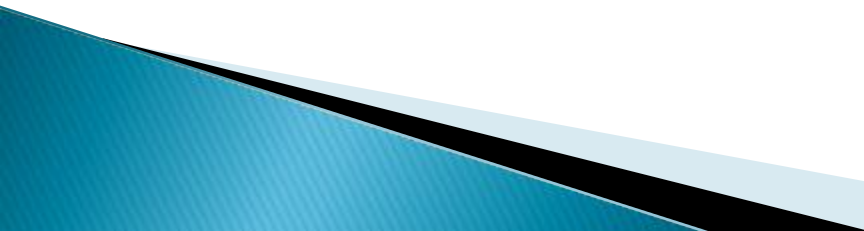
Benchmark #8:

An Integrated Application

Problem Requirements:

- ▶ Verify an application program with a concisely stated set of requirements, where the particular solution relies on integration of at least a few of the previous benchmarks.
- ▶ For example, verify an application program that does the following:
Given input containing a series (in arbitrary order) of terms and their definitions, output an HTML glossary that presents all the terms and their definitions, with (a) the terms in alphabetical order, and (b) a hyperlink from each term that occurs in any definition to that term's location in the glossary.

Our Example:

- ▶ A dictionary is a mapping between words and sequences of words
 - ▶ To set up the dictionary in main we will read a stream of words and put them into the mapping – the first element of the stream is the term, the following words (until we read null) form the terms definition. Then the stream provides the next term etc.
 - ▶ Use the sort method (defined on queue) to sort the words into alphabetical order
- 

Some other Improvements...

- ▶ The Dafny call statement now automatically declares left-hand sides as local variables, if they were not already local variables.
- ▶ Introduced operator `!in` in Dafny. An expression `"x !in S"` is equivalent to `"!(x in S)"`.
- ▶ Redesigned the encoding of Dafny generics, including the built-in types `set` and `seq` (see `Boogie/Binaries/DafnyPrelude.bpl`)
- ▶ Added a sequence update expression
- ▶ Add multisets...

Conclusions:

- ▶ A valuable exercise!
 - ▶ Explores the strengths and weaknesses of tools/languages.
 - ▶ Helps in improving syntax and in determining what language features need to be supported.
 - ▶ Highlights issues with verification e.g. Translation/triggering.
 - ▶ Provides a mechanism for the comparison of languages and tools.
 - ▶ Should lead to improved benchmarks for verification tools.
- 