http://lara.epfl.ch

Laboratory for
Automated Reasoning and Analysis

Viktor Kuncak
Assistant Professor, IC

a project: **http://JavaVerification.org**

ongoing class: **http://RichModels.org/LAT**

Spring, will be like: **http://lara.epfl.ch/sav09**

Automated Reasoning

General Problem Solver (Newell, Simon 1959)

- would take any problem description
  theorems, chess games, …
- output a solution

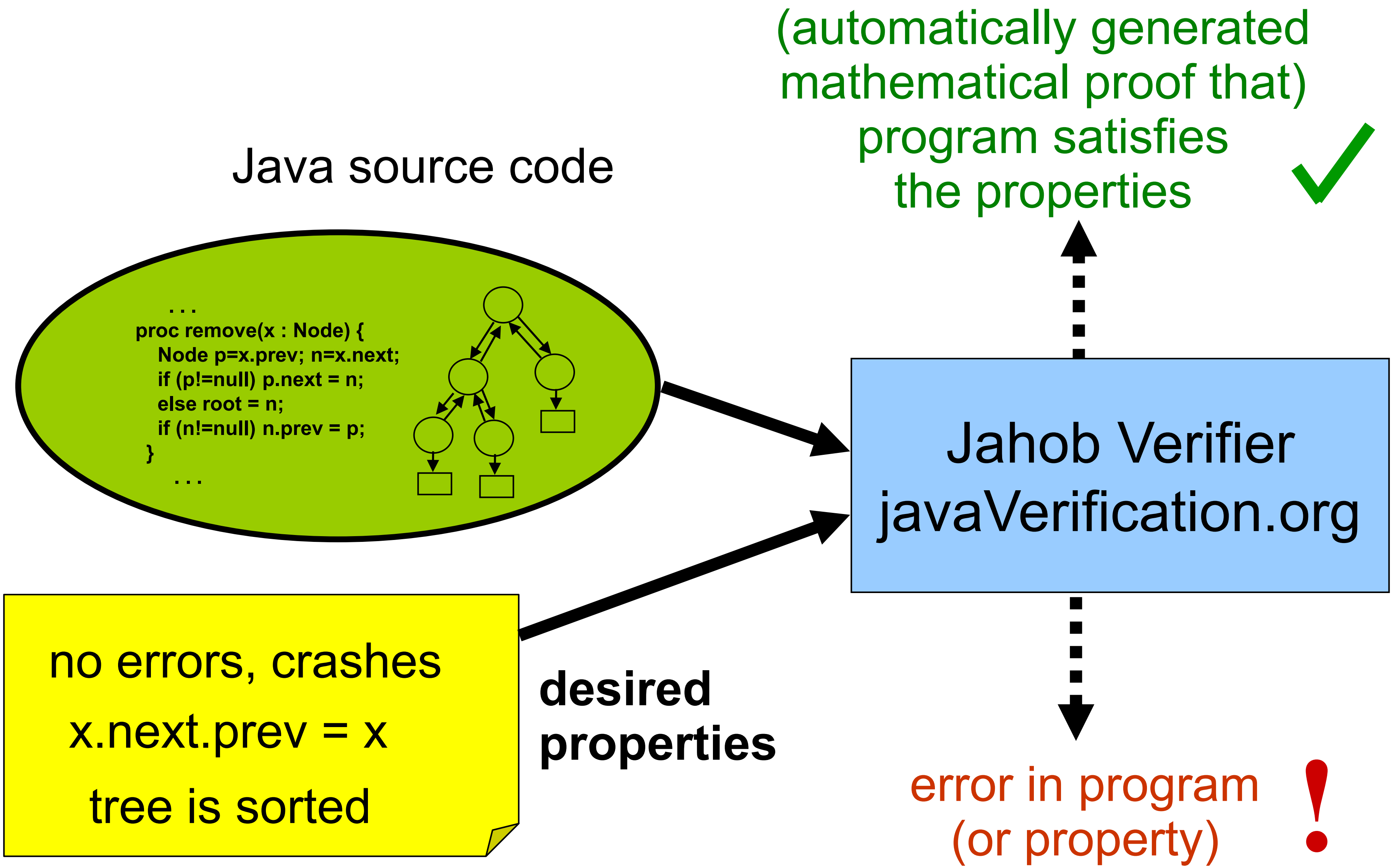
GPS was too ambitious to be useful

Trend since then: look at specific **domains**

An important domain:

- reasoning about models of computer systems (software, hardware, embedded systems)
- math, algorithms, software tools for this

