Your Wish is my Command

Viktor Kunčak

School of Computer and Communication Sciences
Laboratory for Automated Reasoning and Analysis

http://lara.epfl.ch
wish

human effort

Command (program)

compilation

11011001 01011101
11011001 01011101
11011001 01011101
11011001 01011101
11011001 01011101
11011001 01011101
How far can "automatic programming" go beyond "formula translation" towards expressing the wishes even more productively?
Can we further reduce the human effort?

human effort

automatic compilation

specifications in Scala

Simula 1967

LISP 1958

FORTRAN 1957

11011001 01011101
11011001 01011101
11011001 01011101
Example: Sorting
Example: Sorting

Sort a list of cars starting from lowest

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIAT</td>
<td>Punto 1.8 16V Abarth HGT</td>
<td>ab Platz, ohne Aufbereitung und MFH</td>
<td>CHF 6,000.00</td>
</tr>
<tr>
<td>FIAT</td>
<td>Punto 1.4 Dynamic</td>
<td>Das perfekte Auto für angenehmen Fahrspaß</td>
<td>CHF 6,000.00</td>
</tr>
<tr>
<td>FIAT</td>
<td>500 1.2 Lounge</td>
<td></td>
<td>CHF 8,900.00</td>
</tr>
<tr>
<td>FIAT</td>
<td>Punto 1.9 JTD ELX</td>
<td>Sehr sparsames Dieselfahrzeug</td>
<td>CHF 2,900.00</td>
</tr>
<tr>
<td>FIAT</td>
<td>Punto 1.9 JTD ELX</td>
<td>Sehr sparsames Dieselfahrzeug</td>
<td>CHF 2,900.00</td>
</tr>
<tr>
<td>FIAT 500L 1.4 16V Pop Star</td>
<td></td>
<td></td>
<td>CHF 24,140.00</td>
</tr>
</tbody>
</table>
Given a list of numbers, make **this list sorted**

- specification can be reasonably clear, with few alternatives
- many algorithms implement the sorting specification (insertion sort, quick sort, merge sort, external sorts)
Given a list of numbers, make this list sorted.

\[
\begin{array}{c}
8900 \\
6000 \\
24140 \\
2900 \\
\end{array}
\quad \quad
\begin{array}{c}
2900 \\
6000 \\
8900 \\
24140 \\
\end{array}
\]

\[8900 > 6000 \quad \times \quad 2900 < 6000 \quad \checkmark\]
\[6000 < 8900 \quad \checkmark\]
\[8900 < 24140 \quad \checkmark\]

\textbf{def sort_specification(input:List, output:List) : Boolean} = 
\textbf{content(input)}==\textbf{content(output)} \land \textbf{isSorted(output)}

Specification here is a program that checks, for a given input, whether the given output is acceptable.
def C(i: List, o: List): Boolean = // i.e. a constraint
content(i)==content(o) && isSorted(o)

def p(i: List): List =
    // sort i using a sorting algorithm and return the result
How do we bridge this (well-defined) gap between specifications and implementations?
Approaches and Their Guarantees

both specification $C$ and program $p$ are given:

a) Check assertion while program $p$ runs: $C(i, p(i))$

b) Verify whether program always meets the spec: $\forall i. C(i, p(i))$

only specification $C$ is given:

c) Constraint programming: once $i$ is known, find $o$ to satisfy a given constraint: find $o$ such that $C(i, o)$

d) Synthesis: solve $C$ symbolically to obtain program $p$ that is correct by construction, for all inputs: find $p$ such that $\forall i. C(i, p(i))$ i.e. $p \subseteq C$

run-time

compile-time
Runtime Assertion Checking

**a) Check assertion** while program `p` runs: `C(i,p(i))`

```scala
def p(i : List) : List = {
  sort i using a sorting algorithm and return the result
}
ensuring (o ⇒ content(i)==content(o) && isSorted(o))
```

```
def content(lst : List) = lst match {
  case Nil() ⇒ Set.empty
  case Cons(x, xs) ⇒ Set(x) ++ content(xs)
}
def isSorted(lst : List) = lst match {
  case Nil() ⇒ true
  case Cons(_, Nil()) ⇒ true
  case Cons(x, Cons(y, ys)) ⇒
    x < y && isSorted(Cons(y, ys))
}
```

Already works in Scala!
Key design decision: `constraints are programs`

Must come up with example `i`-values
(So, this is a way to do testing.)

