Type Inhabitation Problem for Code Completion and Repair

Ruzica Piskac

Yale University

Joint work with I. Kuraj, V. Kuncak, T. Gvero (EPFL)

Example: Sequence of Streams

```
def main(args:Array[String]) = {
    var body:String = "email.txt"
    var sig:String = "signature.txt"
    var inStream:SeqInStr = [redacted]
    …
}
```

Example: Sequence of Streams

```
def main(args:Array[String]) = {
    var body:String = "email.txt"
    var sig:String = "signature.txt"
    var inStream:SeqInStr =
    ...
}
```

```
new SeqInStr(new FileInStr(sig), new FileInStr(sig))
new SeqInStr(new FileInStr(sig), new FileInStr(body))
new SeqInStr(new FileInStr(body), new FileInStr(sig))
new SeqInStr(new FileInStr(body), new FileInStr(body))
new SeqInStr(new FileInStr(sig), System.in)
```

Example: Sequence of Streams

```
def main(args:Array[String]) = {
    var body:String = "email.txt"
    var sig:String = "signature.txt"
    var inStream:SeqInStr =
    ...
}
```

```
new SeqInStr(new FileInStr(sig), new FileInStr(sig))
new SeqInStr(new FileInStr(sig), new FileInStr(body))
new SeqInStr(new FileInStr(body), new FileInStr(sig))
new SeqInStr(new FileInStr(body), new FileInStr(body))
new SeqInStr(new FileInStr(sig), System.in)
```

Example: Sequence of Streams

```
def main(args:Array[String]) = {
    var body:String = "email.txt"
    var sig:String = "signature.txt"
    var inStream:SeqInStr = new SeqInStr(new FileInStr(sig), new
FileInStr(body))

    …
}
```

Example: Sequence of Streams

```
def main(args:Array[String]) = {
    var body:String = "email.txt"
    var sig:String = "signature.txt"
    var inStream:SeqInStr = new SeqInStr(new FileInStr(sig), new
FileInStr(body))

    …
}
```

Imported over 3300 declarations

Executed in less than 250ms

InSynth - Interactive Synthesis of Code Snippets

- Usually: software synthesis = automatically deriving code from specifications
- InSynth – a tool for synthesis of code fragments (snippets)
  - interactive
    - getting results in a short amount of time
    - multiple solutions – a user needs to select
  - component based
    - assemble program from given components (local values, API)
  - partial specification
    - hard constraints – type constraints
    - soft constraints - use of components “most likely” to be useful

Example: TreeFilter (HOF)

```
def filter(p: Tree => Boolean): List[Tree] = {
  val ft:FilterTreeTraverser = [redacted]
  ft.traverse(tree)
  ft.hits.toList
}
```

Example: TreeFilter (HOF)

```
def filter(p: Tree => Boolean): List[Tree] = {
  val ft:FilterTreeTraverser =
  ft.traverse(tree)
  ft.hits.toList
}
```

```
new FilterTreeTraverser(x => p(x))
new FilterTreeTraverser(x => isType)
new FilterTreeTraverser(x => p(tree))
new FilterTreeTraverser(x => new Wrapper(x).isType)
new FilterTreeTraverser(x => p(new Wrapper(x).tree))
```

Example: TreeFilter (HOF)

```
def filter(p: Tree => Boolean): List[Tree] = {
    val ft:FilterTreeTraverser =
    ft.traverse(tree)
    ft.hits.toList
}
```

new FilterTreeTraverser(x => p(x))
new FilterTreeTraverser(x => isType)
new FilterTreeTraverser(x => p(tree))
new FilterTreeTraverser(x => new Wrapper(x).isType)
new FilterTreeTraverser(x => p(new Wrapper(x).tree))

Example: TreeFilter (HOF)

```
def filter(p: Tree => Boolean): List[Tree] = {
  val ft:FilterTreeTraverser = new FilterTreeTraverser(x => p(x))
  ft.traverse(tree)
  ft.hits.toList
}
```