Software Verification

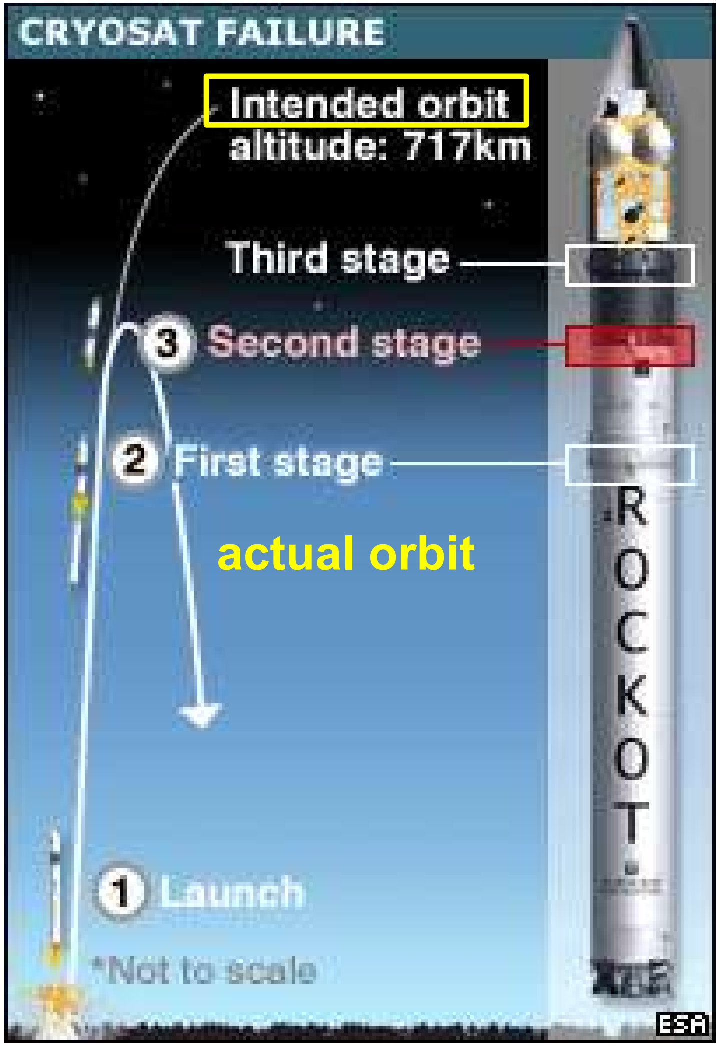


A Desired Property: No Crashes (from a BBC article)

Cryosat, a satelite worth

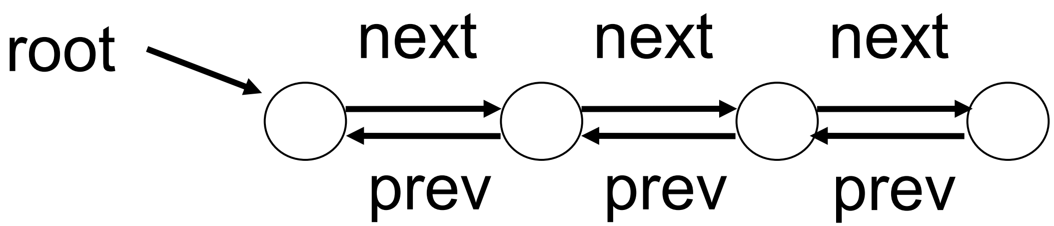
135m euro

October 2007



Desired Properties of Data Structures

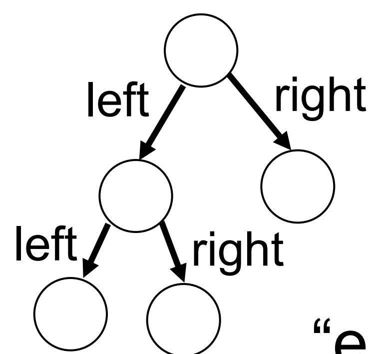
unbounded number of objects, dynamically allocated



“x.next.prev == x”

“acyclicity: $\sim next^{+}(x,x)$”

shape not given by types,
may change over time



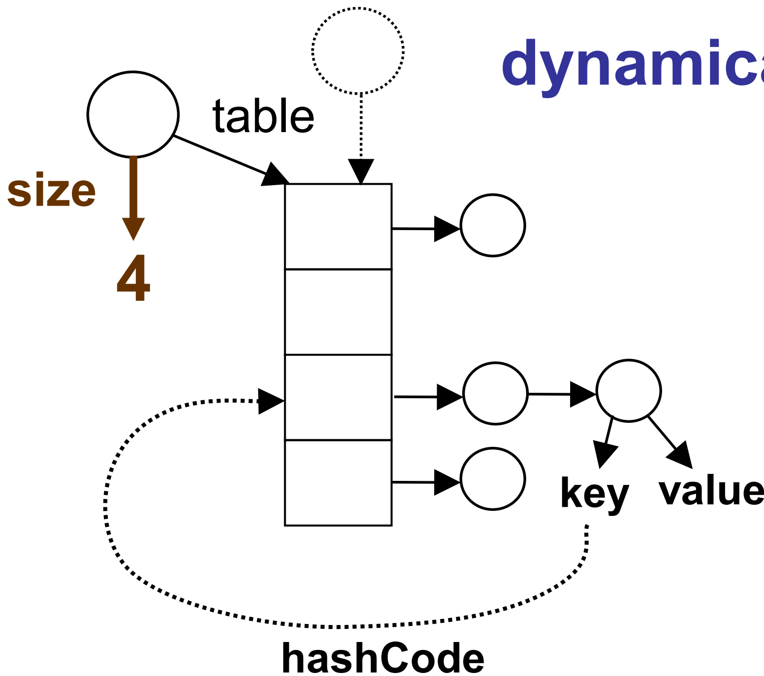
“graph is a tree”

“elements are sorted”

```
class Node {
    Node f1, f2;
}
```

Declaration alone admits both trees & lists – need “**invariants**”

