Synthesis using Variable Elimination

Viktor Kuncak

EPF Lausanne

http://lara.epfl.ch/w/impro
Implicit Programming at All Levels

Opportunities for implicit programming in

- **Development** within an IDE
  - InSynth tool

- **Compilation**
  - Comfusy and RegSy tools

- **Execution**
  - Scala^Z3, UDITA, Kaplan
\[ \forall a. \ pre(a) \rightarrow P(a,f(a)) \]

constraint (relation): \( P(a,x) \)

between inputs and outputs

function: \( x = f(a) \)

from inputs to outputs

precondition: \( pre(a) \)

\[ \exists x. P(a_1, x) \]
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 ))

def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
val t1 =
val t2 =
val t3 =
val t4 =
(t1, t3, t4)
Todays Approach:

```scala
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 )

def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  val t1 =
  val t2 =
  val t3 =
  val t4 =
  (t1, t3, t4)
```

slides with holes
Solve Equations

Use one equation to reduce the number of variables by one:

\[
\text{choose}((h: \text{Int}, m: \text{Int}, s: \text{Int}) \Rightarrow ( \\
\quad h \times 3600 + m \times 60 + s == \text{totalSeconds} \\
\quad && h \geq 0 \\
\quad && m \geq 0 \land m < 60 \\
\quad && s \geq 0 \land s < 60 ))
\]

\[s = \text{totalSeconds} - 3600h - 60m\]

\[0 \leq ts - 3600h - 60m\]

\[ts - 3600h - 60m \leq 59\]
One equation where no coefficient is 1

\[
\text{choose}((x,y) \Rightarrow 10 \, x + 14 \, y == a)
\]

Find one possible pair \(x,y\), as a function of \(a\).  

21a
Constructive (Extended) Euclid’s Algorithm: \( \text{gcd}(10, 14) \)

\[
\begin{align*}
3 \cdot 10 & -2 \cdot 14 = 2 \\
1 \cdot 10 & -2 \cdot 4 = 2 \\
1 \cdot 6 & + -1 \cdot 4 = 2 \\
1 \cdot 2 & + 0 \cdot 4 = 2 \\
1 \cdot 2 & + 0 \cdot 2 = 2 \\
1 \cdot 2 & + 0 \cdot 0 = 2 \\
3b & -2b \\
10x + 14y = 2 / b \\
10x + 14y = a
\end{align*}
\]

precondition: 
\[2 \mid a \]
\[b = \frac{a}{2} \]

\[
\begin{align*}
ax + (3-a)d &= 2 \\
ax + (c-d) + 3 \cdot d &= 2 \\
ax + c - dx + 3 \cdot d &= 2
\end{align*}
\]
Result of Synthesis for one Equation

\[
\text{choose}\{(x,y) \Rightarrow 10\, x + 14\, y == a\}
\]

\[
\text{assert}\ (a \mod 2 == 0);\quad \text{val}\ b = a/2
\]
\[
\text{val}\ x = 3 \times b
\]
\[
\text{val}\ y = -2 \times b
\]

check the solution: 
\[
10 \cdot 3 \frac{a}{2} + 14 \cdot (-2 \cdot \frac{a}{2}) = 15a - 14a
\]

Change in requirements! Customer also wants: \(x < y\)
does our solution work for e.g.  \(a=2\)
Idea: Eliminate Variables

choose((x,y) => 10 x + 14 y == a && x < y)

Cannot easily eliminate one of the x,y.

Eliminate both x,y, get one free - z

x = \frac{3}{2} a + c_1 z
y = -a + c_2 z

Find general form, characterize all solutions, parametric solution
Idea: Eliminate Variables

choose((x,y) => 10 x + 14 y == a && x < y)

\[
\begin{align*}
\text{assert}(a \mod 2 == 0) \\
\text{val } b &= a/2 \\
\text{val } z &= \text{divUp}(5 \times b + 1, 12) \\
\text{val } x &= 3 \times b - 7 \times z \\
\text{val } y &= -a + 5 \times z \\
\end{align*}
\]

\[
\begin{align*}
x &= 3/2 \times a + c_1 \times z \\
y &= -a + c_2 \times z
\end{align*}
\]

