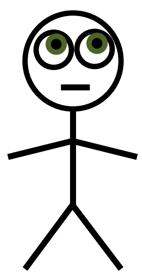
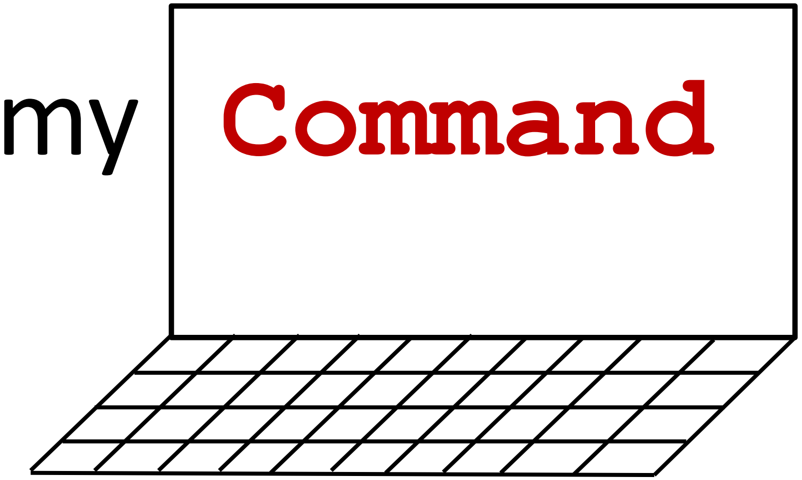
Your Wish is my Command



