Viktor Kunčak

Laboratory for Automated Reasoning and Analysis

http://lara.epfl.ch/w/
enable people to easily program and construct reliable and sophisticated systems

Enable: through building tools such as compilers, verifiers, synthesizers

People: today for developers, tomorrow maybe for everyone

Easily: use high-level languages and program synthesis

Reliable systems: software in general, verified using automated proofs

Sophisticated systems: make developers more productive and use machine learning to make them more productive
Past/graduating students jobs of former members

Tenure Track AssistantProfessor (3)
Postdoc (3)
CTO of own company
Research lab (IBM research)
Google Zurich
CreditSuisse
wish

human effort

Command(program)

automatic compilation

11011001 01011101
11011001
How far can "automatic programming" go beyond formula translation (expression compilation)?
Can we lower this large gap that still remains?
Given a list of numbers, make **this list** sorted.

Sorting as a Wish

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>8900</td>
<td>2900</td>
</tr>
<tr>
<td>6000</td>
<td>6000</td>
</tr>
<tr>
<td>24140</td>
<td>8900</td>
</tr>
<tr>
<td>2900</td>
<td>24140</td>
</tr>
</tbody>
</table>

8900 > 6000 ✗

2900 < 6000 ✓

6000 < 8900 ✓

8900 < 24140 ✓

Sorting is a well-known problem, but illustrates the difficulty:

- specification is clear, not too many possibilities
- many algorithms that implement the sorting specification (insertion sort, quick sort, merge sort, external sorts)
Given a list of numbers, make this list sorted.

Specification is a program that checks, for a given input, whether the given output is acceptable.

```python
def sort_specification(input: List, output: List) -> Boolean:
    return content(input) == content(output) and isSorted(output)
```
Specification vs Implementation

\[
\text{def } \text{sort specification}(\text{input:List, output:List}) : \text{Boolean} = \\
\text{content}(\text{input}) == \text{content}(\text{output}) \text{ && isSorted}(\text{output})
\]

\[
\text{def } \text{sort implementation}(\text{input:List}) : \text{List} = \\
\text{sorting algorithm}
\]
Synthesis using **Leon** system

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

\[\text{def sort(lst : List): List} = \text{choose (}(r: \text{List}) \Rightarrow
\text{isSorted}(r) \&\& \text{content}(r) == \text{content}(\text{lst})\))\]

If we first specify insertion into list, we generate insertion sort

If we specify merge and a hint => merge sort

More examples and results: OOPSLA 2013
Type-Driven Synthesis within an IDE

```java
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;
        System.out.print(c.toChar);
        byteCount+=1;
      } else {
        System.out.print(c.toChar);
        byteCount+=1;
      }
    }
    System.out.println(byteCount + " bytes were read");
    inStream.close();
  }
}
```
Underlying technique: Constraint Solving

• Checking assertions while program runs: $C(x,y)$
• Verifying that program meets spec: $\forall x \ y. \ C(x,y)$
• Falsifying: producing a counterexample when verification fails:
  find $x,y$ such that $\neg \ C(x,y)$
• Computing: find any (or, least) value that satisfies a given constraint:
  find $y$ such that $C(x,y)$
• Generating tests to exercise program behavior:
  enumerate/sample $y$ such that $C(x,y)$
• Synthesis: solving symbolically for all inputs to obtain program that is correct by construction
  find $f$ such that $\forall x. \ C(x,f(x))$