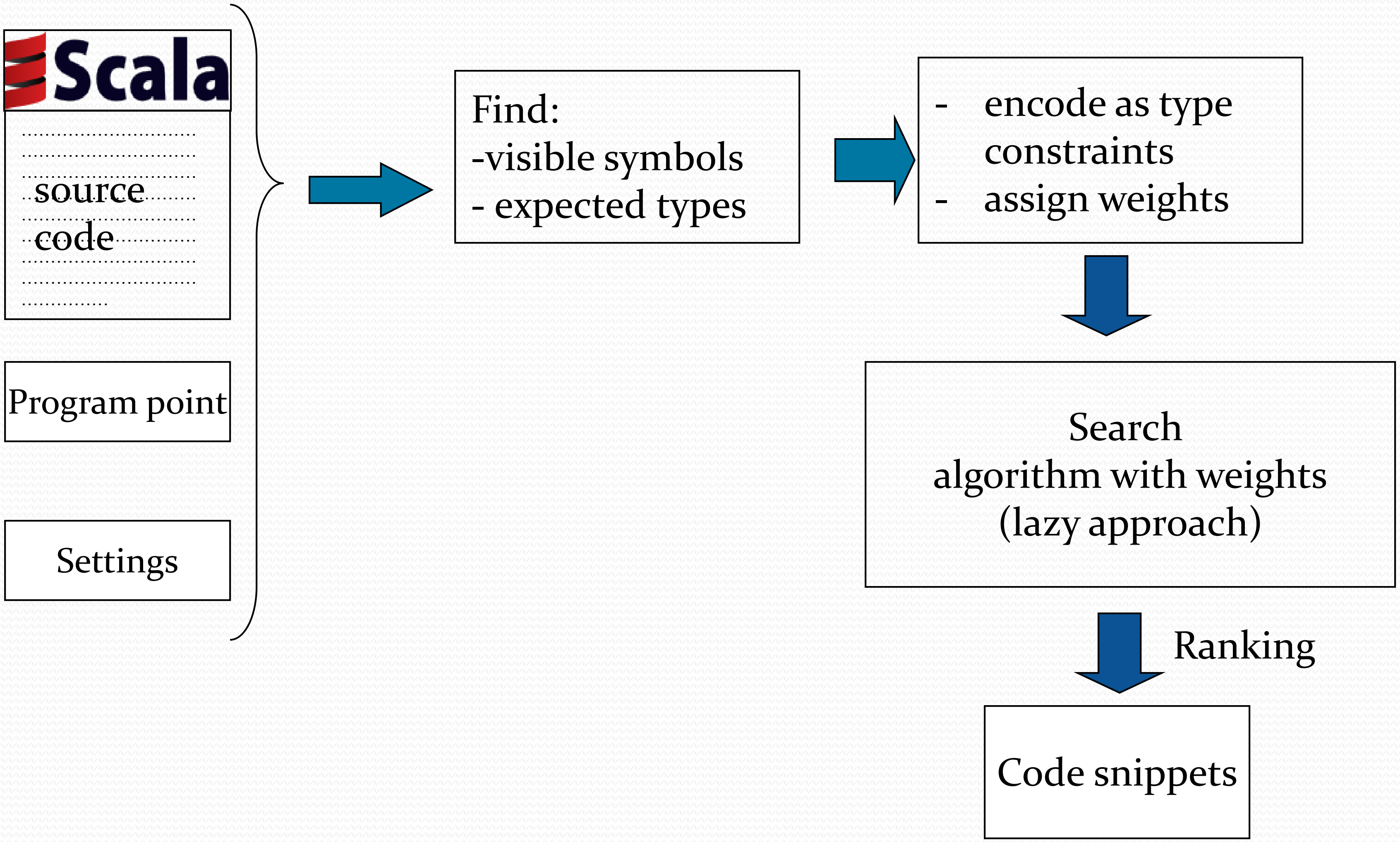
Example: TreeFilter (HOF)

```
def filter(p: Tree => Boolean): List[Tree] = {
   val ft:FilterTreeTraverser = new FilterTreeTraverser(x => p(x))
   ft.traverse(tree)
   ft.hits.toList
}
```

Imported over 4000 declarations

Executed in less than 300ms

Snippet Synthesis inside IDE



Type Inhabitation Problem

- Given a set of types *T* and a set of expressions *E*, a type environment is a set

$$\Gamma = \{e_1 : \tau_1, e_2 : \tau_2, \dots , e_n : \tau_n\}$$

Type Inhabitation Problem

Given a type environment Γ, a type τ and some calculus, is there are an expression *e* such that $\Gamma \vdash e : \tau$

Completion = Inhabitation

def m_1: T_1
...
def m_n: T_n

val a: T = ?