\[
\begin{align*}
10 x_0 + 14 y_0 + (10 c_1 + 14 c_2) &= a \\
\frac{1}{10} c_1 + \frac{7}{10} c_2 &= 0 \\
10 c_1 + 14 c_2 &= 0
\end{align*}
\]

\[
\begin{align*}
c_1 &= -7 \\
c_2 &= 5
\end{align*}
\]
\text{choose}\((x,y) \Rightarrow 10\ x + 14\ y = a \land x < y)\)

\[
x = \frac{3}{2} a + 5\ z
\]
\[
y = -a - 7\ z
\]
\[
z < \left\lfloor \frac{-\frac{5}{2} a - 1}{12} \right\rfloor
\]
I forgot to say, we also wish: \( 0 < x \)

\[
\text{choose}((x,y) \Rightarrow 10 \times + 14 \times y = a \\
&& 0 < x && x < y)
\]
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 ))

def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
val t1 =
val t2 =
val t3 =
val t4 = (t1, t3, t4)
Synthesis Procedure on Example

- process every equality: take an equality $E_i$, compute a parametric description of the solution set and insert those values in the rest of formula

\[
\begin{pmatrix}
  h \\
  m \\
  s
\end{pmatrix} = \lambda \begin{pmatrix}
  1 \\
  0 \\
  -3600
\end{pmatrix} + \mu \begin{pmatrix}
  0 \\
  1 \\
  -60
\end{pmatrix} + \begin{pmatrix}
  0 \\
  0 \\
  totalSeconds
\end{pmatrix} \mid \lambda, \mu \in \mathbb{Z}
\]

CODE:

```scala
val h = lambda
val m = mu
val s = (-3600) * lambda + (-60) * mu + totalSeconds
```
Synthesis Procedure by Example

• process every equality: take an equality $E_i$, compute a parametric description of the solution set and insert those values in the rest of formula

$$
\begin{pmatrix}
  h \\
  m \\
  s
\end{pmatrix} = \lambda \begin{pmatrix}
  1 \\
  0 \\
  -3600
\end{pmatrix} + \mu \begin{pmatrix}
  0 \\
  1 \\
  -60
\end{pmatrix} + \begin{pmatrix}
  0 \\
  0 \\
  totalSeconds
\end{pmatrix} \mid \lambda, \mu \in \mathbb{Z}
$$

Formula (remain specification):

\[0 \leq \lambda, 0 \leq \mu, \mu \leq 59, 0 \leq totalSeconds - 3600\lambda - 60\mu, totalSeconds - 3600\lambda - 60\mu \leq 59\]
Processing Inequalities

process output variables one by one

\[ 0 \leq \lambda, \ 0 \leq \mu, \ \mu \leq 59, \ 0 \leq \text{totalSeconds} - 3600\lambda - 60\mu, \]
\[ \text{totalSeconds} - 3600\lambda - 60\mu \leq 59 \]

expressing constraints as bounds on \( \mu \)

\[ 0 \leq \lambda, \ 0 \leq \mu, \ \mu \leq 59, \ \mu \leq \left\lfloor \frac{\text{totalSeconds} - 3600\lambda}{60} \right\rfloor, \]
\[ \left\lceil \frac{(\text{totalSeconds} - 3600\lambda - 59)}{60} \right\rceil \leq \mu \]

Code:

```plaintext
val mu = min(59, (totalSeconds - 3600* lambda) div 60)
```
Fourier-Motzkin-Style Elimination

\[ 0 \leq \lambda, 0 \leq \mu, \mu \leq \left\lfloor \frac{(\text{totalSeconds} - 3600\lambda)}{60} \right\rfloor, \left\lceil \frac{\text{totalSeconds} - 3600\lambda - 59}{60} \right\rceil \leq \mu \]

Combine each lower and upper bound.

\[ 0 \leq \lambda, 0 \leq 59, 0 \leq \left\lfloor \frac{(\text{totalSeconds} - 3600\lambda)}{60} \right\rfloor, \left\lceil \frac{\text{totalSeconds} - 3600\lambda - 59}{60} \right\rceil \leq \left\lfloor \frac{\text{totalSeconds} - 3600\lambda}{60} \right\rfloor, \left\lceil \frac{\text{totalSeconds} - 3600\lambda - 59}{60} \right\rceil \leq 59 \]