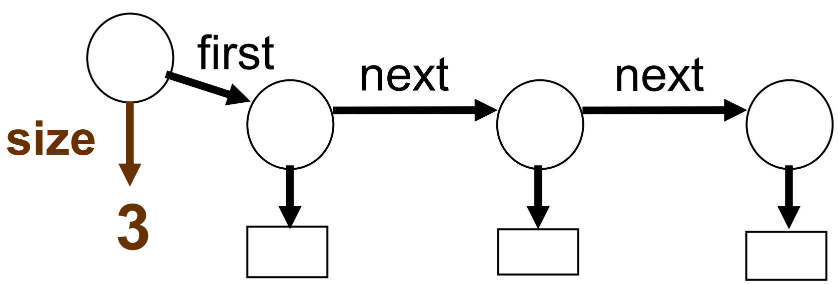
More Examples of Desired Properties



dynamically allocated arrays

node is stored in the bucket
given by the hash of node's key

instances do not share array



numerical quantities

value of size field is
number of stored objects
size = |{x. next*(first,x)}|

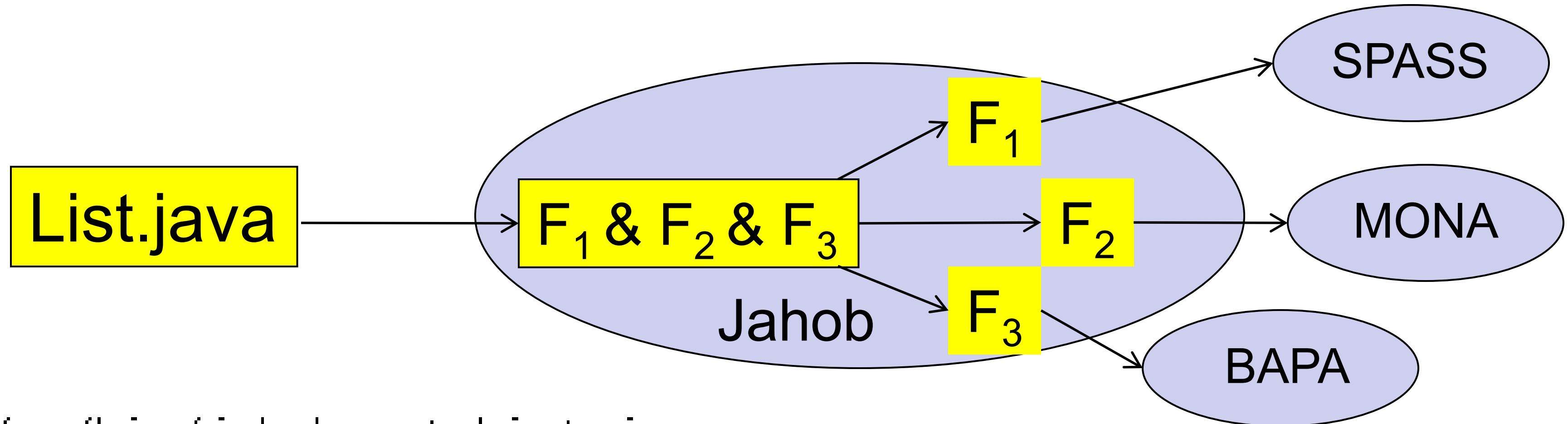
Specification in Jahob

specs as verified comments

public interface is simple

```
class List {
  private List next;
  private Object data;

  private static List root;
  private static int size;
  /*:
    private static ghost specvar nodes :: objset;
    public static ghost specvar content :: objset;

    invariant nodesDef: "nodes = {n. n ≠ null ∧ (root,n) ∈ {(u,
    invariant contentDef: "content = {x. ∃ n. x = List.data n ∧

    invariant sizeInv: "size = cardinality content";
    invariant treeInv: "tree [List.next]";
    invariant rootInv: "root ≠ null ⟶ (∀ n. List.next n ≠ roo
    invariant nodesAlloc: "nodes ⊆ Object.alloc";
    invariant contentAlloc: "content ⊆ Object.alloc";
   */

  public static void addNew(Object x)
  /*: requires "comment ''xFresh'' (x ∉ content)"
      modifies content
      ensures "content = old content ∪ {x}"
  */
  {
    List n1 = new List();
    n1.next = root;
    n1.data = x;
```

