Lab for Automated Reasoning and Analysis (LARA)

Helping construct software that does what we expect

Viktor Kunčak

https://lara.epfl.ch

Thanks to my current and former PhD students:

1. Sankalp Gambhir (topic: verification using Lisa proof assistant)
2. Matthieu Bovel (topic: dependent types; primary supervisor: Martin Odersky)
3. Simon Guilloud (topic: foundations of the Lisa proof assistant)
4. Rodrigo Raya (topic: decision procedures)
5. Dragana Milovančević (topic: program equivalence verification)
6. Georg S. Schmid (2022), Stripe Inc.
7. Romain Edelmann (2021), EPFL and Gymnase de Burier
8. Emmanouil Koukoutos (2019), Google LLC
9. Nicolas Charles Yves Voirol (2019), Google LLC
10. Mikaël Mayer (2017), AWS
11. Ravichandhran Kandhadai Madhavan (2017), Apple Inc.
12. Regis Blanc (2017), Google LLC
13. Etienne Kneuss (2016), CTO and partner at IMMOMIG SA
14. Tihomir Gvero (2015)
15. Eva Darulova (2014), **Associate Professor** at **Uppsala University**
16. Giuliano Losa (2014, co-advised with Rachid Guerraoui), Galois, Inc.
17. Hossein Hojjat (2013), faculty at Rochester Institute of Technology and Tehran
18. Philippe Suter (2012), Two Sigma
19. Ruzica Piskac (2011), Donna L. Dubinsky **Associate Professor**, **Yale** University

Collaboration with New Professors in EDIC

In the coming year or so, two more faculty working (among others) on formal methods will arrive and start new research groups:

1. Clément Pit-Claudel, `http://pit-claudel.fr/clement/`
   "programming languages, compilers, and formal verification; my broader interests include systems engineering, hardware design languages, security, performance engineering, databases, and type theory."
2. Thomas Bourgeat, `https://people.csail.mit.edu/bthom/`
   "My research is between computer architecture and programming languages. I work on leveraging high-level hardware programming languages and formal methods to design hardware, that I can then formally prove is veryfing its specification. I am interested in looking at domain-specifc hardware accelerators, and security-oriented properties to apply the methodology."

If you are ultimately interested to work with them, please contact them and me; I will consider EDIC projects that may be good to prepare you to work with them.

Where to learn more about research in LARA?

The best introduction is a 6 credit MSc (and PhD) course "Formal Verification":

`https://gitlab.epfl.ch/lara/cs550/`

Videos and slides are available. The course has substantial project component.

Example shorter projects of interest:

`https://gitlab.epfl.ch/kuncak/student-projects`

What is Stainless?

`https://github.com/epfl-lara/stainless/`

For a computable function f, Stainless checks:

- safety: $\forall i.f(i)$, finding both proofs and counterexamples
- termination: $\forall i \in D$, executing $step(f, i)$ terminates

Input: Scala (by Martin Odersky @ EPFL), most popular functional language

- #13 on GitHub (GitHut 2.0), #21 most searched (PYPL)

Memory safe, strongly and statically typed

Aim: check code that runs (on JVM via scalac, or via C)

- mapping real language to semantics is a major part of this effort

Flavor of Stainless

Example: list with map and size

```
enum List[T]:
  case Nil()
  case Cons(head: T, tail: List[T])

  def map[U](f: T => U): List[U] =
    this match
      case Nil() => Nil()
      case Cons(head, tail) => Cons(f(head), tail.map(f))

  def size: BigInt = {
    this match
      case Nil() => BigInt(0)
      case Cons(_, tail) => BigInt(1) + tail.size
  }.ensuring(_ >= 0)
```

Demo: zip

```
def zip(xs: List[Int], ys: List[Boolean]): List[(Int, Boolean)] = {

  (xs, ys) match
    case (Cons(x, xs0), Cons(y, ys0)) =>
           Cons((x, y), zip(xs0, ys0))

    case _ => nil
}.ensuring (_.map(_._1) == xs)
```

Demo: zip

```
def zip(xs: List[Int], ys: List[Boolean]): List[(Int, Boolean)] = {

  (xs, ys) match
    case (Cons(x, xs0), Cons(y, ys0)) =>
           Cons((x, y), zip(xs0, ys0))

    case _ => nil
}.ensuring (_.map(_._1) == xs)

warning: Found counter-example:
warning: xs: List[Int] -> Cons[Int](0, Nil[Int]())
  ys: List[Boolean] -> Nil[Boolean]()
```

zip with precondition

```
def zip(xs: List[Int], ys: List[Boolean]): List[(Int, Boolean)] = {
  require(xs.size <= ys.size)
  (xs, ys) match
    case (Cons(x, xs0), Cons(y, ys0)) =>
           Cons((x, y), zip(xs0, ys0))

    case _ => nil
}.ensuring (_.map(_._1) == xs)
```

Observations about zip

The code given to Stainless compiles with standard Scala compiler and interpreter (specifications, valid Scala, become just dynamic assertions).

