Synthesis, Analysis, and Verification

Lecture 01c

About Synthesis
General Background for the Course

Lectures:

Viktor Kuncak

Exercises and Labs:

Eva Darulová

Giuliano Losa

Friday, 25 February 2011

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Logistics, Exercises and Demos

Everyone should register for the course ‘SAV’ in

http://moodle.epfl.ch

whatAFunCourse

Please obtain “The Calculus of Computation” book

Bring your laptops whenever you can

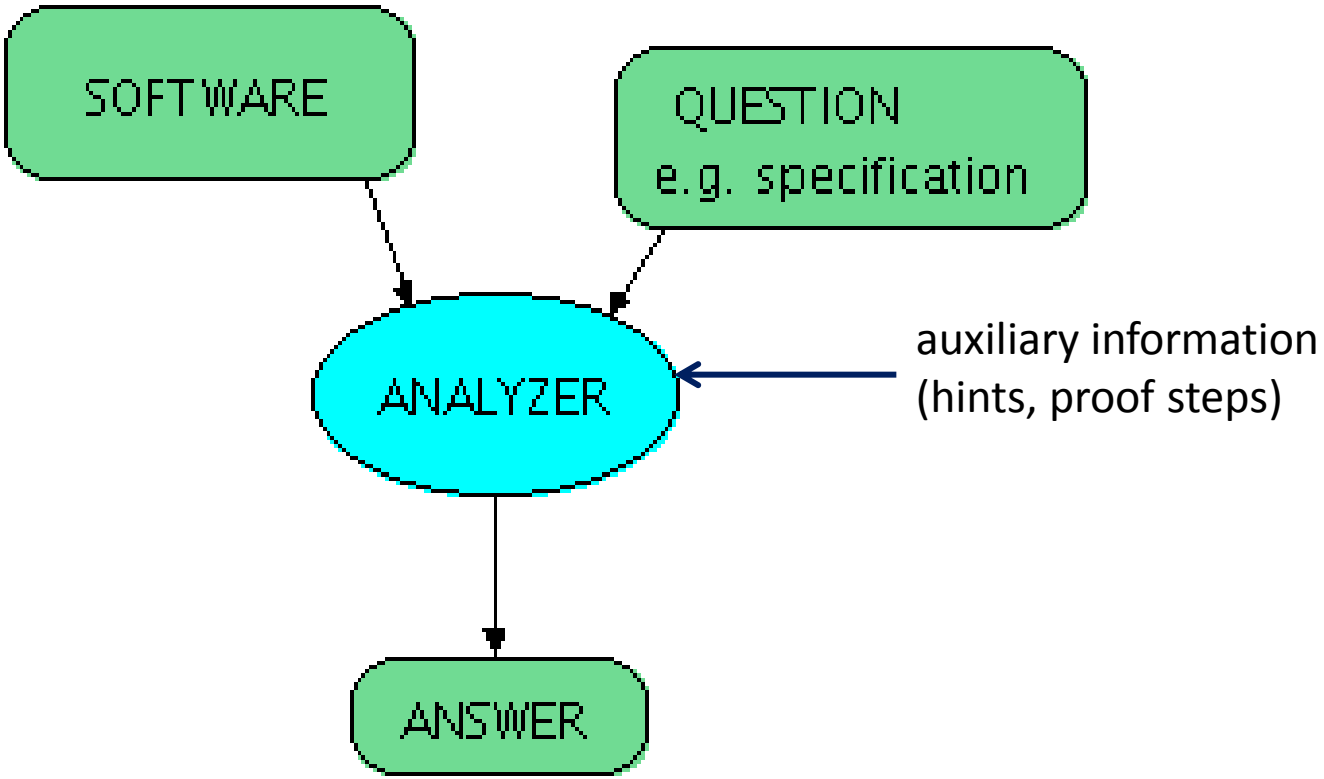
If you do not know Scala, please take a look now

- can write ~Java, can write ~Haskell/Ocaml, see more at:

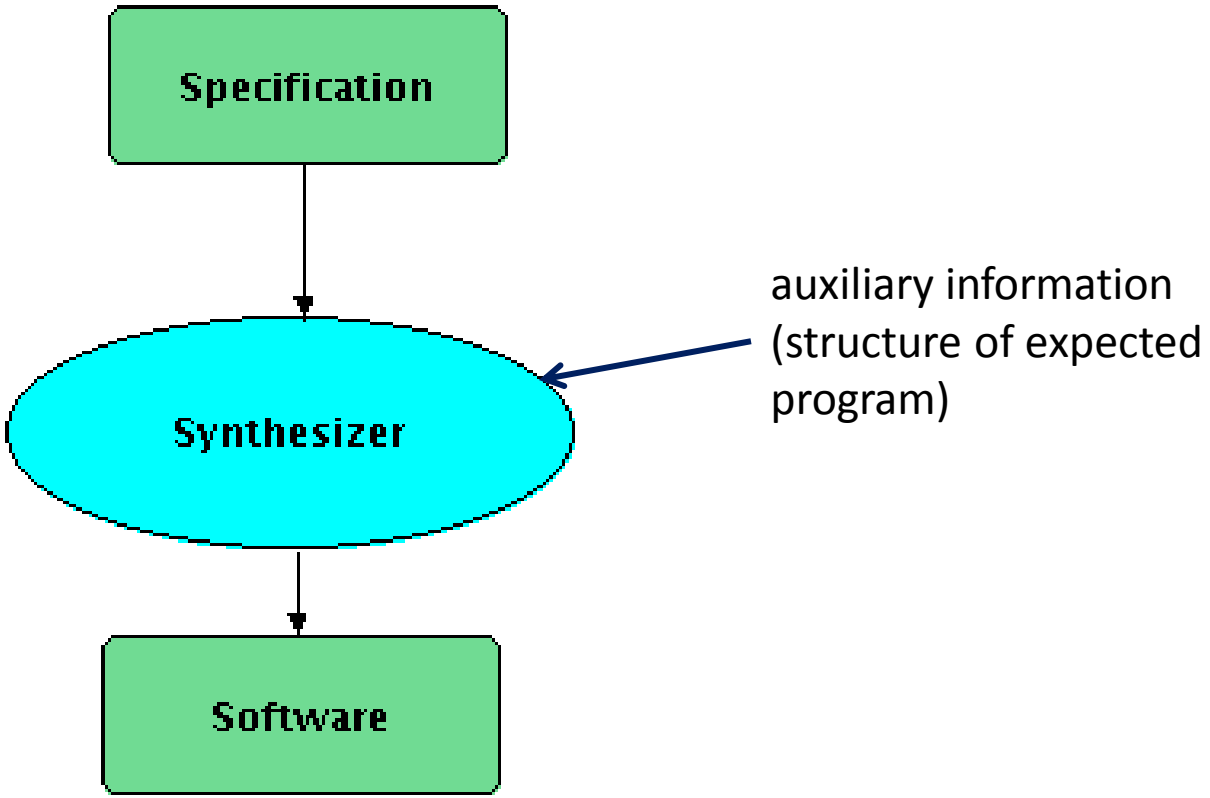
http://scala-lang.org

Example: binary tree

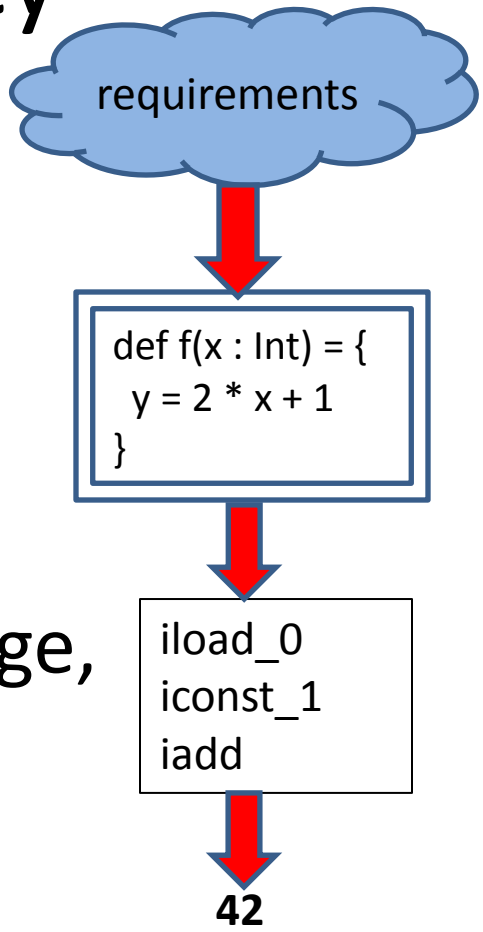
Analysis and Verification



Synthesis


Specification
Synthesizer
Software
auxiliary information
(structure of expected
program)