Basic simplifications.

\[ 0 \leq \lambda, 60\lambda \leq \left\lfloor \frac{\text{totalSeconds}}{60} \right\rfloor, \left\lceil \frac{\text{totalSeconds} - 59}{60} \right\rceil - 59 \leq 60\lambda \]

Code:

```plaintext
val lambda = totalSeconds div 3600
Preconditions: 0 \leq \text{totalSeconds}
```
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))

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)
Handling of Inequalities

• Solve for one by one variable:
  – separate inequalities depending on polarity of $x$:
    • $A_i \leq \alpha_i x$
    • $\beta_j x \leq B_j$
  – define values $a = \max_i [A_i/\alpha_i]$ and $b = \min_j [B_j/\beta_j]$
• if $b$ is defined, return $x = b$ else return $x = a$
• further continue with the conjunction of all formulas $[A_i/\alpha_i] \leq [B_j/\beta_j]$
\[(x, y) \Rightarrow c. \]

\[
2y - b \leq 3x + a \land 2x - a \leq 4y + b
\]

\[
2y - b - a \leq 3x \land 2x \leq 4y + a + b
\]

\[
4y - 2b - 2a \leq 6x \leq 12y + 3a + 3b
\]

\[
(4y - 2b - 2a) / 6 \leq x \leq (12y + 3a + 3b) / 6
\]

\[
(4y - 2b - 2a) / 6 \leq [(12y + 3a + 3b) / 6]
\]

two extra variables:

\[
(4y - 2b - 2a) / 6 \leq l \land 12y + 3a + 3b = 6 \cdot l + k
\]

\[
\land 0 \leq k \leq 5
\]

\[
4y - 2b - 2a \leq 6 \cdot l \land 12y + 3a + 3b = 6 \cdot l + k
\]

\[
\land 0 \leq k \leq 5
\]

pre: \(6 \mid 3a + 3b - k + 12y\)

\[
4y - 2b - 2a \leq 12y + 3a + 3b - k
\]

\[
-5b - 5a + k \leq 8y
\]

\[
\Rightarrow y = [(k - 5a - 5b)/8]
\]
val (x1, y1) = choose(x: Int, y: Int =>
    2*y − b <= 3*x + a && 2*x − a <= 4*y + b)

val kFound = false
for k = 0 to 5 do {
    val v1 = 3 * a + 3 * b − k
    if (v1 mod 6 == 0) {
        val alpha = ((k − 5 * a − 5 * b)/8).ceiling
        val l = (v1 / 6) + 2 * alpha
        val y = alpha
        val kFound = true
        break
    }
}
if (kFound)
    val x = ((4 * y + a + b)/2).floor
else throw new Exception("No solution exists")
NP-Hard Constructs

• Divisibility combined with inequalities:
  – corresponding to big disjunction in q.e., we will generate a for loop with constant bounds (could be expanded if we wish)

• Disjunctions
  – Synthesis of a formula computes program and exact precondition of when output exists
  – Given disjunctive normal form, use preconditions to generate if-then-else expressions (try one by one)
Synthesis for Disjunctions

\[
[x, D_1 \lor \ldots \lor D_n] = \\
\text{let } (\text{pre}_1, \vec{\Psi}_1) = [x, D_1] \\
\ldots \\
(\text{pre}_n, \vec{\Psi}_n) = [x, D_n] \\
in \\
\left( \bigvee_{i=1}^{n} \text{pre}_i, \begin{cases} 
\text{if } (\text{pre}_1) \vec{\Psi}_1 \\
\text{else if } (\text{pre}_2) \vec{\Psi}_2 \\
\ldots \\
\text{else if } (\text{pre}_n) \vec{\Psi}_n \\
\text{else assert(false)}
\end{cases} \right)
\]
choose x such that $F(x,a) \implies x = t(a)$

Result $t(a)$ is expressed in terms of $+, -, C^*, /C, \text{if}$

Need arithmetic for solving equations

Need conditionals for

- disjunctions in input formula
- divisibility and inequalities (find a witness meeting bounds and divisibility by constants)