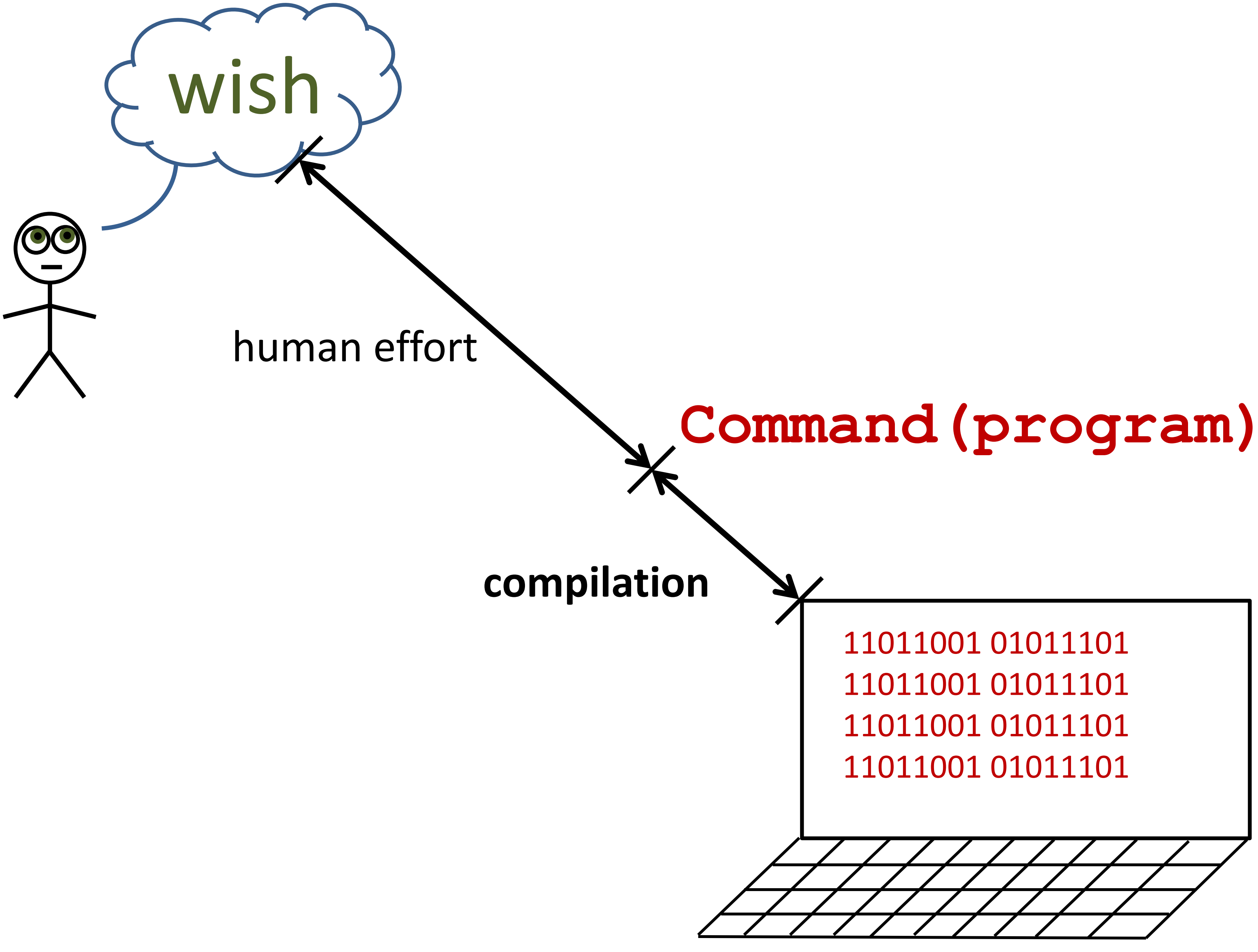


Viktor Kunčak

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

NATIONAL PHYSICAL LABORATORY

TEDDINGTON, MIDDLESEX, ENGLAND

PAPER 2-3

AUTOMATIC PROGRAMMING

PROPERTIES AND PERFORMANCE OF

FORTRAN SYSTEMS I AND II

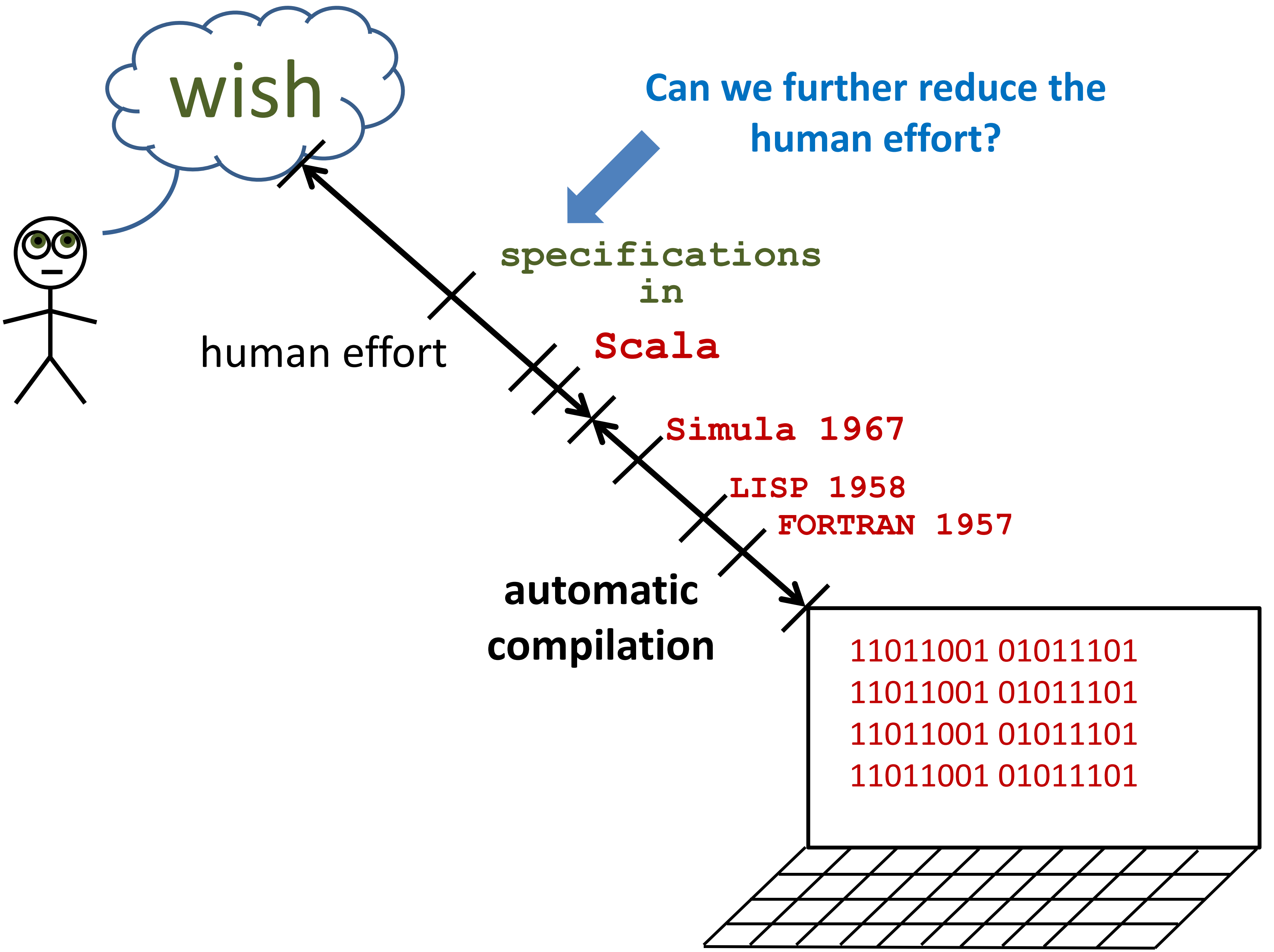
FORmula TRANslation

by

J. W. BACKUS

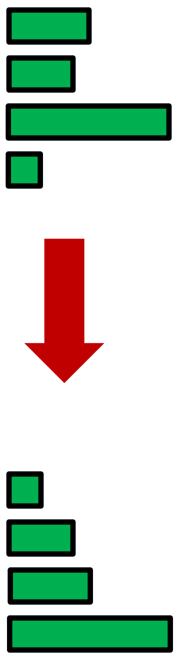
To be presented at a Symposium on
The Mechanization of Thought Processes,
which will be held at the National Physical Laboratory, Teddington, Middlesex, from 24th-27th November 1958. The papers and the discussions are to be published by H.M.S.O. in the Proceedings of the Symposium. This paper should not be reproduced without the permission of the author and of the Secretary, National Physical Laboratory.

How far can "automatic programming" go beyond "formula translation" towards expressing the wishes even more productively?