Completion = Inhabitation

def m_1: T_1
...
def m_n: T_n

$\Gamma = \{ m_1: T_1, \ldots, m_n: T_n \}$

val a: T = ?

Completion = Inhabitation

ENVIRONMENT

def m_1: T_1

…

def m_n: T_n

$\Gamma = \{ m_1: T_1, \ldots, m_n: T_n \}$

val a: T = ?

Completion = Inhabitation

ENVIRONMENT

def m_1: T_1

...

def m_n: T_n

$\Gamma = \{ m_1 : T_1, \ldots, m_n : T_n \}$

val a: T = ?

$\Gamma \vdash ? : T$

Completion = Inhabitation

ENVIRONMENT

def m_1: T_1

...

def m_n: T_n

$\Gamma = \{ m_1 : T_1, \ldots, m_n : T_n \}$

val a: T = ?

$\Gamma \vdash ? : T$

DESIRED TYPE

Type Inhabitation in Lambda Calculus

- Type Inhabitation for ground lambda calculus
  - The problem is PSPACE-complete [Statman, 1979]
  - More constructive algorithm [Urzyczyn, 1997]

- For weak type polymorphism (quantifiers only on the top level), the type inhabitation problem is undecidable

Simply Typed Lambda Calculus

$$\text{AX}\quad \frac{x : T \in \Gamma}{\Gamma \vdash x : T}$$

$$\text{ABS}\quad \frac{\Gamma, x : T_1 \vdash t : T}{\Gamma \vdash \lambda x.t : T_1 \to T}$$

$$\text{APP}\quad \frac{\Gamma \vdash e_1 : T_1 \to T \qquad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1(e_2) : T}$$

Simply Typed Lambda Calculus

$$\Gamma \vdash \; ? : T$$

Simply Typed Lambda Calculus

Backward Search

$$\Gamma \vdash \; ? : T$$

Simply Typed Lambda Calculus

$$\text{APP}\quad \frac{\Gamma \vdash\ ? : T_1 \rightarrow T \qquad\qquad \Gamma \vdash\ ? : T_1}{\Gamma \vdash\ ? : T}$$

Simply Typed Lambda Calculus

$$\text{APP}\ \frac{\Gamma \vdash ? : T_1 \rightarrow T \qquad \Gamma \vdash ? : T_1}{\Gamma \vdash ? : T}$$

Infinitely many

Simply Typed Lambda Calculus

No bound on types in derivation tree(s).

$$\text{APP}\ \frac{\Gamma \vdash\ ? : T_1 \rightarrow T \qquad \Gamma \vdash\ ? : T_1}{\Gamma \vdash\ ? : T}$$

Infinitely many

Long Normal Form

$$\text{ABS}\quad \frac{\Gamma, x_1:T_1,\ldots, x_n:T_n \vdash t: T}{\Gamma \vdash \lambda x_1:T_1,\ldots, x_n:T_n.t: T_1 \to \ldots \to T_n \to T}$$

$$\text{APP}\quad \frac{f : T_1 \to \ldots \to T_n \to T \in \Gamma \qquad \Gamma \vdash a_1: T_1 \quad \ldots \quad \Gamma \vdash a_n: T_n}{\Gamma \vdash f(a_1,\ldots,a_n):T}$$

Comparison between LNF and classic APP

OLD

$$\text{ABS}\quad \frac{\Gamma, x_1{:}T_1, \ldots, x_n{:}T_n \vdash t : T}{\Gamma \vdash \lambda x_1{:}T_1, \ldots, x_n{:}T_n . t : T_1 \to \ldots \to T_n \to T}$$

$$\text{APP}\quad \frac{f : T_1 \to \ldots \to T_n \to T \in \Gamma \qquad \Gamma \vdash a_1 : T_1 \quad \ldots \quad \Gamma \vdash a_n : T_n}{\Gamma \vdash f(a_1, \ldots, a_n) : T}$$