Verifying the addNew method

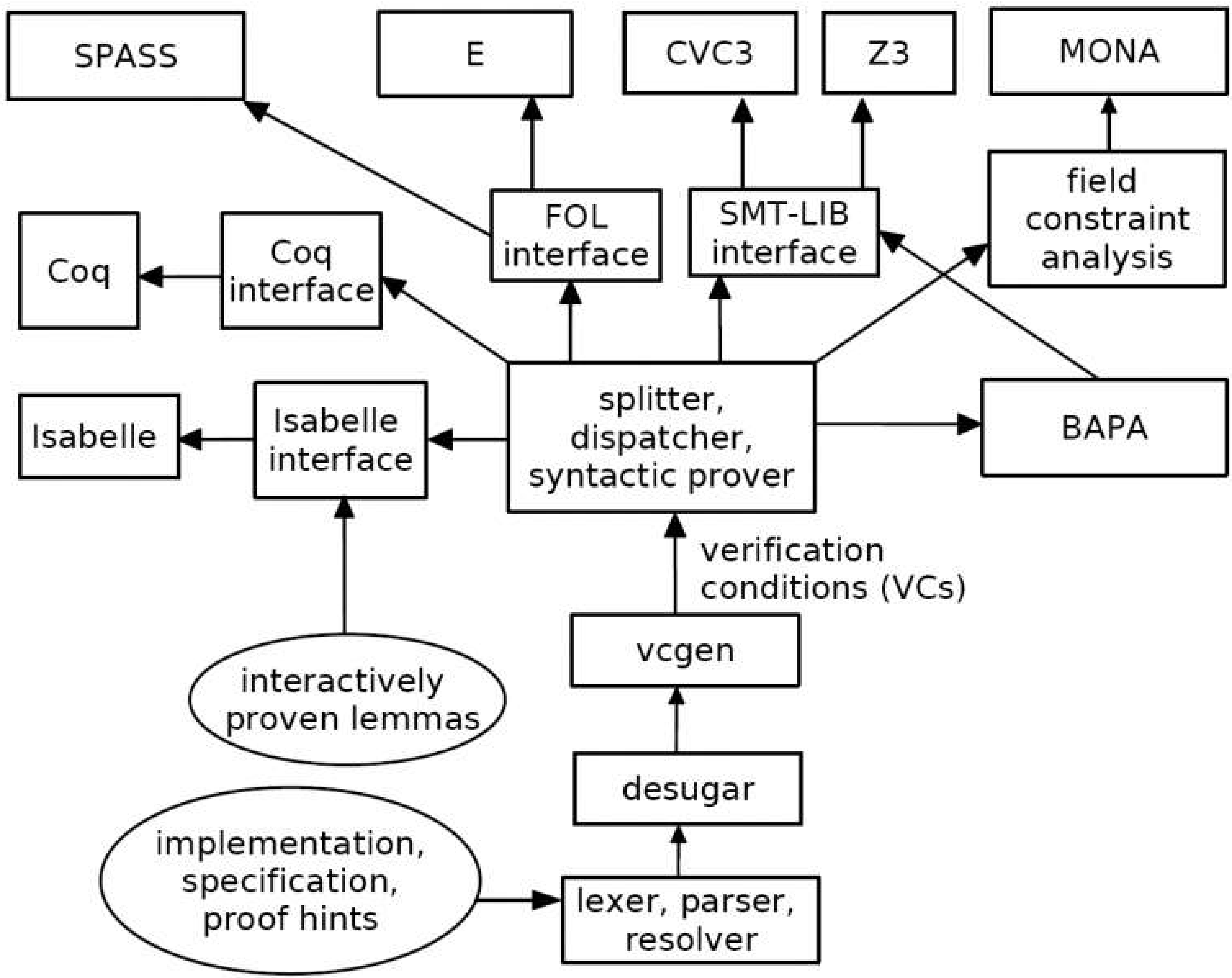



```
../../bin/jahob.opt List.java
-method List.addNew -usedp spass mona bapa
```

Verification steps

- generate verification condition (VC) in logic, stating "The program satisfies its specification"
- split VC into a conjunction of smaller formulas F_i
- prove each F_i conjunct using a number of specialized **theorem provers**

Jahob Verifier



Nature of Research in LARA

Two kinds of activities (closely related):

- Algorithms, Decidability, and Complexity (understand the problem we are solving)
- Making algorithms work in practice

We work with two kinds of objects:

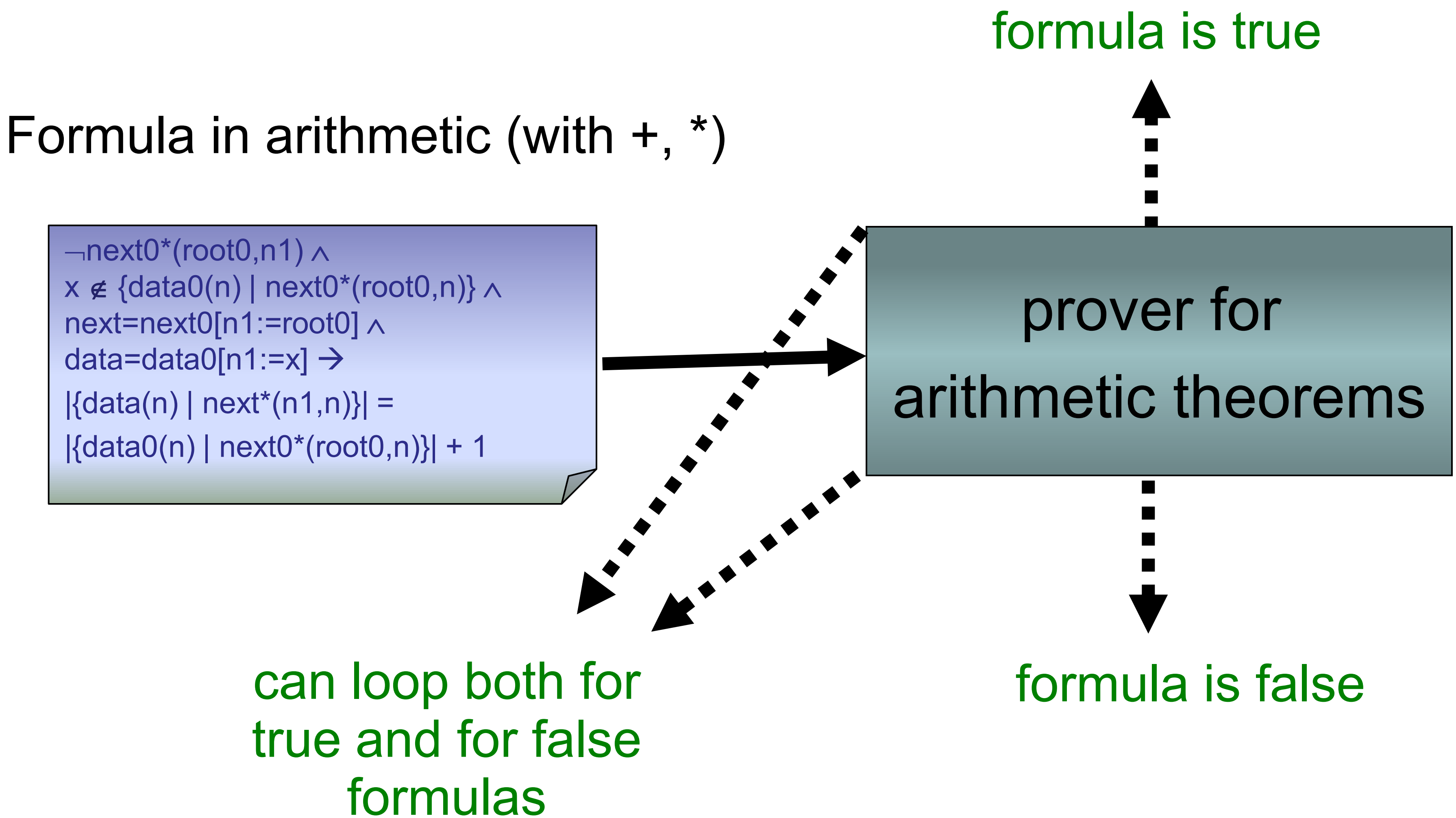
- programs (syntax trees, as in compilers)
- logical formulas (for properties and programs)

