Dafny: An Automatic Program Verifier for Functional Correctness

Paper by K. Rustan M. Leino (Microsoft Research)

Presentation by Patrick Spettel

Seminar: Research Topics in Software Engineering, Spring 2013

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Motivation

- „We envision a world in which **computer programmers** make **no more mistakes** than other professionals, a world in which **computer programs** are always the **most reliable components** of any system or device.“
  - (Hoare et al., 2007, p. 4)

Contribution

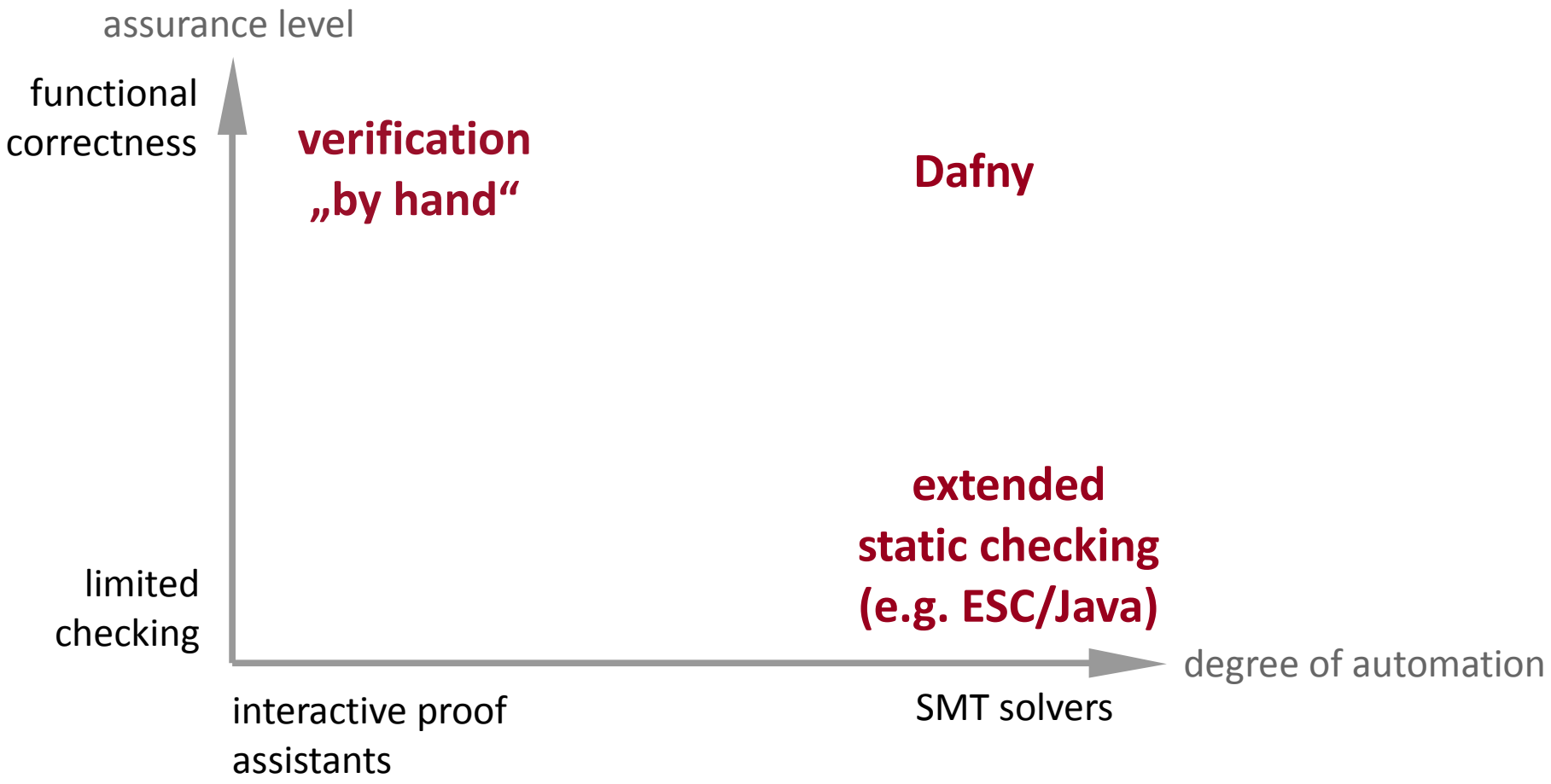


Chart inspired by Leino (2010)

Contribution (cont.)