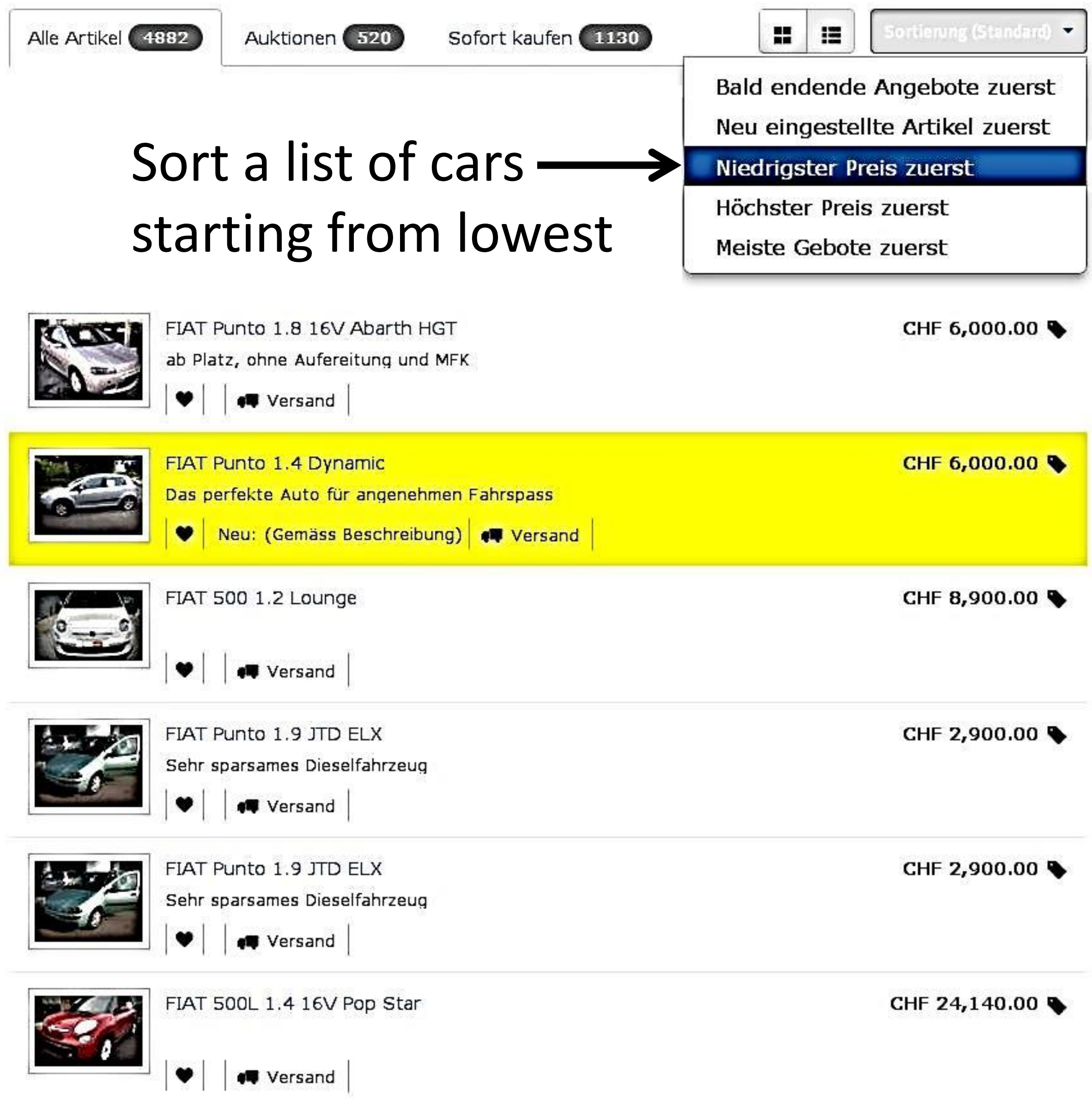

wish
Can we further reduce the human effort?
specifications in
Scala
human effort
Simula 1967
LISP 1958
FORTRAN 1957
automatic compilation
11011001 01011101
11011001 01011101
11011001 01011101
11011001 01011101