Can we give stronger guarantees?
⇒ prove postcondition always true
b) **Verify** that program always meets spec: $\forall i. C(i, p(i))$

```scala
def p(i : List) : List = {
    sort i using a sorting algorithm and return the result
}
```

```scala
def content(lst : List) = lst match {
    case Nil() ⇒ Set.empty
    case Cons(x, xs) ⇒ Set(x) ++ content(xs)
}
```

```scala
def isSorted(lst : List) = lst match {
    case Nil() ⇒ true
    case Cons(_, Nil()) ⇒ true
    case Cons(x, Cons(y, ys)) ⇒
        x < y && isSorted(Cons(y, ys))
}
```

Type in a Scala program and watch it verified

- Proof of $\forall i. C(i, p(i))$
- Input $i$ such that not $C(i, p(i))$
Insertion Sort Verified as You Type It

```haskell
def sortedIns(e: Int, l: List): List = {
  require(isSorted(l))
  l match {
    case Nil() => Cons(e,Nil())
    case Cons(x,xs) =>
      if (x <= e) Cons(x,sortedIns(e, xs)) else Cons(e, l)
  }
} ensuring(res => contents(res) == contents(l) ++ Set(e)
  && isSorted(res)
  && size(res) == size(l) + 1

/* Insertion sort yields a sorted list of
  same size and content as the input list */

def sort(l: List): List = (l match {
  case Nil() => Nil()
  case Cons(x,xs) => sortedIns(x, sort(xs))
}) ensuring(res => contents(res) == contents(l)
  && isSorted(res)
  && size(res) == size(l))
```

Web interface: [http://lara.epfl.ch/leon](http://lara.epfl.ch/leon)
Reported Counterexample in Case of a Bug

Leon

```haskell
31 def sortedIns(e: Int, l: List): List = {
32 require(isSorted(l))
33 1 match {
34    case Nil() => Cons(e,Nil())
35    case Cons(x,xs) => Cons(x,sortedIns(e, xs))
36  }
37 } ensuring(res => contents(res) == contents(l) ++ Set(e)
38     && isSorted(res)
39     && size(res) == size(l) + 1
40 /* Insertion sort yields a sorted list of the same size and content as the input */
```

Verification

Leon verifies the validity all the verification conditions found in the selected function.

Invalid!

<table>
<thead>
<tr>
<th>Function</th>
<th>Kind</th>
<th>Result</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortedIns</td>
<td>precond.</td>
<td>✓ valid</td>
<td>0.012</td>
</tr>
<tr>
<td>sortedIns</td>
<td>postcond.</td>
<td>✗ invalid</td>
<td>0.051</td>
</tr>
</tbody>
</table>

The following valuation violates the VC:

```
1 := Cons(8946, Nil())
e := 8945
```
## Verification of Functional and Imperative Scala Code

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LoC</th>
<th>V/I</th>
<th>#VCs</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Imperative</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListOperations</td>
<td>146</td>
<td>6/1</td>
<td>16</td>
<td>0.62</td>
</tr>
<tr>
<td>AssociativeList</td>
<td>98</td>
<td>3/1</td>
<td>9</td>
<td>0.80</td>
</tr>
<tr>
<td>AmortizedQueue</td>
<td>128</td>
<td>10/1</td>
<td>21</td>
<td>2.57</td>
</tr>
<tr>
<td>SumAndMax</td>
<td>36</td>
<td>2/0</td>
<td>2</td>
<td>0.21</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>84</td>
<td>4/1</td>
<td>8</td>
<td>0.58</td>
</tr>
<tr>
<td><strong>Functional</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListOperations</td>
<td>107</td>
<td>12/0</td>
<td>11</td>
<td>0.43</td>
</tr>
<tr>
<td>AssociativeList</td>
<td>50</td>
<td>4/0</td>
<td>5</td>
<td>0.43</td>
</tr>
<tr>
<td>AmortizedQueue</td>
<td>114</td>
<td>13/0</td>
<td>18</td>
<td>1.56</td>
</tr>
<tr>
<td>SumAndMax</td>
<td>45</td>
<td>4/0</td>
<td>7</td>
<td>0.23</td>
</tr>
<tr>
<td>RedBlackTree</td>
<td>117</td>
<td>7/1</td>
<td>10</td>
<td>1.87</td>
</tr>
<tr>
<td>PropositionalLogic</td>
<td>81</td>
<td>6/1</td>
<td>9</td>
<td>0.72</td>
</tr>
<tr>
<td>SearchLinkedList</td>
<td>38</td>
<td>3/0</td>
<td>2</td>
<td>0.21</td>
</tr>
<tr>
<td>Sorting</td>
<td>175</td>
<td>13/0</td>
<td>17</td>
<td>0.48</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1219</td>
<td>87/6</td>
<td>135</td>
<td>10.71</td>
</tr>
</tbody>
</table>