- **Full verification of functional correctness**
  - Language designed to support the static verification of programs
    - *Influenced by the Euclid language*
  - Total correctness: Dafny always tries to prove termination
  - Automatic verification
  - IDE integration

```
function Sum(): int
{
  if next == null then data else data + next.Sum()
}
```

insufficient reads clause to read field

Language Features

- Dafny is an **imperative, class-based** language
  - Methods
  - Variables
  - Types
  - Type parameters (“Generics”)
  - Loops
  - If statements
  - Arrays
  - ...
  - **No** support for **subclasses** and **no constructors**
- Influenced by **Java** and **C#**

Language Features (cont.)

- **Pre- and Post-conditions**
  - Influenced by Eiffel (Design by Contract)
- **Loop invariants**
- **Termination metrics (variants)**
- **Framing**
  - Influenced by Dynamic Frames (Kassios, 2006)
    - *Remark: Dafny had started as an experiment to encode dynamic frames*
- **(Mathematical) Functions**
- **Algebraic Datatypes**
  - Influenced by ML
- **Lemmas**

Dafny idioms for object invariants and dynamic frames

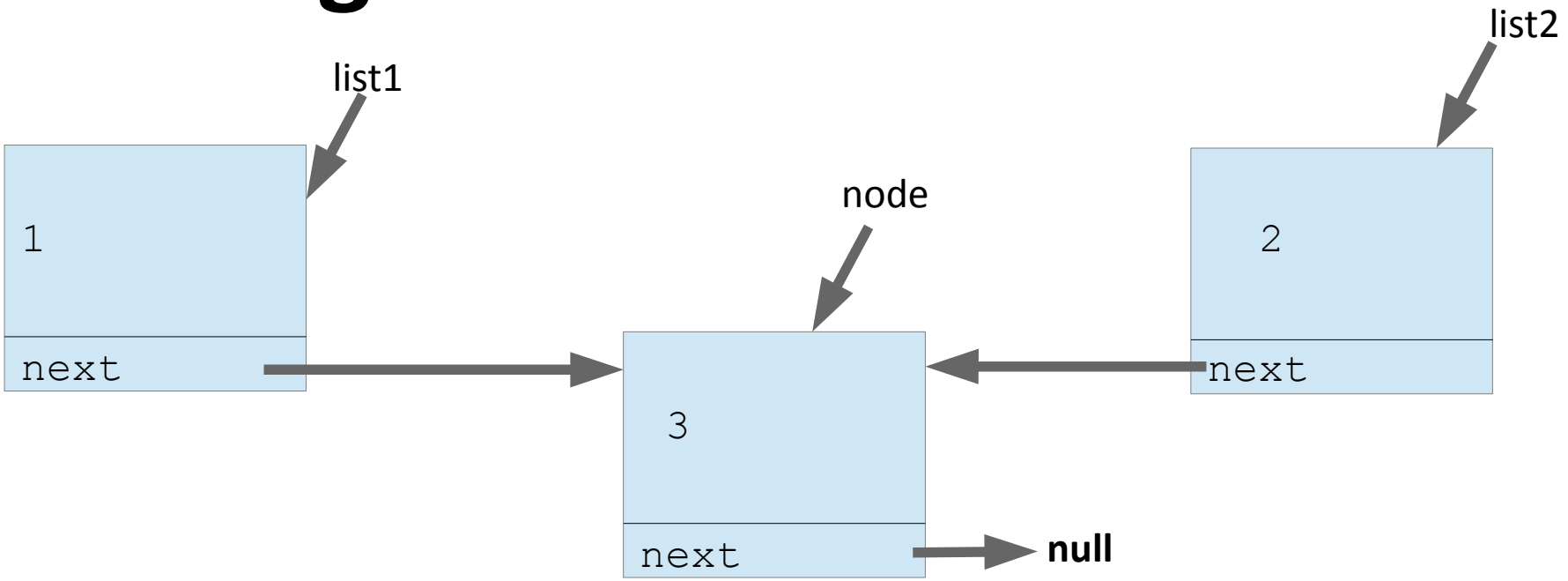
```
class C {
    ghost var Repr : set<object>; // explicit frame

    function Valid() : bool
        reads {this} + Repr;
    { this in Repr && ... }

    method Init()
        modifies {this};
        ensures Valid() && fresh(Repr - {this});
    { ... Repr := {this} + ...; }

    method Update()
        requires Valid();
        modifies Repr;
        ensures Valid() && fresh(Repr - old(Repr));
    { ... }
    ...
}
```

Aliasing




```
var node := new Node<int>; node1.Init(3);
var list1 := node.Prepend(1);
var list2 := node.Prepend(2);

list1.Fill(1); // sets every node's value to the given one
assert(list1.Filled(1));

assert(list2.Valid()); // ??
```