Example: Sorting

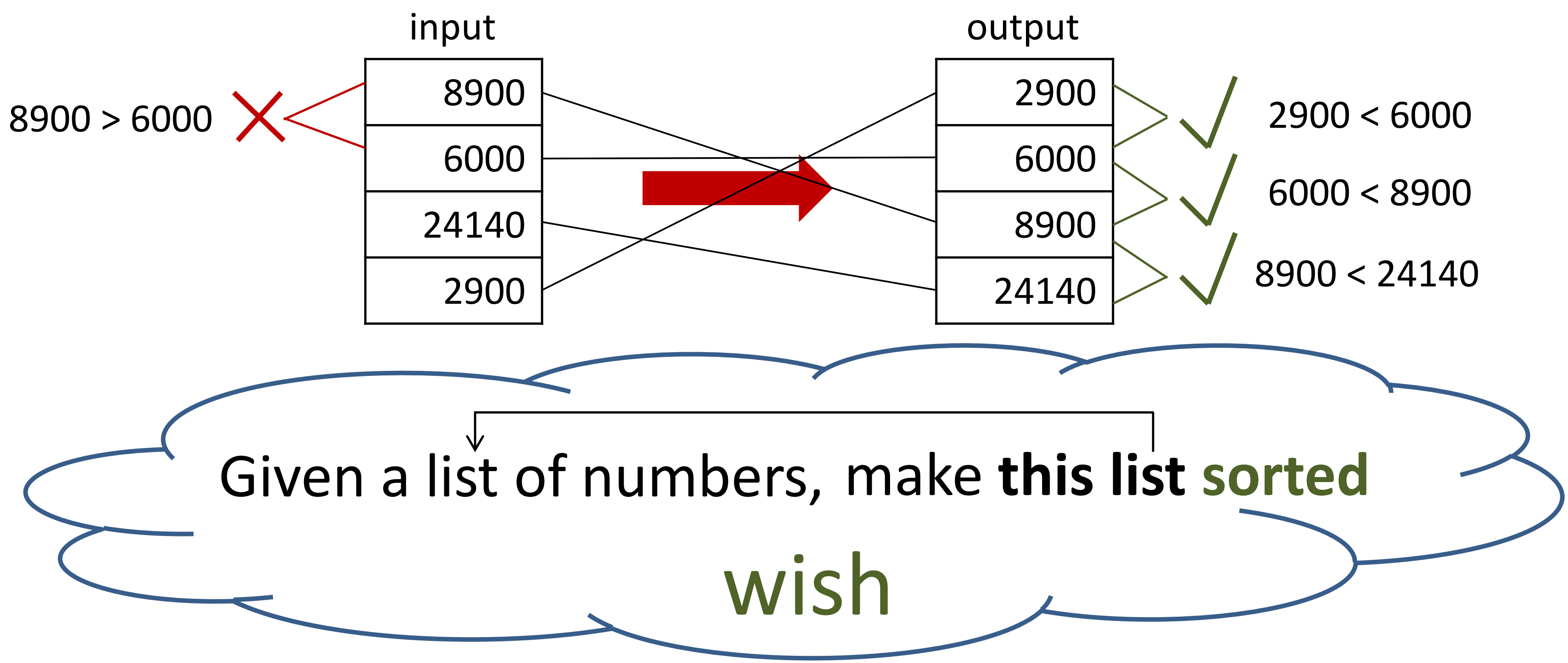
Example: Sorting



Sort a list of cars starting from lowest

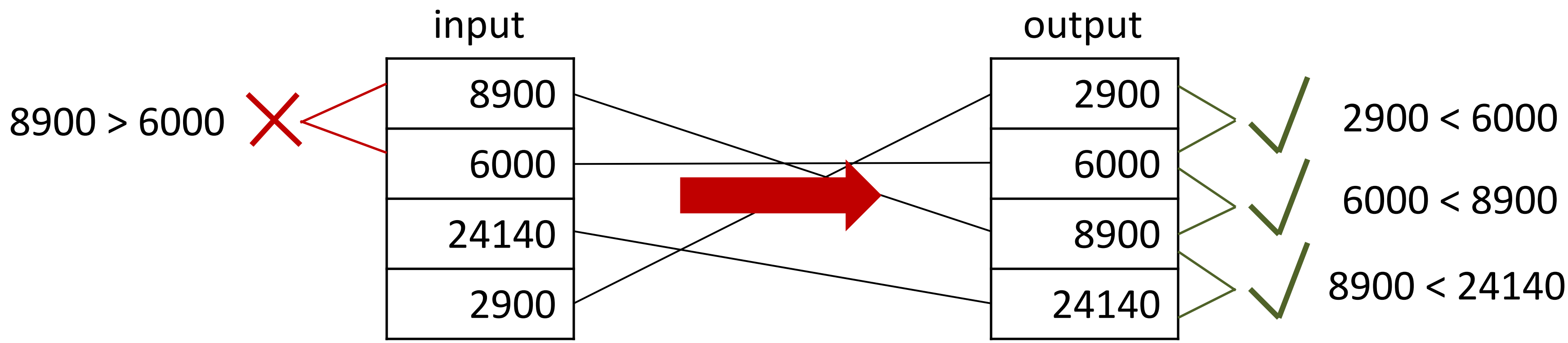


Sorting as a Wish



- specification can be reasonably clear, with few alternatives
- many algorithms implement the sorting specification (insertion sort, quick sort, merge sort, external sorts)

Sorting Specification as a Program



Given a list of numbers, make **this list** **sorted**

wish

```
def sort_specification(input:List, output:List) : Boolean =
  content(input)==content(output)  &&  isSorted(output)
```

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

Specification vs Implementation

def C(i : List, o : List) : **Boolean** =   // i.e. a constraint
  content(i)==content(o) && isSorted(o)

more behaviors

true / false

specification

MIND THE GAP

input
8900
6000
24140
2900

U

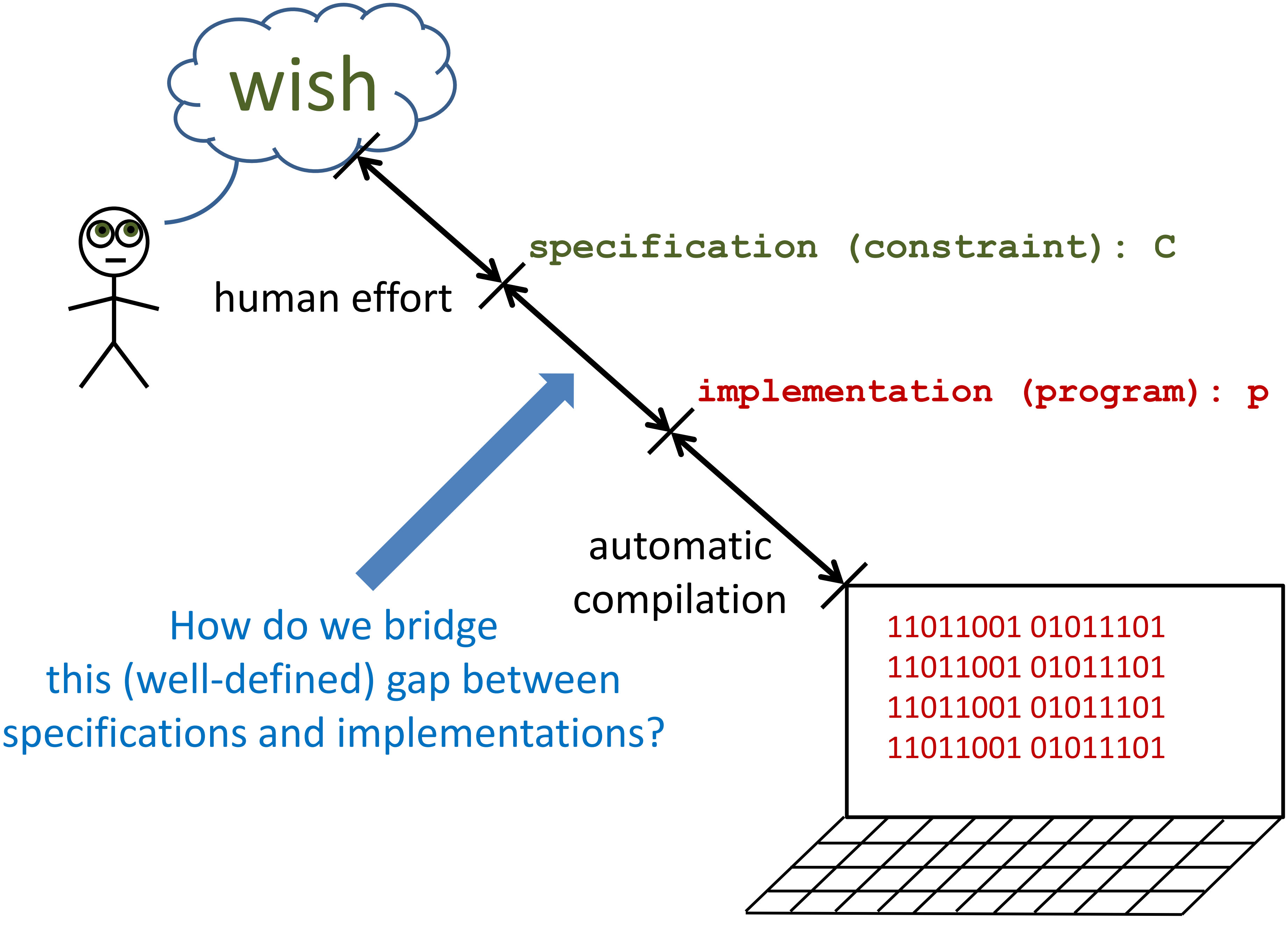
implementation

output
2900
6000
8900
24140

$p \subseteq C$

fewer behaviors

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


wish
specification (constraint): C
human effort
implementation (program): p
automatic
compilation
How do we bridge
this (well-defined) gap between
specifications and implementations?
11011001 01011101
11011001 01011101
11011001 01011101
11011001 01011101