Regis Blanc

Etienne Kneuss

Philippe Suter
Automated Verification: How

1) Induction: assume and prove specification:

```python
def size(l : List) : Int = (l match {
  case Nil() ⇒ 0
  case Cons(_, xs) ⇒ 1 + size(xs)
}) ensuring(res ⇒ (res ≥ 0))
```

Verification conditions:
1) $0 \geq 0$
2) if $res1 \geq 0$ then $1 + res1 \geq 0$

Eliminates recursive function being verified.

2) Algebraic reasoning for formulas over theories:
   – arithmetic, sets, lists, trees

Technology: Satisfiability Modulo Theories (SMT)
   SAT solver + decision procedures for theories
   – Leonardo de Moura (Z3)
   – Andrew Reynolds (CVC4), 18 September 14:15 (Wednesday)
Recursive functions inside specifications

```scala
def sortedIns(e: Int, l: List): List = {
  // insertion into a sorted list
  require(isSorted(l))
  l match {
    case Nil() ⇒ Cons(e, Nil())
    case Cons(x, xs) ⇒ if (x ≤ e) Cons(x, sortedIns(e, xs)) else Cons(e, l)
  }
} ensuring (res ⇒ contents(res) == contents(l) ++ Set(e))
  // contents(l) U {e}
```

Theorem provers for recursive functions?

If content(res1) == content(xs) U {e} then
  if (x ≤ e) then content(Cons(x, res1)) == content(Cons(x, xs)) U {e}
  else content(Cons(e, l)) == content(l) U {e}
...
Reasoning about abstraction functions

Adding all recursive functions \( f : \text{Tree} \rightarrow \text{Tree} \)

\(-\) undecidable 😞 Turing-complete formalism

Consider abstraction functions: \( m : \text{Tree} \rightarrow \mathbb{N} \)

\(-\) \( m \) defined by simple structural recursion on trees

\[
m == \text{fold}(\text{leaf\_const}, \text{combination\_function})
\]

\[
\text{size} == \text{fold}(0, _ + _ + _ )
\]

\[
\text{content} == \text{fold}({}, _ \cup \{ _ \} \cup _)
\]

\(-\) sufficiently surjective, implies \( \text{card}(m^{-1}(n)) \rightarrow \infty \)

Fair function unfolding acts as a decision procedure for such \( m \) 😊

Intuition: after unfolding, innermost calls can be left un-interpreted

Basis of the Leon verifier (along with induction and Z3 encoding)

→ Philippe Suter (PhD 2012, now IBM Research US): POPL’10, SAS’11
Constraint Solvers on top of NASA’s Model Checker for Java (JPF)

Generating not only one, but many values, using delayed non-determinism and heap symmetry detection

**Application:** generate tests to exercise program behavior

*Test generation through programming in UDITA. ICSE 2010*

- Found correctness bugs in existing refactoring implementations of IDE tools Eclipse and Netbeans
- Differences in accepted programs in Eclipse compilers vs javac

Milos Gligoric  Tihomir Gvero  Vilas Jagannath  Sarfraz Khurshid  Darko Marinov
Reasoning about New Theories

Our sorting spec using **sets** allows mapping

\[ \text{List}(1,3,2,3,2) \rightarrow \text{List}(1,1,1,2,3) \]