Programming Activity



Consider three related activities:

- Development within an IDE (Eclipse, Visual Studio, emacs, vim)
- Compilation and static checking (optimizing compiler for the language, static analyzer, contract checker)
- Execution on a (virtual) machine

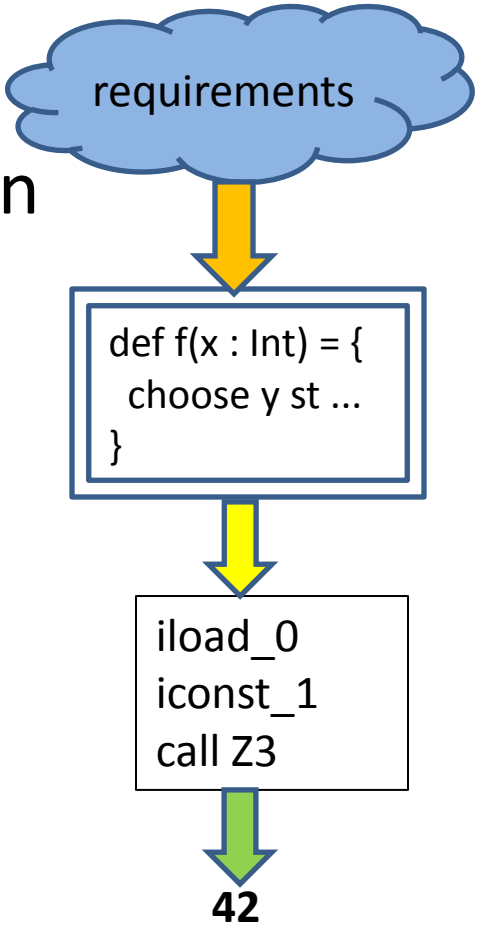
More compute power available for each of these

→ use it to improve programmer productivity

Synthesis at All Levels

Opportunities for implicit programming in

- **Development** within an IDE
  - **isynth** tool
- **Compilation**
  - **Comfusy** and **RegSy** tools
- **Execution**
  - **Scala^Z3** and **UDITA** tools



I next examine these tools, from last to first, focusing on Compilation

isynth - Interactive Synthesis of Code Snippets

```
def map[A,B](f:A => B, l:List[A]): List[B] = { ... }
def stringConcat(lst : List[String]): String = { ... }
...
def printInts(intList:List[Int], prn: Int => String): String = □
```

Returned value:
stringConcat(map[Int, String](prn, intList))

Is there a term of given type in given environment?

Monorphic: decidable. Polymorphic: undecidable

with: **Tihomir Gvero**, **Ruzica Piskac**

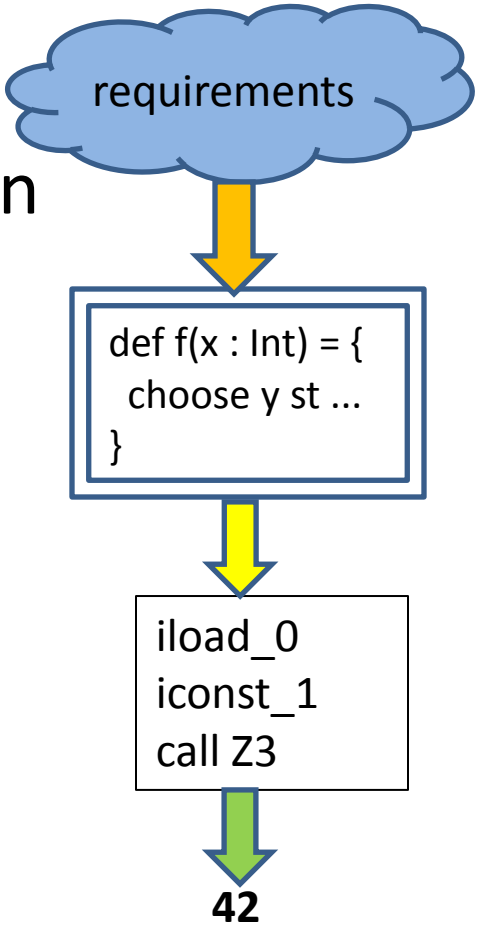
isynth - Interactive Synthesis of Code Snippets