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))**

```
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

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

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

```
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))
}
```

Type in a Scala program and watch it verified

timeout

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

input **i** such that **not** $C(i,p(i))$

Insertion Sort Verified as You Type It

Leon | Save | Undo | Redo | Link | Menu | Console | Params

```scala
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))
```

Load a Program:

-- Select a Program --

Analysis Results ✔

Function	Verif.
size	✔
contents	✔
isSorted	✔
sortedIns	✔
sort	✔

☺

Web interface: http://lara.epfl.ch/leon

Reported Counterexample in Case of a Bug

Leon | Save | Undo | Redo | Link | Menu | Console | Params

```
def sortedIns(e: Int, l: List): List = {
  require(isSorted(l))
  l match {
    case Nil() => Cons(e,Nil())
    case Cons(x,xs) => Cons(x,sortedIns(e, xs))
  }
} ensuring(res => contents(res) == contents(l) ++ Set(e)
                  && isSorted(res)
                  && size(res) == s
        )
/* Insertion sort yields a sorted l
    same size and content as the in
def sort(l: List): List = (l match
  case Nil() => Nil()
  case Cons(x,xs) => sortedIns(x, s
}) ensuring(res => contents(res) ==
                   && isSorted(res)
                   && size(res) ==
}
```

Load a Program:

-- Select a Program --

Analysis Results

Verification

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

Invalid!

Function	Kind	Result	Time
sortedIns	precond.	✔ valid	0.012
sortedIns	postcond.	✖ invalid	0.051

The following valuation violates the VC:

```
l  :=  Cons(8946, Nil())
e  :=  8945
```

Verification of Functional and Imperative Scala Code

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

[Regis Blanc]()

[Etienne Kneuss]()

[Philippe Suter]()

Automated Verification: How

1) Induction: assume and prove specification:

```
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 ≥ 0 **then** $1 + \text{res1} \geq 0$

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

```
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}
```

eliminated

remained

```
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}           ...
```

Theorem provers for recursive functions?

Reasoning about abstraction functions

Adding all recursive functions f : Tree → Tree

– undecidable ☹ Turing-complete formalism

Consider abstraction functions: **m** : Tree → N

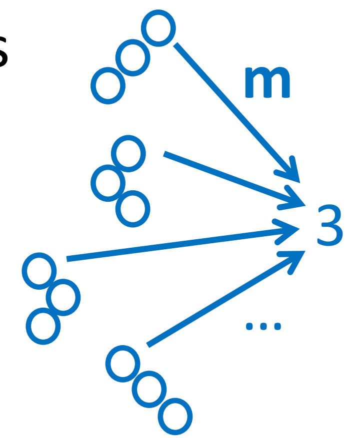
– **m** defined by simple structural recursion on trees

m == fold(leaf_const, combination_function)

size == fold(0, _ + _ + _)

content == fold({}, _ U { _ } U _)

– **sufficiently surjective**, implies card($\mathbf{m}^{-1}$(n)) → ∞



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

List(1,3,2,3,2) → List(1,1,1,2,3)

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

{| 1, 1, 2, 3 |} U {| 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?

Eva Darulova

Gap between **floating points** and **reality**

- input measurement error
- floating-point round-off error
- numerical method error
- all other sources of bugs

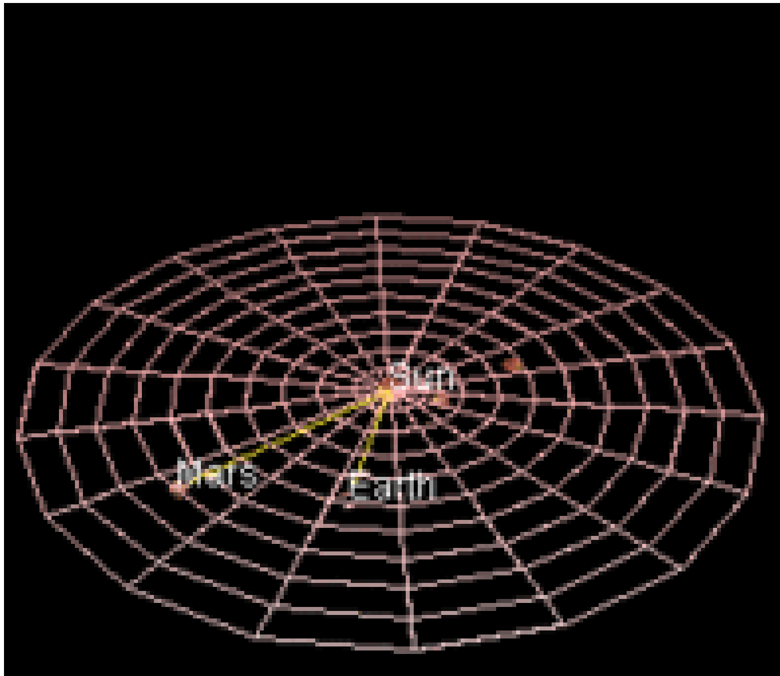
x<y need not mean x*<y*

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 26.779661250742335
```

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 26.779661250742 35
QuadDouble:    -2h 36m 26.7796612340577158626981678...

AffineFloat:   -2h 36m 26.779661250681812 (3.9991e-07)
```

provably correct digits

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)
- Community standards for representation of programs and properties EU COST Action IC0901, http://RichModels.epfl.ch