Precise specification needs to use **multisets** (bags)

\[ \{|1, 1, 2, 3|\} \cup \{|2|\} = \{|1, 1, 2, 2, 3|\} \]

**Algorithm for:** given an expression with operations on multisets, are there values for which expression is true?

Previously: algorithms in NEXPTIME or worse

Our result: algorithm running in NP (NP-hardness is easy)

- enables verification of a larger class of programs

Method: encode problem in integer linear arithmetic, use semilinear set bounds and integer Caratheodory theorem

Ruzica Piskac (PhD 2011): *CAV’08, CSL’08, VMCAI’08*
Can we sort planets by distance?

Gap between **floating points** and **reality**
- input measurement error
- floating-point round-off error
- numerical method error
- all other sources of bugs

x < y need not mean x* < y*

Automated verification tools to compute upper error bound

Applied to code fragments for
- embedded systems (car, train)
- physics simulations

OOPSLA'11, RV'12, EMSOFT'13
Example: Where is the Moon?

Geneva observatory’s software to compute position of the Moon
   – rewritten from Python to Java (great performance)
   – different result computed in some cases!

Which digits can we trust, if any?

Results for date 2012-2-10:

Java:      −2h 36m 26.779661250681812
Python:    −2d 36m 26.77966125074235
Example: Where is the Moon?

Geneva observatory’s software to compute position of the Moon
– rewritten from Python to Java (great performance)
– different result computed in some cases!

Which digits can we trust, if any?

Results for date 2012-2-10:

Java: -2h 36m 26.779661250681812
Python: -2d 36m 26.77966125074235
QuadDouble: -2h 36m 26.7796612340577158626981678...

AffineFloat: -2h 36m \underline{26.779661250681812} \ (3.9991e\text{-07})

\[\uparrow\text{provably correct digits}\]
\[\uparrow\text{rigorous upper bound on error}\]
Beyond Functional: Verifying Imperative C and Concurrent Systems

- Key idea: encode program and properties into recursive logical constraints (Horn clauses)
- Decouple two non-trivial tasks:
  - generation of constraints (language semantics, modeling approach)
  - solving of constraints (new verification algorithms)

ATVA’12, CAV’13
Hossein Hojjat, PhD 2013
w/ Radu Iosif, Filip Konečný, Philipp Ruemmer
Distributed Software – Hardest of All

Perform execution steering of software while it runs, using a continuously running model checker (CrystalBall).

Prove correctness of distributed algorithms in a modular way using interactive theorem provers and model checkers.

Maysam Yabandeh
Qatar CRI
NSDI'09, TOCS'10

Dejan Kostić
IMDEA Networks

Giuliano Losa
Speculative Linearizability, PLDI 2012

Rachid Guerraoui
Approaches and Their Guarantees

Was your wish your command?

a) **Check assertion** while program $p$ runs: $C(i,p(i))$

b) **Verify** that program always meets spec:

   $\forall i. C(i,p(i))$

Your wish is my command!

c) **Constraint programming**: once $i$ is known, find $o$ to satisfy a given constraint: find $o$ such that $C(i,o)$

   - run-time

d) **Synthesis**: solve $C$ symbolically to obtain program $p$ that is correct by construction, for all inputs: find $p$ such that $\forall i.C(i,p(i))$ i.e. $p \subseteq C$

   - compile-time

Your wish is my command!
Approaches and Their Guarantees

both specification $C$ and program $p$ are given:

a) **Check assertion** while program $p$ runs: $C(i,p(i))$

b) **Verify** that program always meets spec:

$$\forall i. \ C(i,p(i))$$

don't know $i$

only specification $C$ is given:

c) **Constraint programming**:

- once $i$ is known, find $o$ to satisfy a given constraint: $\text{find } o \text{ such that } C(i,o)$

- **run-time**

d) **Synthesis**:

- solve $C$ symbolically to obtain program $p$ that is correct by construction, for all inputs: $\text{find } p \text{ such that } \forall i. C(i,p(i))$

- $p \subseteq C$

don't know $i$

- **compile-time**
Programming without Programs

c) Constraint programming: find a value that satisfies a given constraint: \textbf{find } o \textbf{ such that } C(i,o)