$\forall C.\ \exists\ p \in C.\ (\ \ A(p) \rightarrow (\forall x \in C.\ A(x))\ \ )$

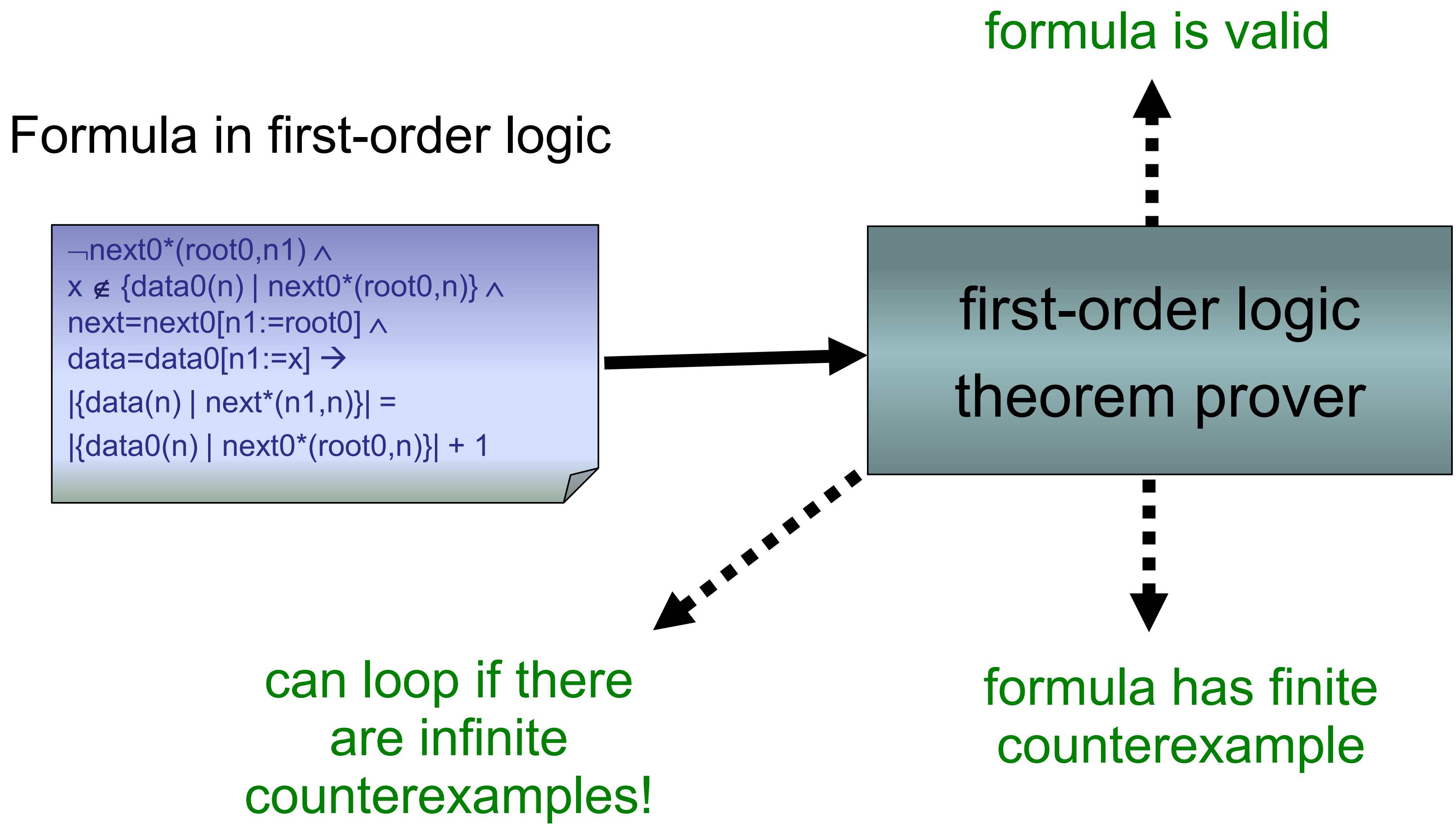
One aspect of our work:

Algorithms for checking validity of
logical formulas that describe correctness

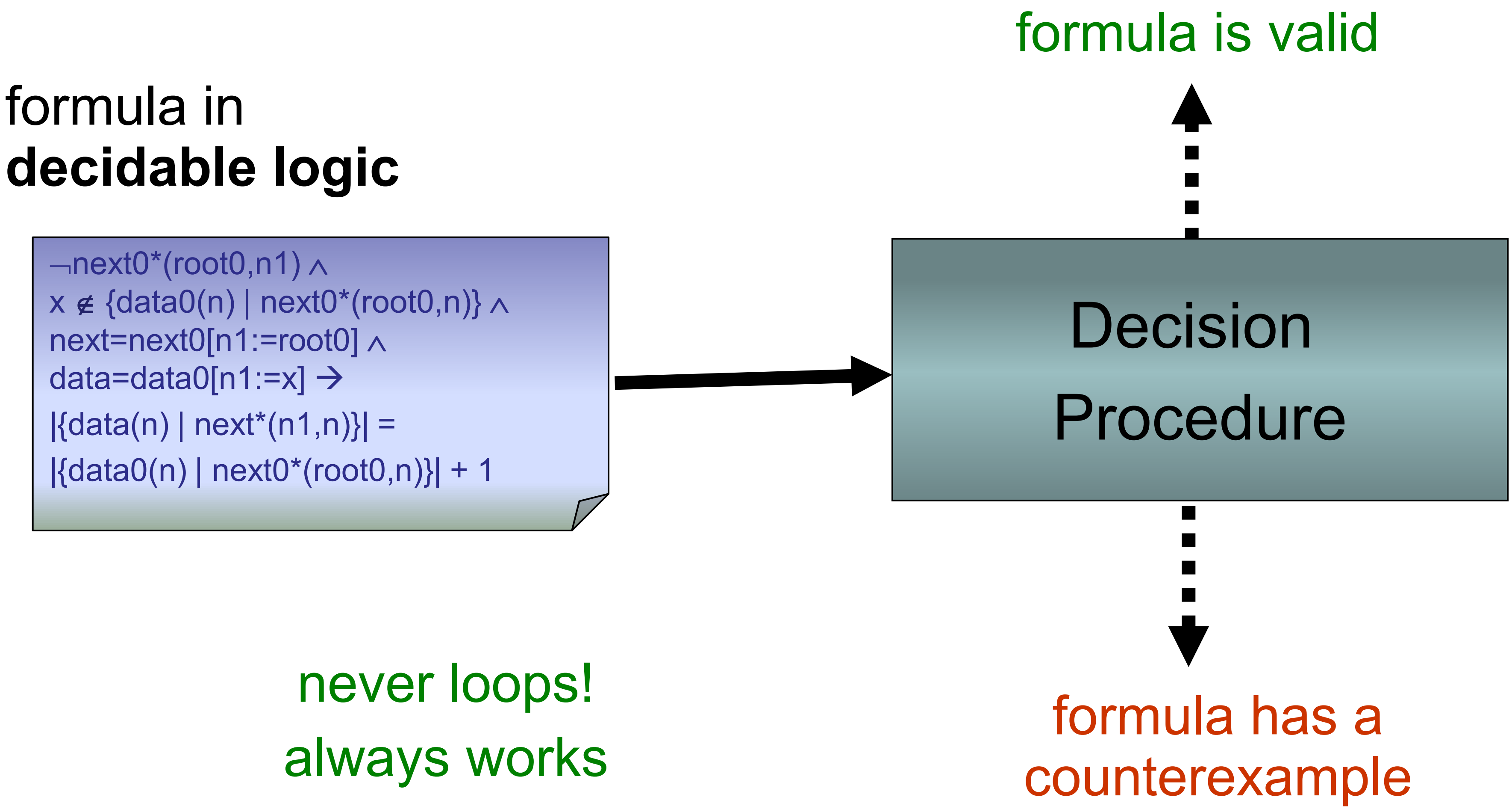
Algorithmic Difficulty for Arithmetic



Algorithmic Difficulty for full FOL



Decision Procedures



Example of Decidable Logics

- Integer arithmetic with only addition
- Integer arithmetic with only multiplication
- Real arithmetic with both addition and multiplication
- Set algebra (without nested sets)
- First-order logic with only two variables
- Logic of sets and elements interpreted over trees
  see http://RichModels.epfl.ch/LAT

Our Correctness Condition Formula

$\neg$next0*(root0,n1) $\wedge$ x $\notin$ {data0(n) | next0*(root0,n)} $\wedge$
next=next0[n1:=root0] $\wedge$ data=data0[n1:=x] $\rightarrow$
|{data(n) . next*(n1,n)}| =
|{data0(n) . next0*(root0,n)}| + 1

"The number of stored objects has increased by one."

Expressing this VC requires a rich logic

- transitive closure * (in lists and also in trees)
- unconstraint functions (data, data0)
- cardinality operator on sets | ... |

We have a decidable logic that can express this!

One component of this logic: Boolean Algebra with Presburger Arithmetic

$$S ::= V \mid S_1 \cup S_2 \mid S_1 \cap S_2 \mid S_1 \setminus S_2$$

$$T ::= k \mid C \mid T_1 + T_2 \mid T_1 - T_2 \mid C \cdot T \mid \mathrm{card}(S)$$