$t(a) = \text{if } P_1(a) \ t_1(a) \ \text{elseif} \ldots \ \text{elseif} \ P_n(a) \ t_n(a) \ \text{else}\ \text{error(“No solution exists for input”,a)}$
Methodology QE → Synthesis

• Each quantifier elimination ‘trick’ we found corresponds to a synthesis trick
• Find the corresponding terms
• Key techniques:
  – one point rule immediately gives a term
  – change variables, using a computable function
  – strengthen formula while preserving realizability
  – recursively eliminate variables one-by one
• Example use
  – transform formula into disjunction
  – strengthen each disjunct using equality
  – apply one-point rule
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
val (x1, y1) = choose(x: Int, y: Int =>
    2*y - b <= 3*x + a && 2*x - a <= 4*y + b)

val kFound = false
for k = 0 to 5 do {
    val v1 = 3 * a + 3 * b - k
    if (v1 mod 6 == 0) {
        val alpha = ((k - 5 * a - 5 * b)/8).ceiling
        val l = (v1 / 6) + 2 * alpha
        val y = alpha
        val kFound = true
        break
    }
}
if (kFound)
    val x = ((4 * y + a + b)/2).floor
else throw new Exception("No solution exists")
Basic Compile-Time Analysis
Compile-time warnings

```python
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
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
Handling Data Structures
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)
• Observation:
  – Reasoning about collections reduces to reasoning about linear integer arithmetic!

\[ a.\text{size} == b.\text{size} && a \cup b == \text{bigSet} && a \cap b == \text{empty} \]
From Data Structures to Numbers

• Observation:
  – Reasoning about collections reduces to reasoning about linear integer arithmetic!

\[ a.\text{size} == b.\text{size} \land a \cup b == \text{bigSet} \land a \cap b == \text{empty} \]
From Data Structures to Numbers

• Observation:
  - Reasoning about collections reduces to reasoning about linear integer arithmetic!

\[ a \text{.size} == b \text{.size} \land \land a \text{ union } b == \text{bigSet } \land \land a \text{ intersect } b == \text{empty} \]
From Data Structures to Numbers

• Observation:
  – Reasoning about collections reduces to reasoning about linear integer arithmetic!

\[ a.\text{size} == b.\text{size} \quad \&\& \quad a \cup b == \text{bigSet} \quad \&\& \quad a \cap b == \text{empty} \]

New specification:

\[ kA = kB \]
From Data Structures to Numbers

• Observation:
  – Reasoning about collections reduces to reasoning about linear integer arithmetic!

\[
a.\text{size} == b.\text{size} \&\& a \text{ union } b == \text{bigSet} \&\& a \text{ intersect } b == \text{empty}
\]

New specification:
\[
kA = kB \&\& kA + kB = |\text{bigSet}|
\]
\[(A, B, d), \ A \cup B = S \land A \cap B = \emptyset \land |A| + 5d = 2|B| \land d \neq 0\]

We start by labelling the sizes of Venn regions of the sets \(A, B\) and \(S\) by variables \(k_i\), as displayed in the following diagram:

We rewrite our synthesis problem using these new variables:

\[A \cup B = S \iff k_1 = k_2 = k_3 = k_4 = 0\]
\[A \cap B = \emptyset \iff k_3 = k_7 = 0\]
\[|A| + 5d = 2|B| \iff \sum_{i \in \{1,3,5,7\}} k_i + 5d = 2 \cdot \sum_{i \in \{2,3,6,7\}} k_i\]
\[d \neq 0 \iff d \neq 0\]
\[(a, b, d), \ a + b = s \land a + 5d = 2b \land a \geq 0 \land b \geq 0 \land d \neq 0\]

\[a + b = s \land a + 5d = 2b \land a \geq 0 \land b \geq 0 \land d \geq 1\]

\[\alpha, \ 5\alpha \leq -s \land -2s \leq 5\alpha \land 1 - s \leq 3\alpha\]
Experience with Comfusy 😊

• Works well for examples we encountered
  – Needed: synthesis for more expressive logics, to handle more examples
  – Seems ideal for domain-specific languages

• Efficient for conjunctions of equations (could be made polynomial)

• **Extends to synthesis with parametric coefficients**

• Extends to logics that reduce to Presburger arithmetic (implemented for BAPA)
Comfusy for Arithmetic 😞