**Method**: use verification technology, try to prove that no such \( o \) exists, report counter-examples!

<table>
<thead>
<tr>
<th>size</th>
<th>list add</th>
<th>list remove</th>
<th>RBT add</th>
<th>RBT remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.07</td>
<td>0.02</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>1</td>
<td>0.08</td>
<td>0.02</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>2</td>
<td>0.12</td>
<td>0.05</td>
<td>0.14</td>
<td>0.09</td>
</tr>
<tr>
<td>3</td>
<td>0.16</td>
<td>0.10</td>
<td>0.55</td>
<td>0.41</td>
</tr>
<tr>
<td>4</td>
<td>0.24</td>
<td>0.18</td>
<td>0.66</td>
<td>0.76</td>
</tr>
<tr>
<td>5</td>
<td>0.39</td>
<td>0.38</td>
<td>1.07</td>
<td>0.91</td>
</tr>
<tr>
<td>6</td>
<td>0.55</td>
<td>0.45</td>
<td>1.51</td>
<td>1.56</td>
</tr>
<tr>
<td>7</td>
<td>0.97</td>
<td>0.67</td>
<td>9.32</td>
<td>13.09</td>
</tr>
<tr>
<td>8</td>
<td>1.48</td>
<td>1.09</td>
<td>11.13</td>
<td>18.80</td>
</tr>
<tr>
<td>9</td>
<td>2.27</td>
<td>1.80</td>
<td>24.49</td>
<td>25.79</td>
</tr>
<tr>
<td>10</td>
<td>3.32</td>
<td>2.22</td>
<td>11.51</td>
<td>20.55</td>
</tr>
</tbody>
</table>

Constraints as Control, POPL 2012
Extension of Scala with constraint programming
Chapter 12 showed that a binary search tree of height $h$ can support any of the basic dynamic-set operations—such as **Search**, **Predecessor**, **Successor**, **Minimum**, **Maximum**, **Insert**, and **Delete**—in $O(h)$ time. Thus, the set operations are fast if the height of the search tree is small. If its height is large, however, the set operations may run no faster than with a linked list. Red-black trees are one of many search-tree schemes that are “balanced” in order to guarantee that basic dynamic-set operations take $O(\lg n)$ time in the worst case.

### 13.1 Properties of red-black trees

A **red-black tree** is a binary search tree with one extra bit of storage per node: its **color**, which can be either **red** or **black**. By constraining the node colors on any simple path from the root to a leaf, red-black trees ensure that no such path is more than twice as long as any other, so that the tree is approximately balanced.

Each node of the tree now contains the attributes **color**, **key**, **left**, **right**, and **p**. If a child or the parent of a node does not exist, the corresponding pointer attribute of the node contains the value NIL. We shall regard these NILs as being pointers to leaves (external nodes) of the binary search tree and the normal, key-bearing nodes as being internal nodes of the tree.

A red-black tree is a binary tree that satisfies the following **red-black properties**:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.
Sorting a List Using Specifications

```scala
def content(lst : List) = lst match {
  case Nil() ⇒ Set.empty
  case Cons(x, xs) ⇒ Set(x) ++ content(xs)
}
def isSorted(lst : List) = lst match {
  case Nil() ⇒ true
  case Cons(_, Nil()) ⇒ true
  case Cons(x, Cons(y, ys)) ⇒ x < y && isSorted(Cons(y,ys))
}

((l : List) ⇒ isSorted(lst) && content(lst) == Set(0, 1, -3))
solve

> Cons(-3, Cons(0, Cons(1, Nil()))))
```
Implicit Programming (ERC project)

specification (constraint) *implicit*

\[ x^2 + y^2 = 1 \]

implementation (function) *explicit*

\[ y = \sqrt{1-x^2} \]

i is a propositional formula and o is an assignment making i true - P

compute a satisfying assignment for i (SAT solver implementation) - NP
Approaches and Their Guarantees

both specification $C$ and program $p$ are given:

a) **Check assertion** while program $p$ runs: $C(i,p(i))$

b) **Verify** that program always meets spec:
   $\forall i. C(i,p(i))$

only specification $C$ is given:

c) **Constraint programming**: once $i$ is known, find $o$ to satisfy a given constraint: find $o$ such that $C(i,o)$

run-time

d) **Synthesis**: solve $C$ symbolically to obtain program $p$ that is correct by construction, for all inputs: find $p$ such that $\forall i. C(i,p(i))$ i.e. $p \subseteq C$

compile-time
Synthesis for Theories

\[ 3i + 2o = 13 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quoi
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  
))

close to a wish

could infer from types

def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =

val t1 = totalSeconds div 3600
val t2 = totalSeconds - 3600 * t1
val t3 = t2 div 60
val t4 = totalSeconds - 3600 * t1 - 60 * t3
(t1, t3, t4)
Synthesis for sets (BAPA)

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

```scala
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)
```
Automata-Based Synthesis for Arithmetic

Given a constraint, generate finite-state automaton that reads input bits and directly emits result bits.

- Result does not depend on the syntax of input formula but only on the relation that the formula defines
- Data complexity for synthesized code: always linear in input
- Modular arithmetic and bitwise operators: can synthesize bit manipulations for unbounded number of bits, uniformly
- Supports quantified constraints
  - including optimization constraints: find best value

FMCAD 2010, IJCAR 2012
Foreword to the Research Highlights Article in the Communications of the ACM

I predict that as we identify more such restricted languages and integrate them into general-purpose (Turing-complete) languages, we will make programming more productive and programs more reliable.

Rastislav Bodik
Professor, UC Berkeley

Upcoming talk on 27 September 2013
Partial Specs + Interaction to Synthesize Expressions

http://lara.epfl.ch/w/insynth

Extend **type inhabitation** with
- enumeration of all inhabitants
- quantitative **ranking** of inhabitants
- learning ranking from corpus of code

```
import java.io._

object Main {
  def main(args: Array[String]) = {
    var body = "email.txt"
    var sig = "signature.txt"

    var inStream: SequenceInputStream = |

    var eof: Boolean = false;
    var byteCount: Int = 0;
    while (!eof) {
      var c: Int = inStream.read();
      if (c == -1) {
        eof = true;
      } else {
        System.out.print(c.toChar);
        byteCount+=1;
      }
    }
    System.out.println(byteCount + " bytes were read");
    inStream.close();
  }
```
Collaboration with LAMP

Martin Odersky

Miguel Garcia

Hubert Plociniczak

Lukas Rytz

Jovanovic Vojin
Combining Approaches: Synthesis in Leon

```
def insertSorted(lst : List, v: Int): List = {
    require(isSorted(lst))
    choose { (r: List) ⇒
        isSorted(r) && content(r) == content(lst) ++ Set(v) } }

def sort(lst : List): List = choose { (r: List) ⇒
    isSorted(r) && content(r) == content(lst) }
```