$$A ::= S_1 = S_2 \mid S_1 \subseteq S_2 \mid T_1 = T_2 \mid T_1 < T_2$$

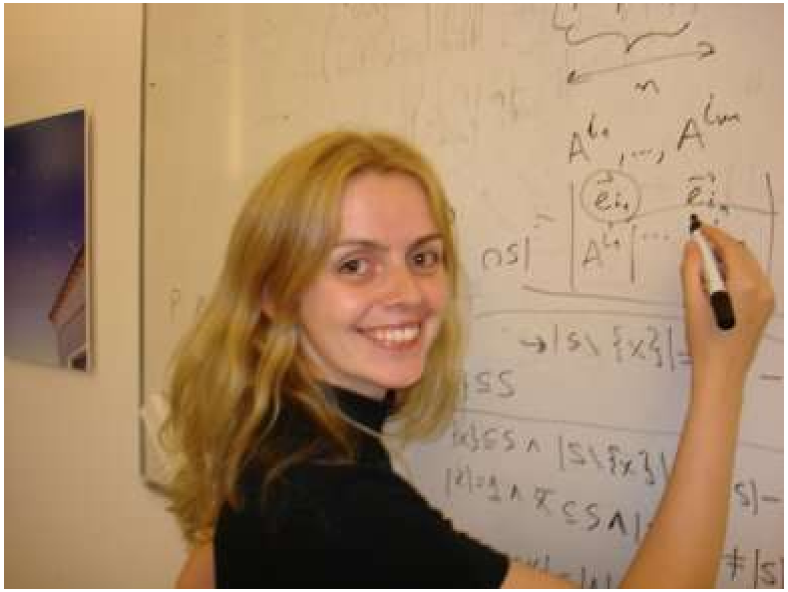
$$F ::= A \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg F \mid \exists S.F \mid \exists k.F$$

Not widely known: Feferman, Vaught: 1959

Our results

- first implementation for BAPA (CADE'05)
- first, exact, complexity for full BAPA (JAR'06)
- polynomial-time fragments of QFBAPA (FOSSACS'07)
- first, exact, complexity for QFBAPA (CADE'07)
- generalizations to bags (VMCAI'08, CAV'08,CSL'08)

Ruzica Piskac



3rd year PhD student

- MSc at the Max-Planck Institute
- Microsoft Resarch internship (Summer 2008)
- working on algorithms for proving formulas about sets, multisets, function images, cardinality

Combining Theories with Shared Set Operations. Symposium on frontiers of combining systems (FroCoS 2009)

Fractional **Collections with Cardinality Bounds**. Computer Science Logic (CSL 2008)

Linear Arith

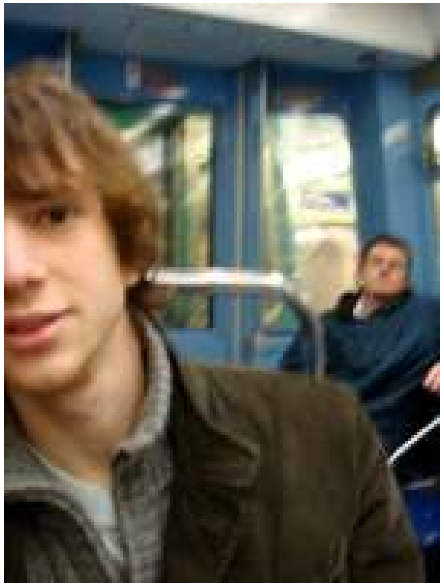
Decision Pr
Verification

$$\forall e.U(e) = 1 \ \wedge \ \forall e.0 \le A(e) \le 1 \ \wedge \ \forall e.0 \le B(e) \le 1 \ \wedge \ \forall e.0 \le U(e) \le 1$$

We next apply the definition of the cardinality operator, $|C| = \sum_{e \in E} C(e)$:

$$n_1 + n_2 < n_3 + n_4 \ \wedge \ n_1 = \sum_{e \in E} A(e) \ \wedge \ n_2 = \sum_{e \in E} U(e) \ \wedge$$
$$n_3 = \sum_{e \in E} (A \cap B)(e) \ \wedge \ n_4 = \sum_{e \in E} (A \cup B)(e) \ \wedge$$
$$\forall e.U(e) = 1 \ \wedge \ \forall e.0 \le A(e) \le 1 \ \wedge \ \forall e.0 \le B(e) \le 1 \ \wedge \ \forall e.0 \le U(e) \le 1$$

Philippe Suter



2^{nd} year PhD student

- MSc from EPFL, while visiting MIT

- Current work: verifying executable program specifications (written as functional Scala code)

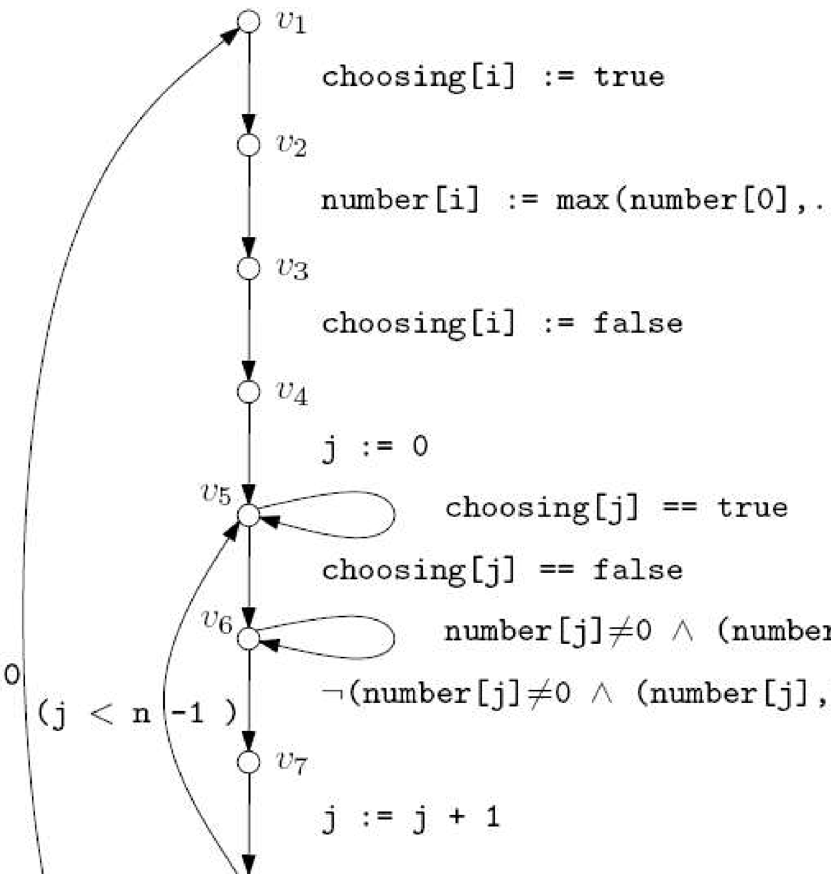
On Decision Procedures for Algebraic Data Types with Abstractions. EPFL Technical report, 2009

