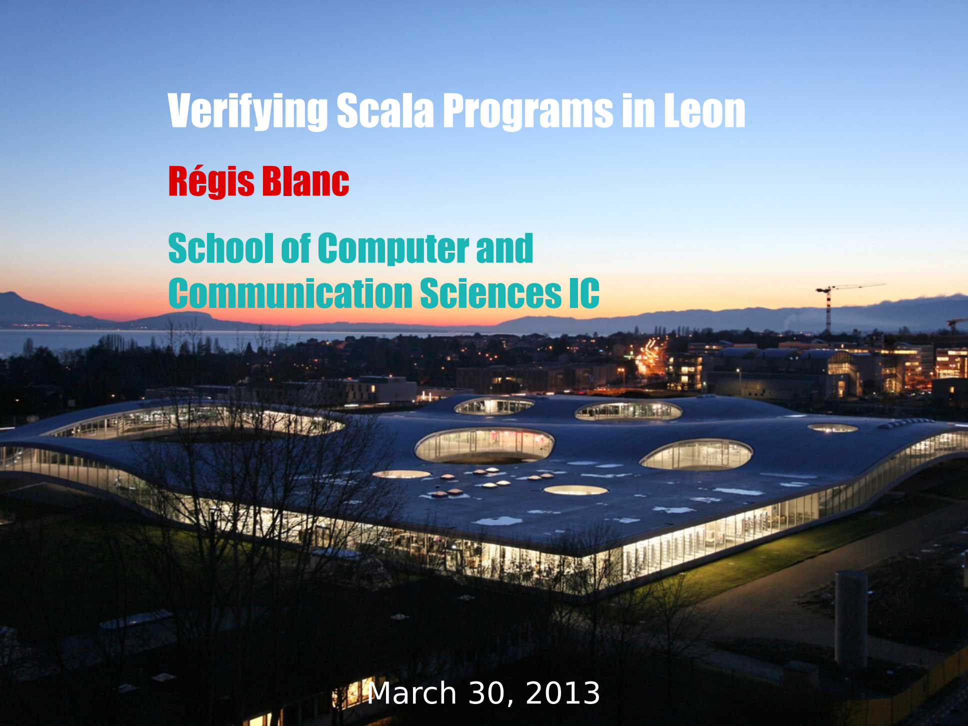


Verifying Scala Programs in Leon

Régis Blanc

School of Computer and Communication Sciences IC

March 30, 2013

The Leon Verification System

- Verifier for the Scala language.
- Support a well-defined subset of Scala.
  - A functional core language.
  - Many imperative extensions.
  - Some ways to express non-determinism.
- Complete for finding counterexamples.
- Current team: Régis Blanc, Etienne Kneuss, Viktor Kuncak, Philippe Suter
- Past contributors: Ali Sinan Köksal, Octavian Ganea, Robin Steiger, Utkarsh Upadhyay.

Contracts

Specifications can be defined using contracts.

Contracts

Specifications can be defined using contracts.

- Postconditions

```
def abs(n: Int): Int = {
  if(n <= 0) -n else n
} ensuring(res => res >= 0)
```

Contracts

Specifications can be defined using contracts.

- Postconditions

```
def abs(n: Int): Int = {
  if(n <= 0) -n else n
} ensuring(res => res >= 0)
```

- Preconditions

```
def fact(n: Int): Int = {
  require(n >= 0)
  if(n == 0) 1 else n * fact(n-1)
}
```

Contracts

Specifications can be defined using contracts.

- Postconditions

```
def abs(n: Int): Int = {
  if(n <= 0) -n else n
} ensuring(res => res >= 0)
```

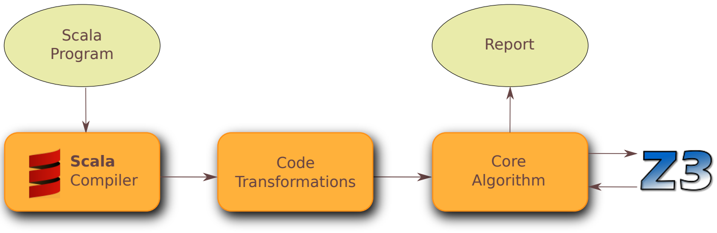
- Preconditions

```
def fact(n: Int): Int = {
  require(n >= 0)
  if(n == 0) 1 else n * fact(n-1)
}
```

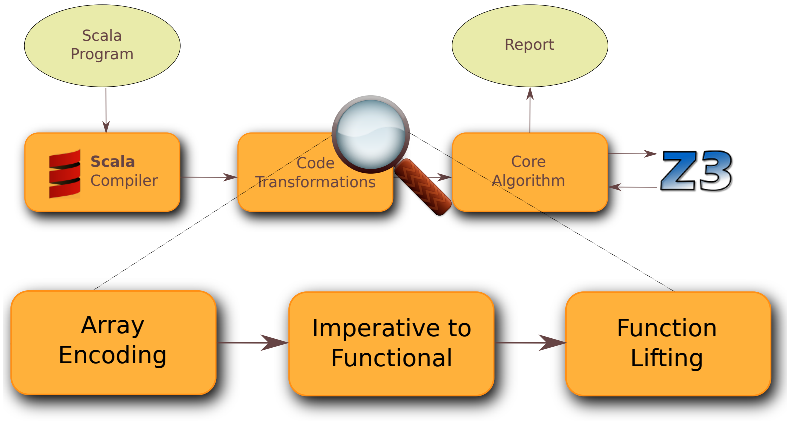
The implementation and specification languages are the same.

Demo

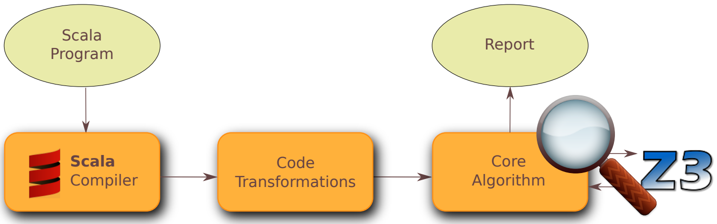
Our Architecture



Our Architecture



Our Architecture



1. Over-approximate with uninterpreted functions, if UNSAT then return UNSAT.
2. Add blocking predicates to block branches containing non-unrolled function invocations, if SAT, return SAT.
3. Choose, in some fair way, function invocations to unroll and unroll them, go to step 1.

Reference:

P. Suter, A.S. Köksal, V. Kuncak, Satisfiability Modulo Recursive Programs, SAS'11

Overview of some Results

Benchmark	LOC	#VCs			Time (s)
		V	I	U	
`AssociativeList`	50	11	0	0	0.23
`InsertionSort`	99	14	1	0	0.42
`RedBlackTree`	117	20	4	0	3.73
`PropositionalLogic`	86	22	1	0	2.36
`AmortizedQueue`	124	32	0	0	3.37
`Arithmetic`	73	10	1	0	0.33
`ArrayOperations`	207	36	0	7	2.37
`ListOperations`	146	21	4	1	4.34
`Constraints`	76	6	3	1	2.41

- Each verification condition (VC) can be Valid, Invalid or Unknown (timeout).
- Different kinds of VCs:
  - loop invariants, preconditions, postconditions, array accesses, and exhaustiveness of match expressions.

Thank you for your attention

Régis Blanc
School of Computer and
Communication Sciences IC

ic.epfl.ch