- supports method combinations, type polymorphism, user preferences
- based on first-order resolution – combines forward and backward reasoning
- ranking of returned solutions is obtained through a system of weights

Synthesis at All Levels

Opportunities for implicit programming in

- √ **Development** within an IDE
  - **isynth** tool
- **Compilation**
  - **Comfusy** and **RegSy** tools
- **Execution**
  - **Scala^Z3** and **UDITA** tools



An example

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  choose((h: Int, m: Int, s: Int) ⇒ (
        h * 3600 + m * 60 + s == totalSeconds
    && h ≥ 0
    && m ≥ 0 && m < 60
    && s ≥ 0 && s < 60    ))
```

3787 seconds ⟶ 1 hour, 3 mins. and 7 secs.

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  val t1 = totalSeconds div 3600
  val t2 = totalSeconds + ((-3600) * t1)
  val t3 = min(t2 div 60, 59)
  val t4 = totalSeconds + ((-3600) * t1) + (-60 * t3)
  (t1, t3, t4)
```

Possible starting point: quantifier elimination

- A specification statement of the form

$$\vec{r} = \textbf{\textit{choose}}(\vec{x} \Rightarrow F(\vec{a}, \vec{x}))$$

"*let* r *be* x *such that* F(a, x) *holds*"

- Corresponds to constructively solving the **quantifier elimination** problem

$$\exists \vec{x} . F(\vec{a}, \vec{x})$$

where $\vec{a}$ is a parameter

choose((x, y) ⇒ 5 * x + 7 * y == a && x ≤ y)

Corresponding quantifier elimination problem:

$$\exists x \exists y \,.\, 5x + 7y = a \wedge x \leq y$$

Use extended Euclid's algorithm to find particular solution to 5x + 7y = a:

x = 3a
y = -2a

(5,7 are mutually prime, else we get divisibility pre.)

Express general solution of *equations* for x, y using a new variable z:

x = -7z + 3a
y = 5z - 2a

Rewrite *inequations* x ≤ y in terms of z:

5a ≤ 12z

→ z ≥ ceil(5a/12)

Obtain synthesized program:

```
val z = ceil(5*a/12)
val x = -7*z + 3*a
val y = 5*z + -2*a
```

For a = 31:

z = ceil(5*31/12) = 13
x = -7*13 + 3*31 = 2
y = 5*13 – 2*31 = 3

choose((x, y) ⇒ 5 * x + 7 * y == a && x ≤ y && x ≥ 0)

Express general solution of *equations* for x, y using a new variable z:

x = -7z + 3a
y = 5z - 2a

Rewrite *inequations* x ≤ y in terms of z:

z ≥ ceil(5a/12)

Rewrite x ≥ 0:

z ≤ floor(3a/7)

Precondition on a:

ceil(5a/12) ≤ floor(3a/7)

(exact precondition)

Obtain synthesized program:

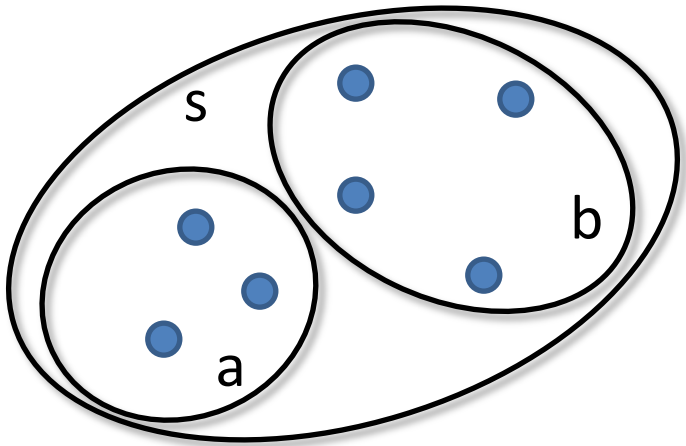
```
assert(ceil(5*a/12) ≤ floor(3*a/7))
val z = ceil(5*a/12)
val x = -7*z + 3*a
val y = 5*z + -2*a
```

With more inequalities we may generate a for loop

Synthesis for sets

```
def splitBalanced[T](s: Set[T]) : (Set[T], Set[T]) =
  choose((a: Set[T], b: Set[T]) ⇒ (
        a union b == s && a intersect b == empty
    && a.size – b.size ≤ 1
    && b.size – a.size ≤ 1
  ))
```

```
def splitBalanced[T](s: Set[T]) : (Set[T], Set[T]) =
  val k = ((s.size + 1)/2).floor
  val t1 = k
  val t2 = s.size – k
  val s1 = take(t1, s)
  val s2 = take(t2, s minus s1)
  (s1, s2)
```



Synthesis for non-linear arithmetic

```
def decomposeOffset(offset: Int, dimension: Int) : (Int, Int) =
  choose((x: Int, y: Int) ⇒ (
    offset == x + dimension * y && 0 ≤ x && x < dimension
  ))
```

- The predicate becomes linear *at run-time*
- Synthesized program must do case analysis on the sign of the input variables
- Some coefficients are computed at run-time

Compile-time warnings

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  choose((h: Int, m: Int, s: Int) ⇒ (
        h * 3600 + m * 60 + s == totalSeconds
    && h ≥ 0 && h < 24
    && m ≥ 0 && m < 60
    && s ≥ 0 && s < 60
  ))
```

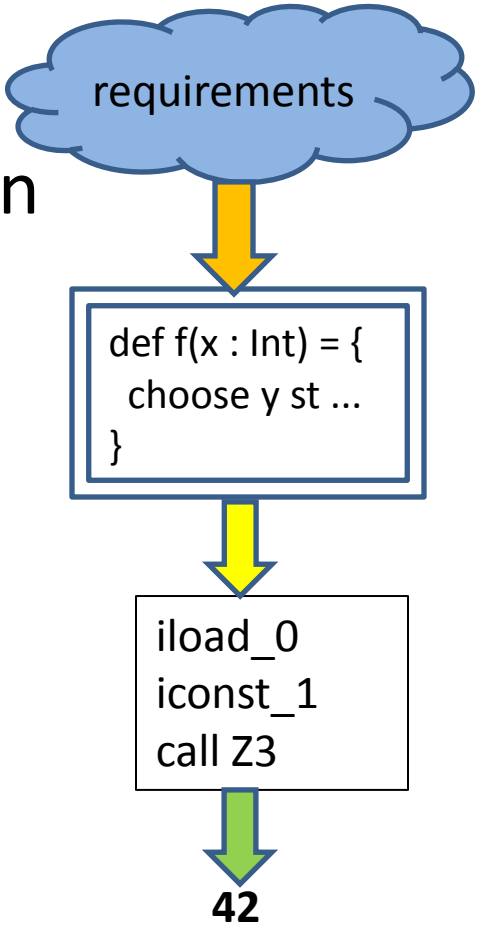
```
Warning: Synthesis predicate is not
satisfiable for variable assignment:
  totalSeconds = 86400
```

Compile-time warnings

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  choose((h: Int, m: Int, s: Int) ⇒ (
        h * 3600 + m * 60 + s == totalSeconds
    && h ≥ 0
    && m ≥ 0 && m ≤ 60
    && s ≥ 0 && s < 60
  ))
```

```
Warning: Synthesis predicate has multiple
solutions for variable assignment:
  totalSeconds = 60
Solution 1: h = 0, m = 0, s = 60
Solution 2: h = 0, m = 1, s = 0
```