http://lara.epfl.ch/w/leon

OOPSLA 2013:
Synthesis Modulo Recursive Functions

Etienne Kneuss
Ivan Kuraj
Philippe Suter
### Results for Synthesis in Leon

<table>
<thead>
<tr>
<th>Operation</th>
<th>Size</th>
<th>Calls</th>
<th>Proof</th>
<th>sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>List.Insert</td>
<td>3</td>
<td>0</td>
<td>✓</td>
<td>0.6</td>
</tr>
<tr>
<td>List.Delete</td>
<td>19</td>
<td>1</td>
<td>✓</td>
<td>1.8</td>
</tr>
<tr>
<td>List.Union</td>
<td>12</td>
<td>1</td>
<td>✓</td>
<td>2.1</td>
</tr>
<tr>
<td>List.Diff</td>
<td>12</td>
<td>2</td>
<td>✓</td>
<td>7.6</td>
</tr>
<tr>
<td>List.Split</td>
<td>27</td>
<td>1</td>
<td>✓</td>
<td>9.3</td>
</tr>
<tr>
<td>SortedList.Insert</td>
<td>34</td>
<td>1</td>
<td>✓</td>
<td>9.9</td>
</tr>
<tr>
<td>SortedList.InsertAlways</td>
<td>36</td>
<td>1</td>
<td>✓</td>
<td>7.2</td>
</tr>
<tr>
<td>SortedList.Delete</td>
<td>23</td>
<td>1</td>
<td>✓</td>
<td>4.1</td>
</tr>
<tr>
<td>SortedList.Union</td>
<td>19</td>
<td>2</td>
<td>✓</td>
<td>4.5</td>
</tr>
<tr>
<td>SortedList.Diff</td>
<td>13</td>
<td>2</td>
<td>✓</td>
<td>4.0</td>
</tr>
<tr>
<td>SortedList.InsertionSort</td>
<td>10</td>
<td>2</td>
<td>✓</td>
<td>4.2</td>
</tr>
<tr>
<td>SortedList.MergeSort</td>
<td>17</td>
<td>4</td>
<td>✓</td>
<td>14.3</td>
</tr>
<tr>
<td>StrictSortedList.Insert</td>
<td>34</td>
<td>1</td>
<td>✓</td>
<td>14.1</td>
</tr>
<tr>
<td>StrictSortedList.Delete</td>
<td>21</td>
<td>1</td>
<td></td>
<td>15.1</td>
</tr>
<tr>
<td>StrictSortedList.Union</td>
<td>19</td>
<td>2</td>
<td>✓</td>
<td>3</td>
</tr>
<tr>
<td>UnaryNumerals.Add</td>
<td>11</td>
<td>1</td>
<td>✓</td>
<td>1.3</td>
</tr>
<tr>
<td>UnaryNumerals.Distinct</td>
<td>12</td>
<td>0</td>
<td>✓</td>
<td>1.1</td>
</tr>
<tr>
<td>UnaryNumerals.Mult</td>
<td>12</td>
<td>1</td>
<td>✓</td>
<td>2.7</td>
</tr>
<tr>
<td>BatchedQueue.Checkf</td>
<td>14</td>
<td>4</td>
<td>✓</td>
<td>7.4</td>
</tr>
<tr>
<td>BatchedQueue.Snoc</td>
<td>7</td>
<td>2</td>
<td>✓</td>
<td>3.7</td>
</tr>
<tr>
<td>AddressBook.Make</td>
<td>50</td>
<td>14</td>
<td></td>
<td>8.8</td>
</tr>
<tr>
<td>AddressBook.MakeHelpers</td>
<td>21</td>
<td>5</td>
<td></td>
<td>4.9</td>
</tr>
<tr>
<td>AddressBook.Merge</td>
<td>11</td>
<td>3</td>
<td></td>
<td>8.9</td>
</tr>
</tbody>
</table>

**Techniques used:**
- Leon’s verification capabilities
- synthesis for theory of trees
- recursion schemas
- case splitting
- symbolic exploration of the space of programs
- synthesis based on type inhabitation
- fast falsification using previous counterexamples
- learning conditional expressions
- cost-based search over possible synthesis steps
From In-Memory to External Sorting

Transform **functional specification** of data base operations into **algorithms** that work when not all data fits into memory (sort -> external sort)  SIGMOD’13

Approach:

- transformation rules for list algebra
- exploration of equivalent algorithms through performance estimation w/ non-linear constraint solving

Ioannis Klonatos  Christoph Koch  Andres Nötzli  Andrej Spielmann
Can we help with designing specification themselves, to make programming accessible to non-experts?
Programming by Demonstration

Describe functionality by demonstrating and modifying behaviors while the program runs

– demonstrate desired actions by moving back in time and referring to past events
– system generalizes demonstrations into rules

http://www.youtube.com/watch?v=bErU--8GRsQ
Try "Pong Designer" in Android Play Store

Mikael Mayer and Lomig Mégard

SPLASH Onward'13
your wish is my command

or some such gibberish. To go a step further he would like to write \( \sum_{aij \cdot bj} \) instead of the fairly involved set of instructions corresponding to this expression. In fact a programmer might not be considered too

111

1954

IBM 701 SPEEDCODING SYSTEM

unreasonable if he were willing only to produce the formulas for the numerical solution of his problem and perhaps a plan showing how the data was to be moved from one storage hierarchy to another and then demand that the machine produce the results for his problem. No doubt if he were too insistent next week about this sort of thing he would be subject to psychiatric observation. However, next year he might be taken more seriously.
Thank you for listening!