Multiple **recursive** functions in code and specifications (zip, size, map)

Function map is defined in user code as **higher order**

The system was able to **prove** properties of such code

The system was able to **find counterexamples**

All functions were shown terminating

Demo: non-terminating code

```
def map[U](f: T => U): List[U] =
  this match
    case Nil() => Nil()
    case Cons(head, tail) => Cons(f(head), this.map(f))
```

Demo: non-terminating code

```
  def map[U](f: T => U): List[U] =
    this match
      case Nil() => Nil()
      case Cons(head, tail) => Cons(f(head), this.map(f))
```

```
zip.scala:5:7: warning: Function map loops given inputs:
thiss: List[T] -> Cons[T](T#2, Nil[T]())
f: (T) => U -> (x158: T) => U#0
              def map[U](f: T => U): List[U] =
                  ^
```

Verified case studies using Stainless

`https://github.com/epfl-lara/bolts/`

Balanced trees, ConcTrees (LCPC 2015), hash tables - shown $\approx$ lists

Quite Okay Image Format `https://qoiformat.org/`
decode(encode(img))=img

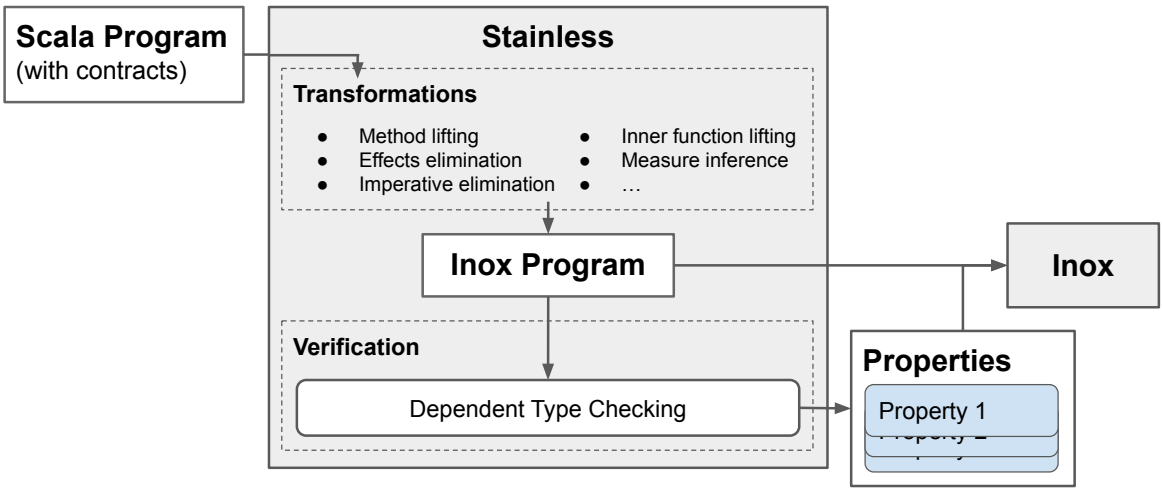
Soundness of System F (first-class polymorphism), 2.8 kLOC (class project)

Tendermint blockchain client, distributed systems using actors

From Verified Scala to STIX File System Embedded Code using Stainless (NFM 2022)

- Hamza, Felix, Kunčak, Nussbaumer, Schramka - collaboration with Ateleris
- generated efficient C code (with preallocated data), can replace existing one

Semantic-preserving translation pipeline



Inox: model checking functional programs

`https://github.com/epfl-lara/inox`

$f(x) = E(x, f(x-1))$ ensuring $x \geq 0$

$f(x) = E(x, f(x-1)) = E(x, E(x-1, \mathbf{f}(x-2))) = \ldots$

Consider condition $f(x) = 42$
Inox progressive and fairly unfolds bodies and postconditions of called functions

- underapproximating check is like BMC for recursive functions
  (innermost calls made unreachable by blocking branches that lead to them)
  $E(x, E(x-1, \bot)) = 42$, with $\bot$ encoded by blocking branches leading to $f(x-2)$
- overapproximating check is a form of k-induction on well-foundness of definition
  (innermost calls uninterpreted)
  $E(x, E(x-1, \mathbf{a})) = 42 \wedge E(x, E(x-1, a)) \geq 0 \wedge E(x-1, a) \geq 0$, where a is arbitrary

Future Work on Stainless

Stainless is usable (by us, 100+ students, industrial partners)

Open source: `https://github.com/epfl-lara/stainless/`

Future directions:

- more automation for equivalence and other important high-level properties
- further refactoring (Horn clauses) to help applicability to other languages
- caching and learning formulas
- easier use of imperative features
- simple forms of concurrency
- more control and (hopefully simpler) foundation via set theory

LISA: towards a set theoretic proof assistant

The use of type theory as foundation for proof assistant is a historical and social phenomenon.