Implicit Programming at All Levels



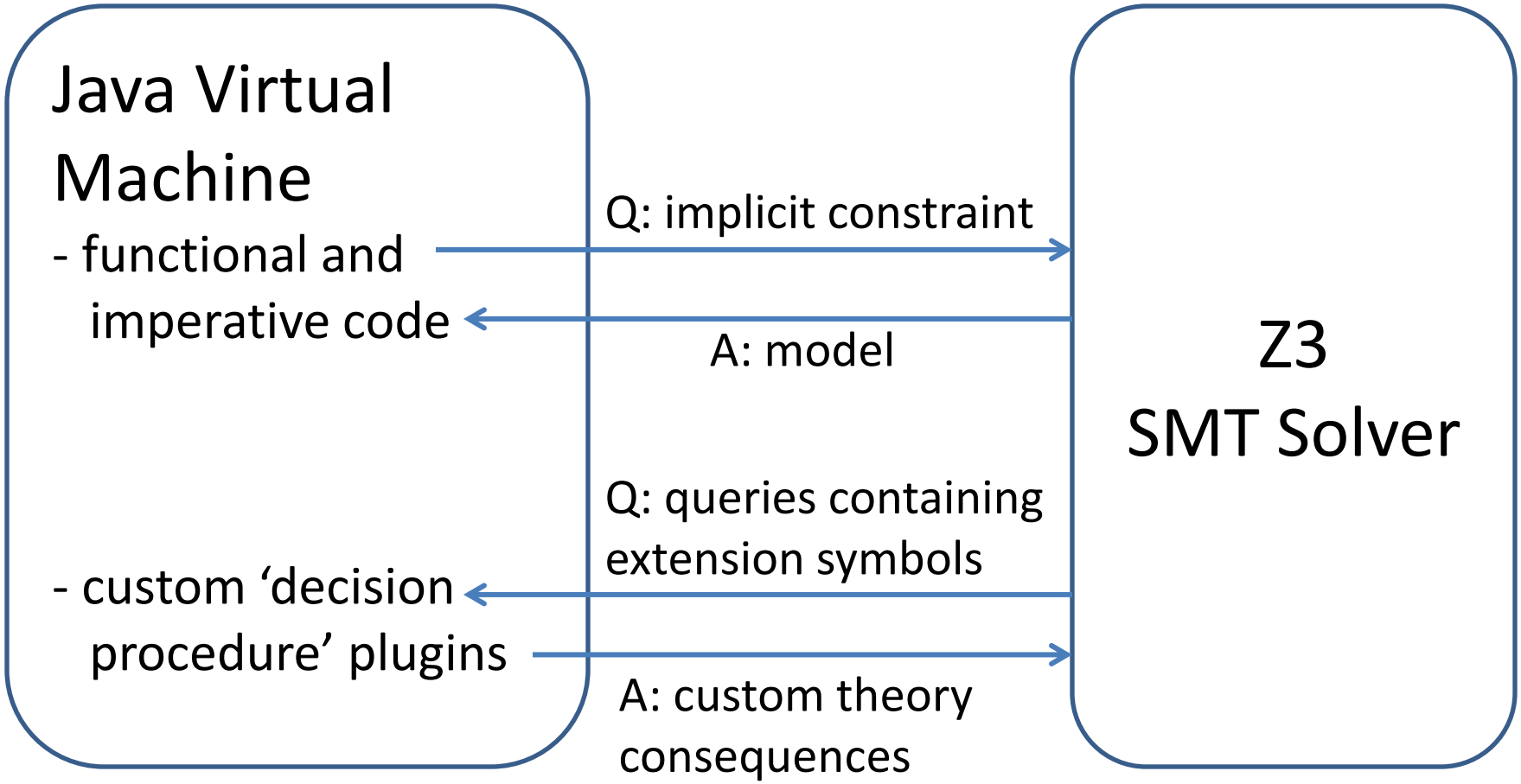
Opportunities for implicit programming in

- **Development** within an IDE
  - **isynth** tool
- **Compilation**
  - **Comfusy** and **RegSy** tools
- **Execution**
  - **Scala^Z3** and **UDITA** tools