ATVA'12, CAV'13

Hossein Hojjat, PhD 2013

w/ Radu Iosif, Filip Konečny, Philipp Ruemmer




Distributed Software – Hardest of All

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

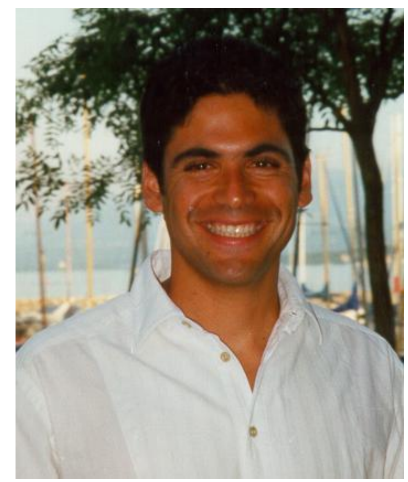
Maysam Yabandeh
Qatar CRI

Dejan Kostić
IMDEA Networks

NSDI'09, TOCS'10

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

Giuliano Losa



Rachid Guerraoui

Speculative Linearizability, PLDI 2012

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

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

Programming without Programs

c) Constraint programming: find a value that satisfies a given constraint: **find o such that C(i,o)**

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

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

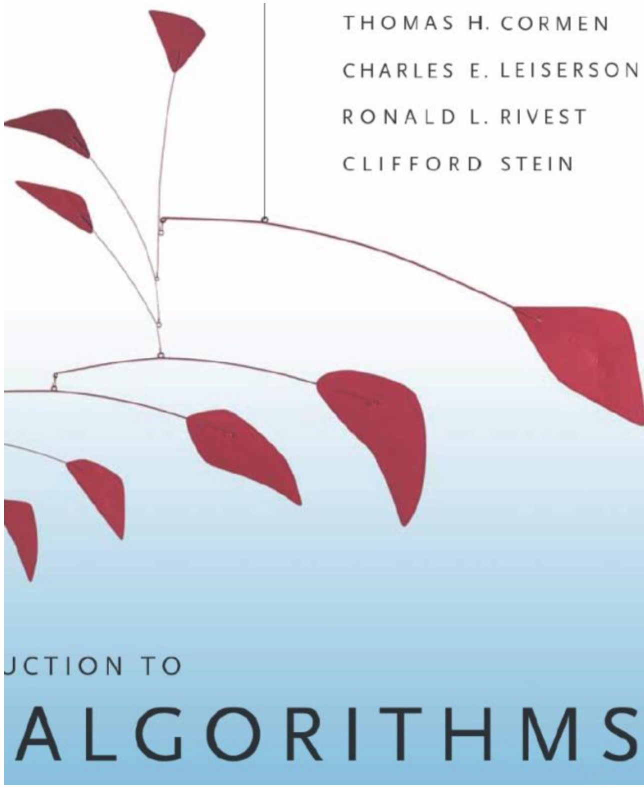
Philippe Suter

Ali Sinan Köksal

Constraints as Control, POPL 2012
Extension of Scala with constraint programming

13 Red-Black Trees

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.



Implementation: next 30 pages

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.

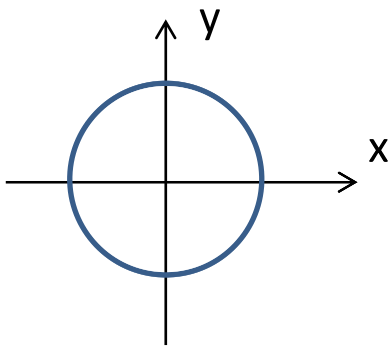
invariants - specification

Sorting a List Using Specifications

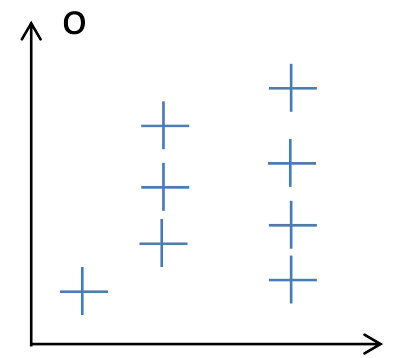
```
def content(lst : List) = lst match {
  case Nil() ⇒ Set.empty
  case Cons(x, xs) ⇒ Set(x) ++ content(xs)
}
def isSorted(lst : List) = lst match {
  case Nil()                 ⇒ true
  case Cons(_, Nil())        ⇒ true
  case Cons(x, Cons(y, ys)) ⇒ x < y && isSorted(Cons(y,ys))
}
((l : List) ⇒ isSorted(lst) && content(lst) == Set(0, 1, -3))
.solve

> Cons(-3, Cons(0, Cons(1, Nil())))
```

Implicit Programming (ERC project)

specification
(constraint)
implicit



$x^2 + y^2 = 1$

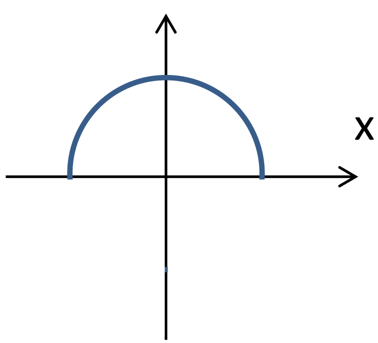


i is a propositional formula and
o is an assignment making i true - P

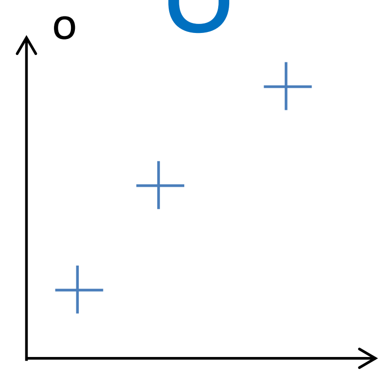
∪

∪

implementation
(function)
explicit



$y = \mathrm{sqrt}(1-x^2)$



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

assert(i % 2 == 1)

3 i + 2 o = 13 ➡ o = (13 – 3 i)/2

- Wanted: "Gaussian elimination" for programs
  - for linear integer equations: extended Euclid's algorithm
  - need to handle disjunctions, negations, more data types
- For every formula in Presburger arithmetic
  - synthesis algorithm terminates
  - produces the most general precondition (assertion characterizing when the result exists)
  - generated code always terminates and gives correct result
- If there are multiple or no solutions for some input parameters, the algorithm identifies those inputs
- Works not only for arithmetic but also for e.g. sets with sizes and for trees
- Goal: lift everything done for SMT solvers to synthesizers