• Limitations of Comfusy for arithmetic:
  – Naïve handling of disjunctions
  – Blowup in elimination, divisibility constraints
  – *Complexity of running synthesized code (from QE): doubly exponential in formula size*
  – Not tested on modular arithmetic, or on synthesis with optimization objectives
  – Arbitrary-precision arithmetic with multiple operations generates time-inefficient code
  – Cannot do bitwise operations (not in PA)
Arithmetic Synthesis using Automata
RegSy

Synthesis for regular specifications over unbounded domains

J. Hamza, B. Jobstmann, V. Kuncak
FMCAD 2010
Automata-Based Synthesis

• Disjunctions not a problem
• Result not depends on the syntax of input formula
• Complexity of running synthesized code: *ALWAYS LINEAR IN THE NUMBER OF BITS*
• Modular arithmetic and bitwise operators: synthesize bit tricks for
  – unbounded number of bits, uniformly
  – without skeletons
• Supports quantified constraints
  – including optimization constraints
Expressiveness of Spec Language

- Non-negative integer constants and variables
- Boolean operators ($\land, \lor, \neg$)
- Linear arithmetic operator ($+ , c \cdot x$)
- Bitwise operators ($|, &, !$)
- Quantifiers over numbers and bit positions

$\text{PAbit} = \text{Presburger arithmetic with bitwise operators}$

$\text{WS1S} = \text{weak monadic second-order logic of one successor}$
Fig. 1. Automaton $A$ for parity specification between $x$ and $y$

Fig. 2. Input $x$ and output $y$ satisfying parity specification

$\begin{array}{c}
\text{Transition} \\
\{q_0\} \rightarrow \{q_1, q_2\} \\
\{q_0\} \rightarrow \{q_1, q_2\} \\
\{q_0\} \rightarrow \{q_1, q_2\} \\
\{q_0\} \rightarrow \{q_1, q_2\} \\
\{q_1, q_2\} \rightarrow \{q_1, q_2\} \\
\{q_1, q_2\} \rightarrow \{q_1, q_2\} \\
\{q_1, q_2\} \rightarrow \{q_1, q_2\} \\
\{q_1, q_2\} \rightarrow \{q_1, q_2\}
\end{array}$

$\begin{array}{c|c|c}
\text{State} & \tau \\
q_1 & (q_0, 0) \\
q_2 & (q_0, 1) \\
q_1 & (q_0, 1) \\
q_2 & (q_0, 0) \\
q_1 & (q_1, 0) \\
q_2 & (q_2, 0) \\
q_1 & (q_2, 0) \\
q_2 & (q_1, 0)
\end{array}$

Fig. 3. Automaton $A'$ for computing parity $y$ of input $x$

Fig. 4. Running synthesized function on input shown in Fig. 2
Basic Idea

• View integers as finite (unbounded) bit-streams (binary representation starting with LSB)
• Specification in WS1S (PAbit)
• Synthesis approach:
  – Step 1: Compile specification to automaton over combined input/output alphabet (automaton specifying relation)
  – Step 2: Use automaton to generate efficient function from inputs to outputs realizing relation
Inefficient but Correct Method

• Run over a deterministic automaton over inputs and outputs
• Becomes run of a non-deterministic if we look only at outputs
• Simulate all branches until the end of input
• Successful branches indicate outputs that work
Our Approach: Precompute
without losing backward information

Synthesis:
1. Det. automaton for spec over joint alphabet
2. Project, determinize, extract lookup table

Synthesized program:
Automaton + lookup table

Execution:
1. Run A on input w and record trace
2. Use table to run backwards and output

Input word

WS1S spec

Mona

Output word
### Experiments

| No | Example     | MONA (ms) | Syn (ms) | |A| | |A'| | 512b | 1024b | 2048b | 4096b |
|----|-------------|-----------|----------|-----|-----|-----|-----|------|------|------|------|------|
| 1  | addition    | 318       | 132      | 4   | 9   | 509 | 995 | 1967 | 3978 |
| 2  | approx      | 719       | 670      | 27  | 35  | 470 | 932 | 1821 | 3641 |
| 3  | company     | 8291      | 1306     | 58  | 177 | 608 | 1312| 2391 | 4930 |
| 4  | parity      | 346       | 108      | 4   | 5   | 336 | 670 | 1310 | 2572 |
| 5  | mod-6       | 341       | 242      | 23  | 27  | 460 | 917 | 1765 | 3567 |
| 6  | 3-weights-min | 26963     | 640      | 22  | 13  | 438 | 875 | 1688 | 3391 |