I next examine these tools, from last to first, focusing on Compilation

Scala^Z3

Invoking Constraint Solver at Run-Time



with: **Philippe Suter**, **Ali Sinan Köksal**, **Robin Steiger**

Executing **choose** using Z3

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  choose((h: Var[Int], m: Var[Int], s: Var[Int]) ⇒ (
        h * 3600 + m * 60 + s == totalSeconds
    && 0 <= h
    && 0 <= m && m < 60
    && 0 <= s && s < 60    ))
```

will be constant at run-time

syntax tree constructor

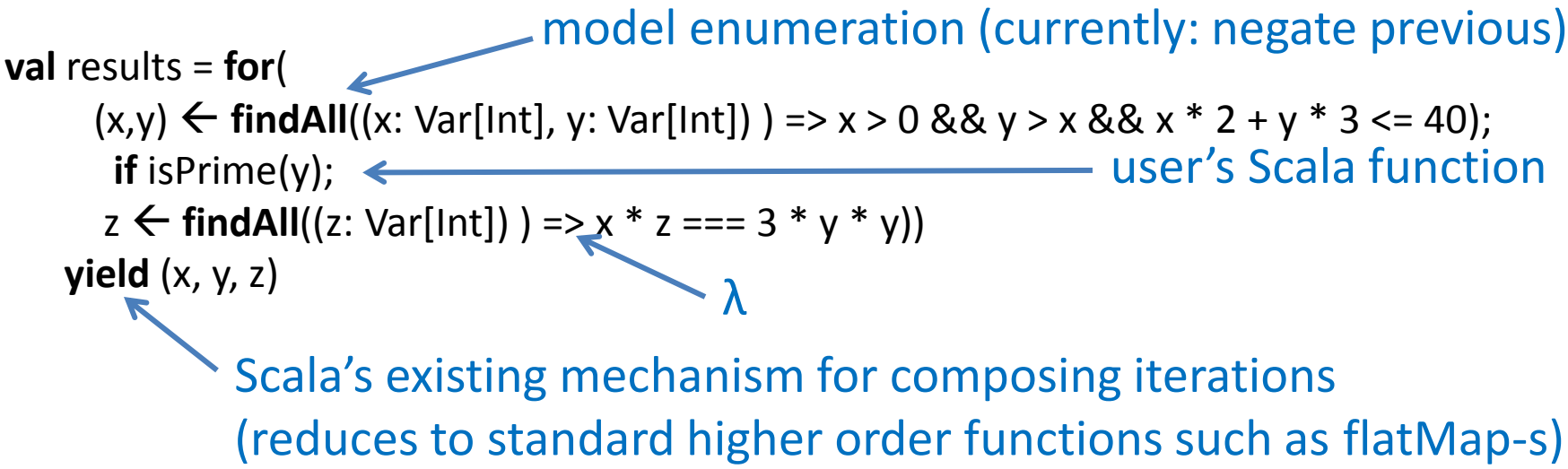
3787 seconds ⟶ 1 hour, 3 mins. and 7 secs.

It works, certainly for constraints within Z3's supported theories

Implemented as a library (jar + z3.so / dll) – no compiler extensions

Programming in Scala^Z3

find triples of integers x, y, z such that x > 0, y > x,
2x + 3y <= 40, $x \cdot z = 3y^2$, and y is prime



Use Scala syntax to construct Z3 syntax trees

a type system prevents certain ill-typed Z3 trees

Obtain models as Scala values

Can also write own plugin decision procedures in Scala

Other Forms of Synthesis

Automata-Theoretic Synthesis

- reactive synthesis
- regular synthesis over unbounded domains

Synthesis of Synchronization Constructs

Quantitative Synthesis

Synthesis from examples:

- **Sumit Gulwani**: *Automating String Processing in Spreadsheets using Input-Output Examples*

Recommended Reading

- Recent *Research Highlights* from the **Communications of the ACM**
  - A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World
  - Retrospective: An Axiomatic Basis for Computer Programming
  - Model Checking: Algorithmic Verification and Debugging
  - Software Model Checking Takes Off
  - Formal Verification of a Realistic Compiler
  - seL4: Formal Verification of an Operating-System Kernel