Synthesis for Linear Arithmetic

```
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)

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 &&
balanced
we can conjoin specs
a **union** b == s && a **intersect** b == empty
))
partition

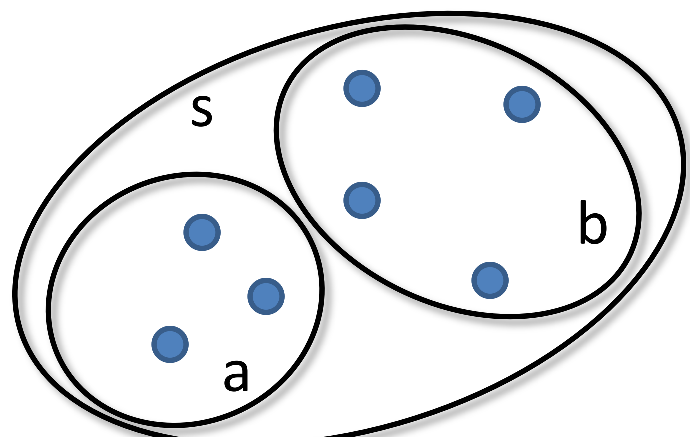
def splitBalanced[T](s: **Set[T]**) : (**Set[T]**, **Set[T]**) =

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)


Mikael Mayer


Ruzica Piskac


Philippe Suter



Automata-Based Synthesis for Arithmetic

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

Jad Hamza
ENS Cachan

Barbara Jobstmann
EPFL, Jasper DA

Andrej Spielmann

- 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

```
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();
  }
```

```
new SequenceInputStream(new FileInputStream(sig), new FileInputStream(sig))
new SequenceInputStream(new FileInputStream(sig), new FileInputStream(body))
new SequenceInputStream(new FileInputStream(body), new FileInputStream(sig))
new SequenceInputStream(new FileInputStream(body), new FileInputStream(body))
new SequenceInputStream(new FileInputStream(sig), System.in)

Press 'Ctrl+Space' to show Default Prop
```

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

Iulian Dragoș

PLDI 2013

Tihomir Gvero Ivan Kuraj Ruzica Piskac

Collaboration with LAMP

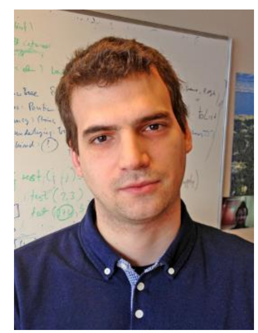
Scala

[Martin Odersky]

[Miguel Garcia]

[Lukas Rytz]

[Hubert Plociniczak]



[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) }
```

```
def insertSorted(lst: List, v: Int): List = {
  require(isSorted(lst))
  lst match {
    case Nil ⇒ Cons(v, Nil)
    case Cons(h, tail) ⇒
      val r = insertSorted(t, v)
      if (v > h) Cons(h, r)
      else if (h == v) r
      else Cons(v, Cons(h, t)) }

def sort(lst : List): List = lst match {
  case Nil ⇒ Nil
  case Cons(h, t) ⇒ insertSorted(sort(t), h) }
```

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

OOPSLA 2013:
Synthesis Modulo Recursive Functions


Etienne Kneuss


Ivan Kuraj


Philippe Suter

Results for Synthesis in Leon

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

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

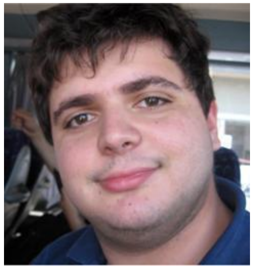
From In-Memory to External Sorting

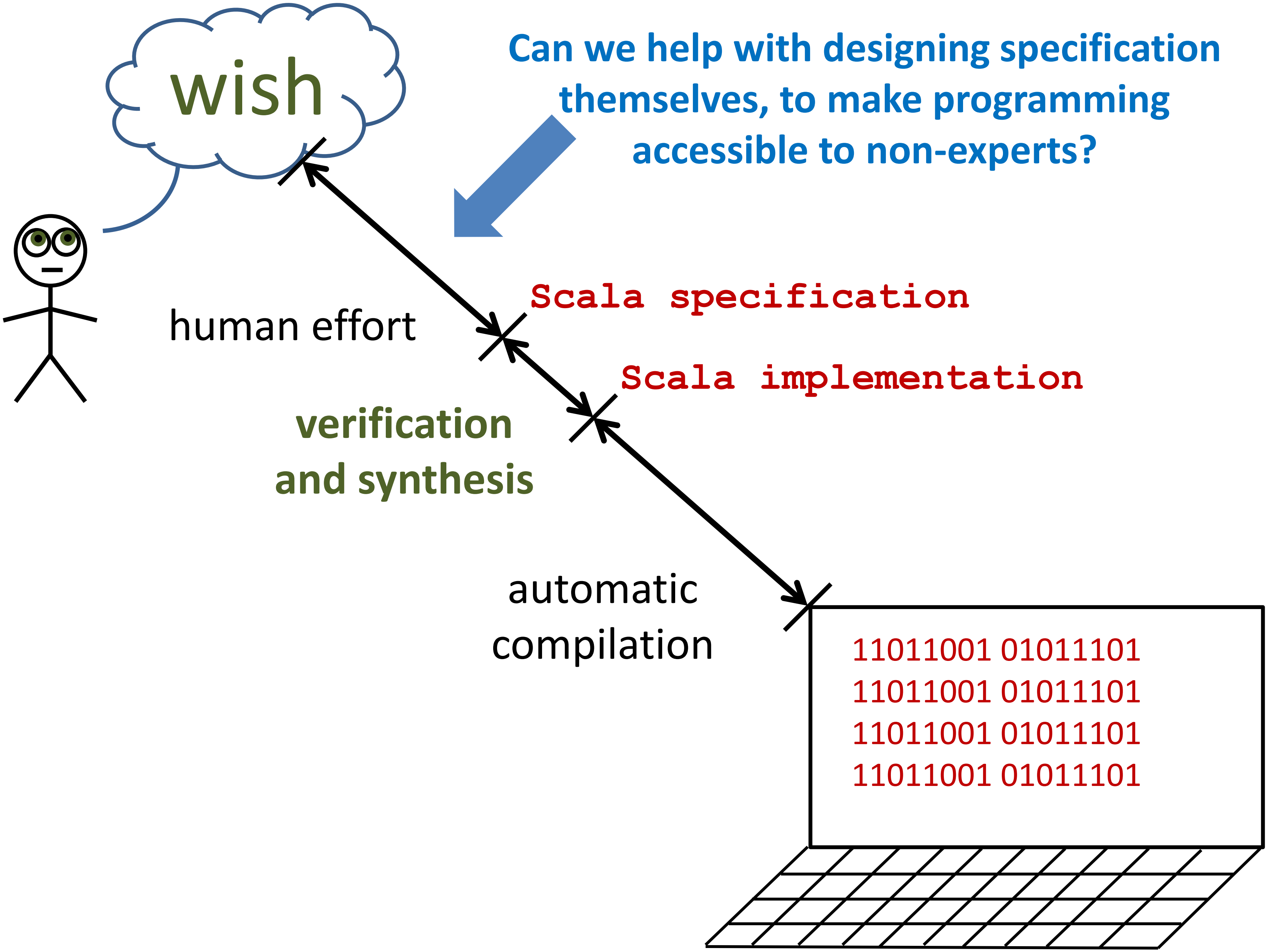
Transform **functional specification** of data base operations into **algorithms** that work when not all data fits into memory (sort -> external sort)   SIGMOD'13