| 7  | 4-weights   | 2707      | 1537     | 55  | 19  | 458 | 903 | 1781 | 3605 |
| 8  | smooth-4b   | 51578     | 1950     | 1781| 955 | 637 | 1271| 2505 | 4942 |
| 9  | smooth-f-2b | 569       | 331      | 73  | 67  | 531 | 989 | 1990 | 3905 |
| 10 | smooth-b-2b | 569       | 1241     | 73  | 342 | 169 | 347 | 628  | 1304 |
| 11 | 6-3n+1      | 834       | 1007     | 233 | 79  | 556 | 953 | 1882 | 4022 |

In 3 seconds solve constraint, minimizing the output; Inputs and outputs are of order \(2^{4000}\)
Thought: A quantifier not to forget

• Automatically synthesized and only automatically tested code is great
  – But probably even less likely to be correct

Synthesizing programs that are guaranteed to be correct by construction will remain important.
Implicit Programming at All Levels

Opportunities for implicit programming in

• **Development** within an IDE
  – **InSynth** tool

✓ • **Compilation**
  – **Comfusy** and **RegSy** tools

✓ • **Execution**
  – **Scala^Z3** and **UDITA** tools
import java.io.FileInputStream
import java.io.IOException
import java.io.SequenceInputStream

... 
def main() = {
  var body = "email.txt"
  var sig = "signature.txt"
  val all : SequenceInputStream = ■
}

If we invoke tool at the program point indicated by ■, in a fraction of second it displays the following ranked list of five expressions:

1. `new SequenceInputStream(new FileInputStream(sig),
                           new FileInputStream(sig))`
2. `new SequenceInputStream(new FileInputStream(body),
                           new FileInputStream(sig))`
3. `new SequenceInputStream(new FileInputStream(sig),
                           new FileInputStream(body))`
4. `new SequenceInputStream(new FileInputStream(body),
                           new FileInputStream(body))`
5. `new SequenceInputStream(System.in,
                           new FileInputStream(sig))`
Code Synthesis inside IDE

- Find all visible symbols
- Create clauses:
  - encode in FOL
  - assign weights

Resolution algorithm with weights

Code snippets
Curry-Howard Correspondence

\[ \forall A, B. \text{List}(A) \to (A \to B) \to \text{List}(B) \]

Example inference:

\[ f : (P(x) \to Q(x)) \quad g : (Q(t(y)) \to R(y)) \]

\[ (\forall z, g(f(z))) : P(t(y)) \to R(y) \]

Code fragment

newly derived type
joint work with: Tihomir Gvero, Ruzica Piskac

**InSynth** - Interactive Synthesis of Code Snippets

```scala
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 (prn, intList))

Is there a term of given type in given environment?  
Monomorphic: decidable. Polymorphic: undecidable
Features

• Parametric polymorphism
• Branching expressions with multiple arguments (not just chains of calls)
• Mining API usage frequencies
  – but can still synthesize previously unseen code
• Distance to program point heuristics
• Filtering based on test-cases
Wrapping Up
Synthesis and Verification

- **Some** functionality is best *synthesized* from specs
  - **other**: better *implemented, verified* (and put into library)
- Currently, no choice – must always implement
  - specifications and verification are viewed as **overhead**
- Goal: make specifications intrinsic part of programs, with clear benefits to programmers – **they run!**
- Example: write an assertion, not how to establish it
- This will help both
  - verifiability: document, not reverse-engineer the invariants
  - productivity: avoid writing messy implementation
Conclusion: Implicit Programming

Development within an IDE
- **InSynth** tool – theorem proving as code completion

Compilation
- **Comfusy**: decision procedure → synthesis procedure
  Scala implementation for integer arithmetic, BAPA
- **RegSy**: solving WS1S constraints

Execution
- **UDITA**: Java + choice as test generation language
- **Scala^Z3**: invoking Z3
- **Kaplan**: constraint programming in Scala

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