Non-Clausal Satisfiability Modulo Theories.
Master's Thesis, EPFL, September 2008

Hossein Hojjat

2[nd] year PhD student

- MSc from Eindhoven, Netherlands



Current work:

- verifying (Scala) programs
- using formulas for automated
- building automated reasoning

Giuliano Losa

1^{st} year PhD student

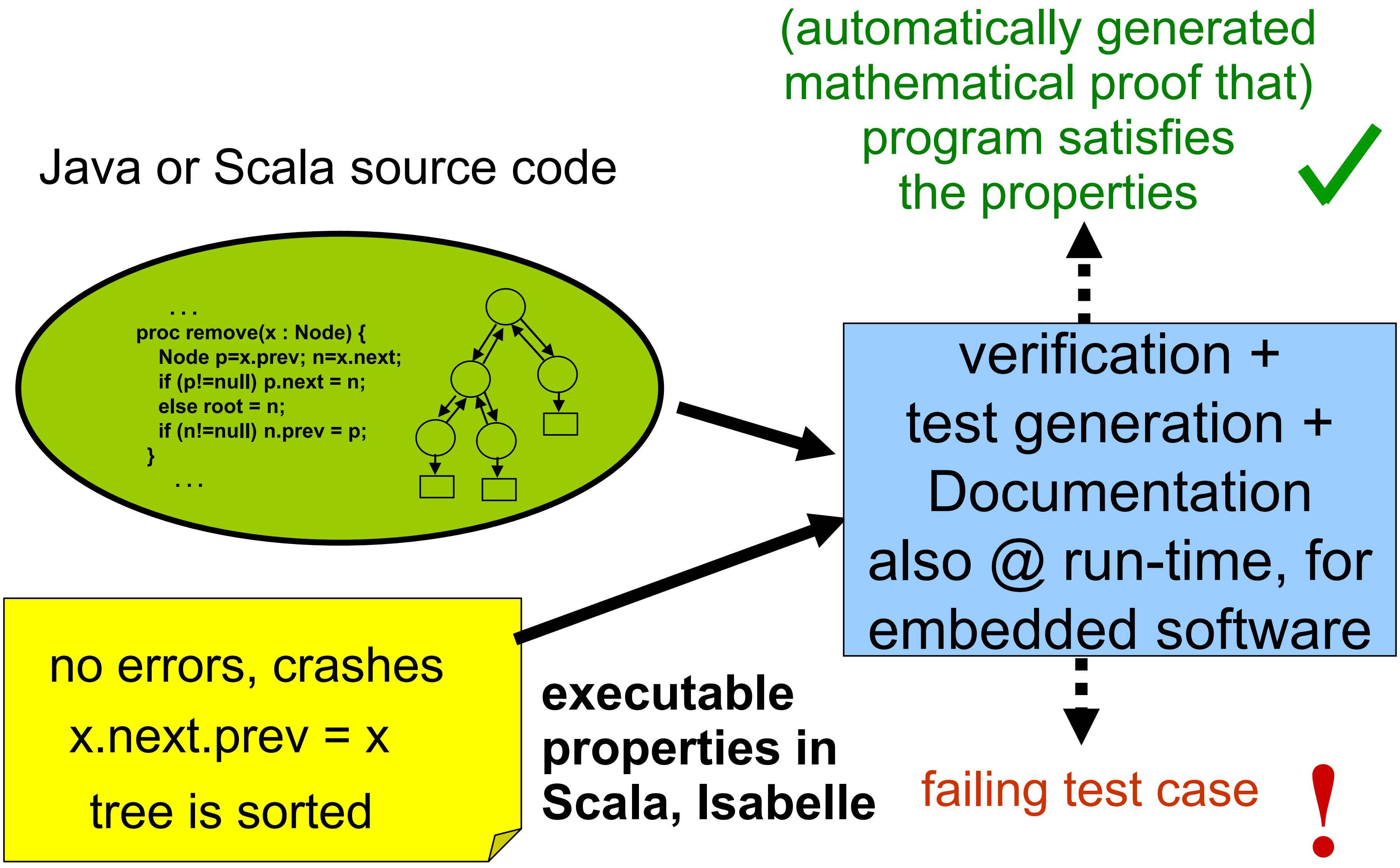
- MSc at EPFL

-Current work:
verifying distributed algorithms

Co-supervised w/
Prof. Rachid Guerraoui

Can we **prove** that "the penguins will indeed survive", (even in presence of evil penguins) and can automated reasoning help in this process?

Some Further Directions



http://lara.epfl.ch

Laboratory for Automated Reasoning and Analysis

Viktor Kuncak

Assistant Professor, IC

a project: **http://JavaVerification.org**

ongoing class: **http://RichModels.org/LAT**

Spring, will be like: **http://lara.epfl.ch/sav09**