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




wish
Can we help with designing specification themselves, to make programming accessible to non-experts?
human effort
Scala specification
verification
and synthesis
Scala implementation
automatic
compilation
11011001 01011101
11011001 01011101
11011001 01011101
11011001 01011101

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

Wiktionary
['wɪkʃənrɪ] *n.*,
a wiki-based Open
Content dictionary

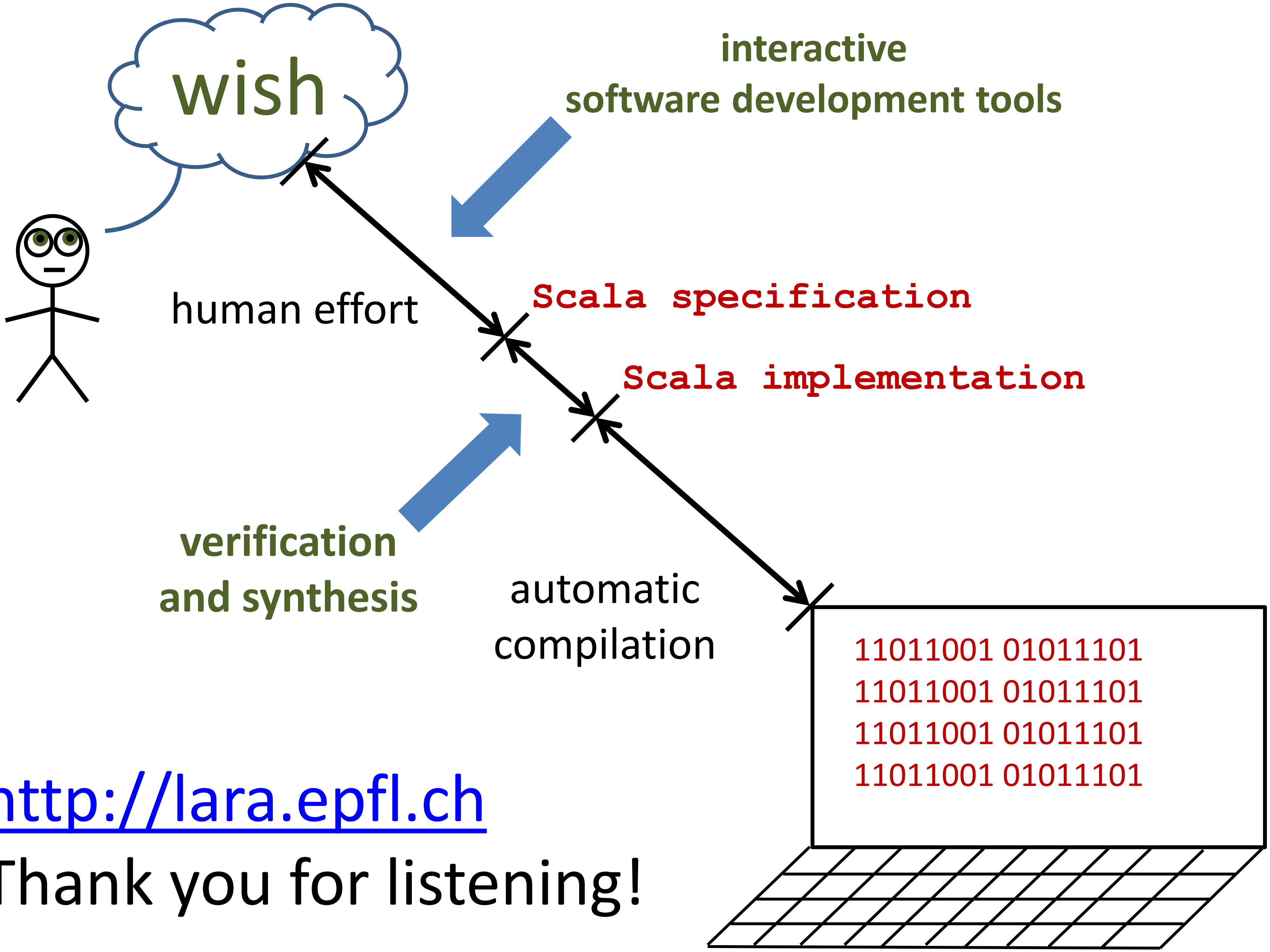
Entry | Discussion | Citations

your wish is my command

or some such gibberish. To go a step further he would like to write $\Sigma a_{ij} \cdot b_{jk}$ instead of the fairly involved set of instructions corresponding to this expression. In fact a programmer might not be considered too

1954

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.



http://lara.epfl.ch

Thank you for listening!