Example

```
class Node<T> {
    ghost var list : seq<T>;
    ghost var Repr : set<object>;

    var data : T;
    var next : Node<T>;

    function Valid() : bool
        reads this, Repr;
    {
        this in Repr && !(null in Repr) &&
        (next == null ==> list == [data]) &&
        (next != null ==>
               next in Repr && next.Repr <= Repr &&
               !(this in next.Repr) &&
               list == [data] + next.list &&
               next.Valid())
    }
...
```

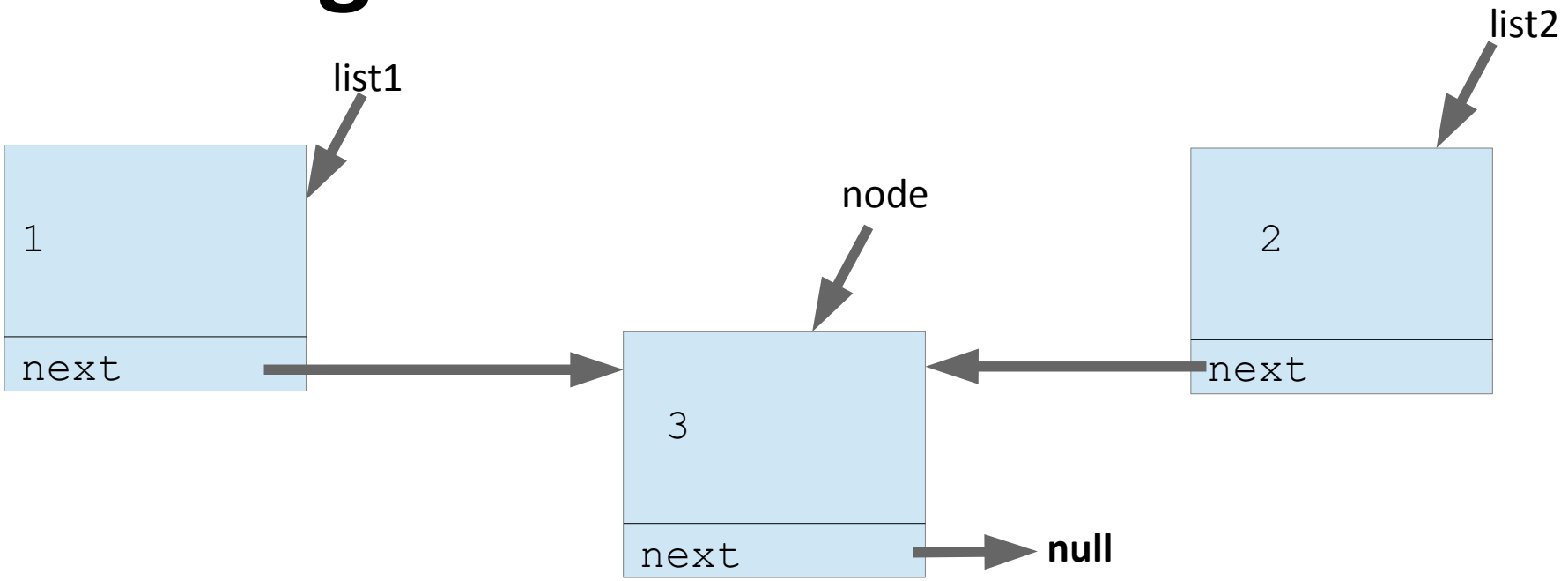
Example (cont.)

```
...
    function Filled(d : T) : bool
        reads Repr;
        requires Valid();
        ensures Valid();
    { forall k :: 0 <= k < |list| ==> list[k] == d }

    method Fill(d : T)
        modifies Repr;
        requires Valid();
        ensures Repr==old(Repr) && Valid() && Filled(d);
        decreases |list|;
    {
        data := d;

        if (next != null) {
            next.Fill(data);
            list := [data] + next.list;
        } else {
            list := [data];
        }
    }
}
```

Aliasing




```
var node := new Node<int>; node1.Init(3);
var list1 := node.Prepend(1);
var list2 := node.Prepend(2);

list1.Fill(1); // sets every node's value to the given one
assert(list1.Filled(1));

assert(list1.Repr * list2.Repr == {}); // not disjoint, so ...
assert(list2.Valid()); //... list attribute may be violated
```

Related work

- **Java Modeling Language (JML)**
  - Many of the same specification features as Dafny
  - Support for subtyping
  - No algebraic datatypes
  - Big difference on tool side
    - *Tools for runtime checking*
    - *ESC/Java: extended static checking*
  - Different approach for modular verification
    - *Object invariants and data groups*

Related work (cont.)

