TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

The Dafny and Boogie Verification Tools

Software-Engineering Seminar, Winter 2017/18

Philipp Schepper

FB Informatik
TU Kaiserslautern

January 11, 2018

Content

Motivation

A correct implementation of the recursive insertion sort?

```
1  method insertionSort(a: array<int>, b: int) {
2    if (b = 1) { return; }
3    insertionSort(a, b-1);
4    var j: int := b-1;
5    while j > 0 ∧ (a[j] < a[j-1]) {
6      swap (a, j-1, j);
7      j := j - 1;
8    }
9  }
```

Motivation

- Writing correct algorithms = Writing + Verification
- Traditionally shown by proof with pen and paper
- Interactive proof-systems simplified process
- SMT-solvers allow automatic verification

Motivation

- Writing correct algorithms = Writing + Verification
- Traditionally shown by proof with pen and paper
- Interactive proof-systems simplified process
- SMT-solvers allow automatic verification

Desirable features:

- Completeness (*i.e.* find all errors)
- Abstraction (algebraic datatypes, side-effect free methods)
- Unrestricted quantifier support
- Termination metrics
- Verified standard libraries

Developed tools (*e.g.* ESC/Java, Spec#) miss some functionalities.

Dafny

- Imperative, class-based language
- Identically named, automatic verifier
- Uses Boogie as underlying system
- Just-in-time verification by using appropriate tools

Dafny

- Imperative, class-based language
- Identically named, automatic verifier
- Uses Boogie as underlying system
- Just-in-time verification by using appropriate tools

Language features:

- No syntactic sugar (*e.g.* do-while loops)
- Pre- and postconditions, predicates
- Loop invariants, termination metrics
- Ghost variables, functions, assertions
- Integrated datatypes, user defined datatypes, custom classes
- *No* libraries

Boogie

- Unstructured, non-deterministic, intermediate language
- Automatic verifier

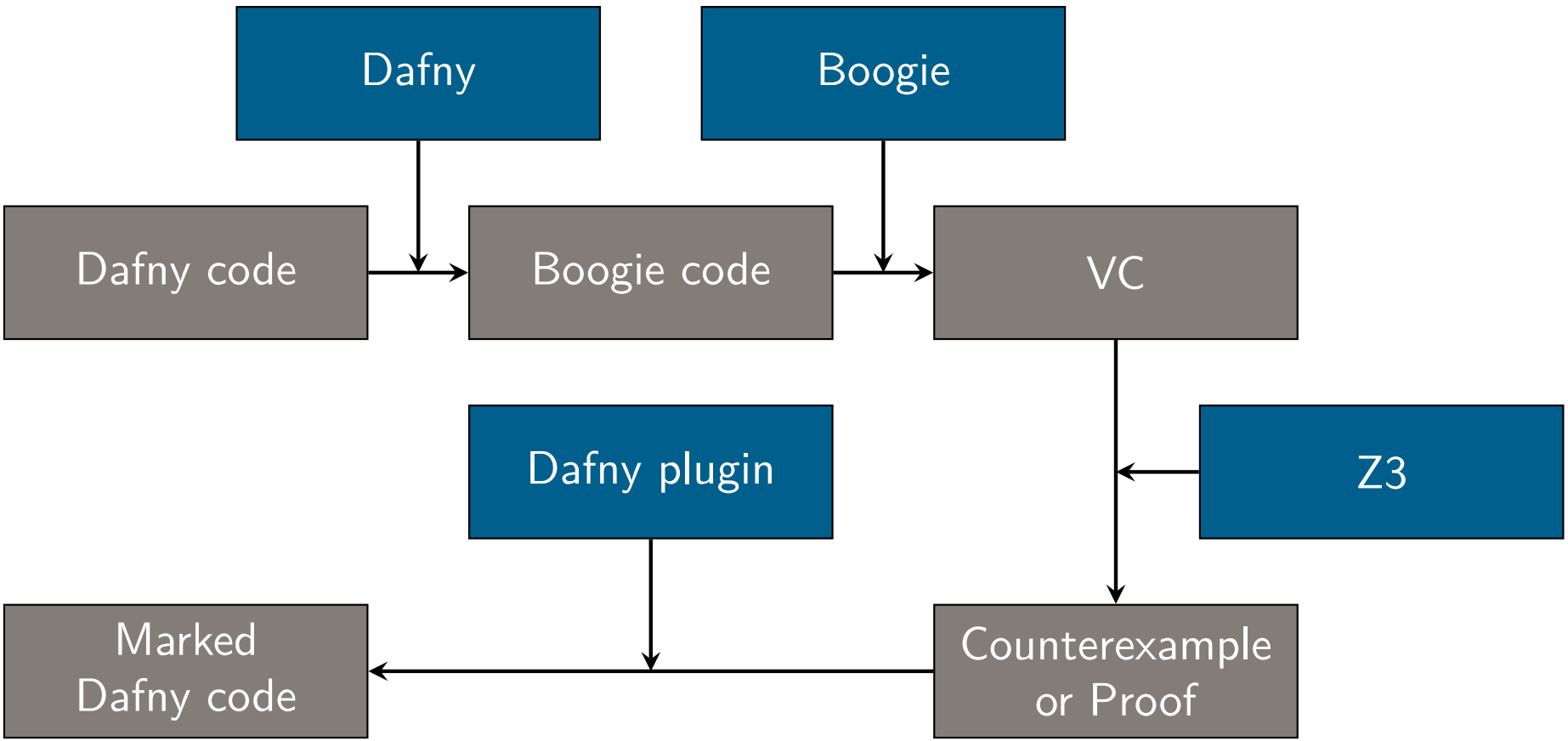
Boogie

- Unstructured, non-deterministic, intermediate language
- Automatic verifier

Language features:

- Similar annotations as in Dafny
- Implementations and declarations for methods
- `goto` with labels
- `havoc` commands and `where` clauses
- Explicit Heap
- No call-by-reference parameter passing
- Axioms for language features

Interaction of the Tools



Dafny – Conclusion

The Good

- Slim, Easy to use
- Design-time-feedback
- Finds all errors

Dafny – Conclusion

The Good

- Slim, Easy to use
- Design-time-feedback
- Finds all errors

The Bad

- Spurious counterexamples
- No inheritance and sub-typing for classes

Dafny – Conclusion

The Good

- Slim, Easy to use
- Design-time-feedback
- Finds all errors

The Bad

- Spurious counterexamples
- No inheritance and sub-typing for classes

The Ugly

- No true invariants for objects
- Boogie knowledge required (*e.g.* error messages, counterexamples)

Back to the Start

A correct implementation of the recursive insertion sort?

```
1  method insertionSort(a: array<int>, b: int) {
2    if (b = 1) { return; }
3    insertionSort(a, b-1);
4    var j: int := b-1;
5    while j > 0 ∧ (a[j] < a[j-1]) {
6      swap (a, j-1, j);
7      j := j - 1;
8    }
9  }
```

```
1  method insertionSort(a: array<int>, b: int)
2
3
4
5
6                {
7    if (b = 1) { return; }
8    insertionSort(a, b-1);
9    var j: int := b-1;
10   while j > 0 ∧ (a[j] < a[j-1])
11
12
13
14                   {
15     swap (a, j-1, j);
16     j := j - 1;
17   }
18 }
```

```
1  method insertionSort(a: array<int>, b: int)
2   requires a ≠ null
3   requires a.Length ≥ b > 0
4   modifies a
5   ensures sorted(a, 0, b)
6                 {
7    if (b = 1) { return; }
8    insertionSort(a, b-1);
9    var j: int := b-1;
10   while j > 0 ∧ (a[j] < a[j-1])
11
12
13
14                    {
15      swap (a, j-1, j);
16      j := j - 1;
17   }
18 }
```

```
1  method insertionSort(a: array<int>, b: int)
2   requires a ≠ null
3   requires a.Length ≥ b > 0
4   modifies a
5   ensures sorted(a, 0, b)
6                 {
7    if (b = 1) { return; }
8    insertionSort(a, b-1);
9    var j: int := b-1;
10   while j > 0 ∧ (a[j] < a[j-1])
11    invariant b ≥ j ≥ 0
12    invariant sorted(a, 0, j) ∧ sorted(a, j, b)
13    invariant ∀ k,l • 0 ≤ k < j ∧ j < l < b ⟹ a[k] ≤ a[l]
14                    {
15     swap (a, j-1, j);
16     j := j - 1;
17   }
18 }
```

```
1  method insertionSort(a: array<int>, b: int)
2   requires a ≠ null
3   requires a.Length ≥ b > 0
4   modifies a
5   ensures sorted(a, 0, b)
6   decreases b {
7    if (b = 1) { return; }
8    insertionSort(a, b-1);
9    var j: int := b-1;
10   while j > 0 ∧ (a[j] < a[j-1])
11    invariant b ≥ j ≥ 0
12    invariant sorted(a, 0, j) ∧ sorted(a, j, b)
13    invariant ∀ k,l • 0 ≤ k < j ∧ j < l < b ⟹ a[k] ≤ a[l]
14    decreases j-1 {
15     swap (a, j-1, j);
16     j := j - 1;
17   }
18 }
```

```
1  method insertionSort(a: array<int>, b: int)
2   requires a ≠ null
3   requires a.Length ≥ b > 0
4   modifies a
5   ensures sorted(a, 0, b)
6   decreases b {
7    if (b = 1) { return; }
8    insertionSort(a, b-1);
9    var j: int := b-1;
10   while j > 0 ∧ (a[j] < a[j-1])
11    invariant b ≥ j ≥ 0
12    invariant sorted(a, 0, j) ∧ sorted(a, j, b)
13    invariant ∀ k,l • 0 ≤ k < j ∧ j < l < b ⟹ a[k] ≤ a[l]
14    decreases j-1 {
15     swap (a, j-1, j);
16     j := j - 1;
17   }
18 }
```

References

K. Rustan M. Leino. *Dafny: An Automatic Program Verifier for Functional Correctness*, pages 348–370.
Springer Berlin Heidelberg, Berlin, Heidelberg, 2010 Mike Barnett,
Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs, and K. Rustan M. Leino.

Boogie: A Modular Reusable Verifier for Object-Oriented Programs, pages 364–387.
Springer Berlin Heidelberg, Berlin, Heidelberg, 2006 Microsoft Research.

Getting started with dafny: A guide.
https://rise4fun.com/Dafny/tutorial/Guide.
Last checked: November 1., 2017