An older phenomenon: set theoretic foundations of mathematics

- inconsistencies in type theory found as late as 1970ies
- Zermelo-Fraenkel (1921 letter): still standing, still useful, after over 100 years

Claim: we can build effective proof assistants on mainstream set theory
past examples: TLA+, Isabelle/ZF, Isabelle/HOL/ZF, Mizar

LISA: towards a set theoretic proof assistant

The use of type theory as foundation for proof assistant is a historical and social phenomenon.

An older phenomenon: set theoretic foundations of mathematics

- inconsistencies in type theory found as late as 1970ies
- Zermelo-Fraenkel (1921 letter): still standing, still useful, after over 100 years

Claim: we can build effective proof assistants on mainstream set theory
past examples: TLA+, Isabelle/ZF, Isabelle/HOL/ZF, Mizar

Proof is a sequence steps in classical sequent calculus (Gentzen 1934)

- sequences are linearized DAGs (trees are less practical)

What is a good notion of polynomially checkable formal proof?

Obviously equivalent Boolean formulas

Consider a proven theorem of the form:

$$\neg(A \wedge B \wedge C) \vee D \quad (T)$$

and suppose we need to prove this goal somewhere in the proof:

$$D \vee \neg B \vee \neg(C \wedge A) \quad (G)$$

The goal should follows immediately from the theorem!

- (G) is a trivial variation of (P) via de Morgan laws and AC for $\vee, \wedge$

Syntactically, these formulas are distinct.

Checking propositional equivalence is coNP-complete.

Obviously equivalent Boolean formulas

Consider a proven theorem of the form:

$$\neg(A \wedge B \wedge C) \vee D \quad (T)$$

and suppose we need to prove this goal somewhere in the proof:

$$D \vee \neg B \vee \neg(C \wedge A) \quad (G)$$

The goal should follows immediately from the theorem!

- (G) is a trivial variation of (P) via de Morgan laws and AC for $\vee, \wedge$

Syntactically, these formulas are distinct.

Checking propositional equivalence is coNP-complete.

Is there a principled sufficient equivalence check that accepts such pairs of formulas as equivalent and that is efficient?

Boolean algebras without distributivity - OCBSL

L1:	$x \sqcup y = y \sqcup x$	L1':	$x \wedge y = y \wedge x$
L2:	$x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$	L2':	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
L3:	$x \sqcup x = x$	L3':	$x \wedge x = x$
L4:	$x \sqcup 1 = 1$	L4':	$x \wedge 0 = 0$
L5:	$x \sqcup 0 = x$	L5':	$x \wedge 1 = x$
L6:	$\neg\neg x = x$	L6':	same as L6
L7:	$x \sqcup \neg x = 1$	L7':	$x \wedge \neg x = 0$
L8:	$\neg(x \sqcup y) = \neg x \wedge \neg y$	L8':	$\neg(x \wedge y) = \neg x \sqcup \neg y$
$\Uparrow$	$(\text{ortho complemented bi semilattice})$		

Boolean algebras without distributivity - OCBSL

L1:	$x \sqcup y = y \sqcup x$	L1':	$x \wedge y = y \wedge x$
L2:	$x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$	L2':	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
L3:	$x \sqcup x = x$	L3':	$x \wedge x = x$
L4:	$x \sqcup 1 = 1$	L4':	$x \wedge 0 = 0$
L5:	$x \sqcup 0 = x$	L5':	$x \wedge 1 = x$
L6:	$\neg\neg x = x$	L6':	same as L6
L7:	$x \sqcup \neg x = 1$	L7':	$x \wedge \neg x = 0$
L8:	$\neg(x \sqcup y) = \neg x \wedge \neg y$	L8':	$\neg(x \wedge y) = \neg x \sqcup \neg y$
$\Uparrow$	(ortho complemented bi semilattice)		
L10:	$x \sqcup (y \wedge z) = (x \sqcup y) \wedge (x \sqcup z)$	L10':	$x \wedge (y \sqcup z) = (x \wedge y) \sqcup (x \wedge z)$
$\Uparrow$	(Boolean algebra)		

Result (Guilloud, K. TACAS'22)

$O(n \log^2(n))$ algorithm for equality in the class of structures given by those axioms

- rooted tree isomorphism algorithm: numbering, *not* a GI-complete problem
- completion of the term-rewrite system modulo AC

Axioms have a number of simple critical pairs, resolving gives convergent and terminating system.

Implemented in LISA proof assistant as part of proof checking procedure.

Could be deployed as fast tactic in other proof assistants and verifiers