- **Spec# (superset of C# 2.0)**
  - SMT-based automatic verifier
  - Support for subtyping
  - Lacks mathematical specification constructs
    - *No ghost-variables*
    - *No built-in sets or sequences*
    - *No algebraic datatypes*
    - *No termination metrics*
  - Different approach for modular verification
    - *Enforces ownership discipline among objects in the heap*

Future work

- **Compiler which generates executable code**
  - Mentioned in the paper from 2010
  - Now there is a **Dafny to C#** translator in the source tree
- **Future of Dafny goes into direction of proof assistant**
  - Inductive proofs (recent paper: “Co-induction Simply”)
  - “Verified Calculations”
    - *Writing steps of a proof and Dafny proves the steps*
    - *Paper from this year in collaboration with Nadia Polikarpova*
- **My ideas:**
  - Better error reporting (counter examples) when used from the command line
  - Support for subclasses

Resources

- Source code
  - https://dafny.codeplex.com/
  - https://boogie.codeplex.com/
  - Works with Mono
    - *I used Mono version 2.10.8 and MonoDevelop 3.0.6*
- Trying Dafny in the webbrowser
  - https://rise4fun.com/dafny
  - The example with the lists: http://rise4fun.com/Dafny/leKY
- More about Dafny
  - https://research.microsoft.com/en-us/projects/dafny/
- A collection of verification benchmarks
  - http://www.verifythis.org/
- Lecture on “Satisfiability Modulo Theories”
  - http://www.oprover.org/roberto/teaching/smt/

References

- Burdy, L. et al. (2005): An overview of JML tools and applications. International Journal on Software Tools for Technology Transfer, Volume 7 Issue 3. Springer, 2005.
- Hoare, T. et al. (2007): The Verified Software Initiative: A Manifesto
- Kassios, I. T. (2006): Dynamic Frames: Support for Framing, Dependencies and Sharing Without Restrictions. In J. Misra, T. Nipkow, E. Sekerinski (eds.) Formal Methods 2006, vol. 4085 of Lecture Notes in Computer Science, pages 268-283. Springer-Verlag, 2006.
- Leino, K. R. M. (2009a) Specification and verification of object-oriented software. Marktoberdorf International Summer School 2008, lecture notes.
- Leino, K. R. M. and Müller, P. (2009b): Using the Spec# language, methodology, and tools to write bug-free programs. LASER Summer School 2007/2008, LNCS 6029. Springer, 2009.
- Leino, K. R. M. (2010): Slides of the talk „Dafny: An automatic program verifier for functional correctness“ (available at https://research.microsoft.com/en-us/um/people/leino/papers/Dafny-LPAR-16.pptx)

Questions?

Implementation overview

Dafny source file (typically *.dfy)

Scanning/parsing using Coco/R
(Dafny.atg, Scanner.cs, Parser.cs)

Dafny AST (DafnyAst.cs)

Translation into Boogie

Boogie AST

Using the Boogie verifier for verification

Boogie verification results
(used to generate output)

Dafny.cs
DafnyDriver.cs

Example

```
class Node<T> {
    ghost var list : seq<T>;
    ghost var Repr : set<object>;

    var data : T;
    var next : Node<T>;

    function Valid() : bool
        reads this, Repr;
    {
        this in Repr && !(null in Repr) &&
        (next == null ==> list == [data]) &&
        (next != null ==>
               next in Repr && next.Repr <= Repr &&
               !(this in next.Repr) &&
               list == [data] + next.list &&
               next.Valid())
    }
...
```

Example (cont.)

```
...
    method Init(d : T)
        modifies this;
        ensures Valid() && fresh(Repr - {this});
        ensures list == [d];
        ensures data == d;
    {
        data := d; next := null;
        list := [d]; Repr := {this};
    }

    method Prepend(d : T) returns (r : Node<T>)
        requires Valid();
        ensures r != null;
        ensures r.Valid() && fresh(r.Repr - old(Repr));
        ensures r.list == [d] + list;
        ensures r.data == d;
    {
        r := new Node<T>;
        r.data := d; r.next := this;
        r.Repr := {r} + Repr;
        r.list := [r.data] + list;
    }
...
```

Example (cont.)

```
...
    function Filled(d : T) : bool
        reads Repr;
        requires Valid();
        ensures Valid();
    { forall k :: 0 <= k < |list| ==> list[k] == d }

    method Fill(d : T)
        modifies Repr;
        requires Valid();
        ensures Repr==old(Repr) && Valid() && Filled(d);
        decreases |list|;
    {
        data := d;

        if (next != null) {
            next.Fill(data);
            list := [data] + next.list;
        } else {
            list := [data];
        }
    }
}
```