NEW

Comparison between LNF and classic APP

$$\text{ABS}\quad \frac{\Gamma, x_1{:}T_1, \ldots, x_n{:}T_n \vdash t : T}{\Gamma \vdash \lambda x_1{:}T_1, \ldots, x_n{:}T_n.t : T_1 \to \ldots \to T_n \to T}$$

$$\text{APP}\quad \frac{f : T_1 \to \ldots \to T_n \to T \in \Gamma \qquad \Gamma \vdash a_1 : T_1 \quad \ldots \quad \Gamma \vdash a_n : T_n}{\Gamma \vdash f(a_1, \ldots, a_n) : T}$$

DECLARATION from Γ

Long Normal Form

$$\Gamma \vdash \ ? : T$$

Long Normal Form

$$\text{APP}\;\frac{f : (T_1 \to T_2) \to T \in \Gamma \qquad\qquad \Gamma \vdash ? : T_1 \to T_2}{\Gamma \vdash f(?) : T}$$

Long Normal Form

$f : (T_1 \to T_2) \to T \in \Gamma$ $\qquad$ $\Gamma \vdash ? : T_1 \to T_2$

APP

$\Gamma \vdash f(?) : T$

Narrows the search space

Only one

Long Normal Form

$$\text{ABS}\ \frac{\Gamma, x_1{:}T_1 \vdash ? : T_2}{\Gamma \vdash \lambda x_1{:}T_1.? : T_1 \rightarrow T_2}$$

$$\text{APP}\ \frac{f : (T_1 \rightarrow T_2) \rightarrow T \in \Gamma \qquad \Gamma \vdash \lambda x_1{:}T_1.? : T_1 \rightarrow T_2}{\Gamma \vdash f(\lambda x_1{:}T_1.?) : T}$$

Long Normal Form

$$
\textsf{APP}\;\dfrac{f : (T_1 \to T_2) \to T \in \Gamma \qquad \textsf{ABS}\;\dfrac{\textsf{APP}\;\dfrac{\vdots}{\Gamma, x_1{:}T_1 \vdash e : T_2}}{\Gamma \vdash \lambda x_1{:}T_1.e : T_1 \to T_2}}{\Gamma \vdash f(\lambda x_1{:}T_1.e) : T}
$$

Long Normal Form

Finitely many types in derivation tree(s)

$$\vdots$$

$$\text{APP}\ \frac{}{\Gamma, x_1{:}T_1 \vdash e : T_2}$$

$$\text{ABS}\ \frac{\Gamma, x_1{:}T_1 \vdash e : T_2}{\Gamma \vdash \lambda x_1{:}T_1.e : T_1 \to T_2}$$

$$\text{APP}\ \frac{f : (T_1 \to T_2) \to T \in \Gamma \qquad \Gamma \vdash \lambda x_1{:}T_1.e : T_1 \to T_2}{\Gamma \vdash f(\lambda x_1{:}T_1.e) : T}$$

Algorithm

- Algorithm builds finite graph (with cycles) that
  - Represents all (infinitely many) solutions
  - Later we use it to construct expressions
  - Less than 10ms
- Algorithm Properties
  - Graph generation terminates
    - Type inhabitation is decidable
  - Complete - generates all solutions
  - PSPACE-complete
- Techniques
  - Succinct type representation
  - Backward search
  - Weights mechanism

Succinct Lambda Calculus

- Succinct representation of type declarations
  - def iTs (a: Int, b:Int, c: Int): String
  - iTs : {Int} → String
- Reason: efficiency

Without succinct types	With succinct types
74% cases: desired snippet is among top 5 returned solution	94% cases: desired snippet is among top 5 returned solution
56% cases: desired snippet is top ranked	64% cases: desired snippet is top ranked
Average total time: 401ms (prover 266ms, reconstructor 135ms)	Average total time: 145ms (prover 9ms, reconstructor 136ms)

Succinct Lambda Calculus

- Efficient encoding of “patterns” - a witness that type *t* is inhabited – finite graph representation of possibly infinite terms
- To derive the corresponding code snippets, we use a reconstruction function, combined with the weight function (to obtain the ranking of snippets)
- Succinct lambda calculus is sound and complete:

> **Theorem**
> A lambda term can be derived in the (standard) lambda calculus iff it can be “derived” in the succinct lambda calculus.

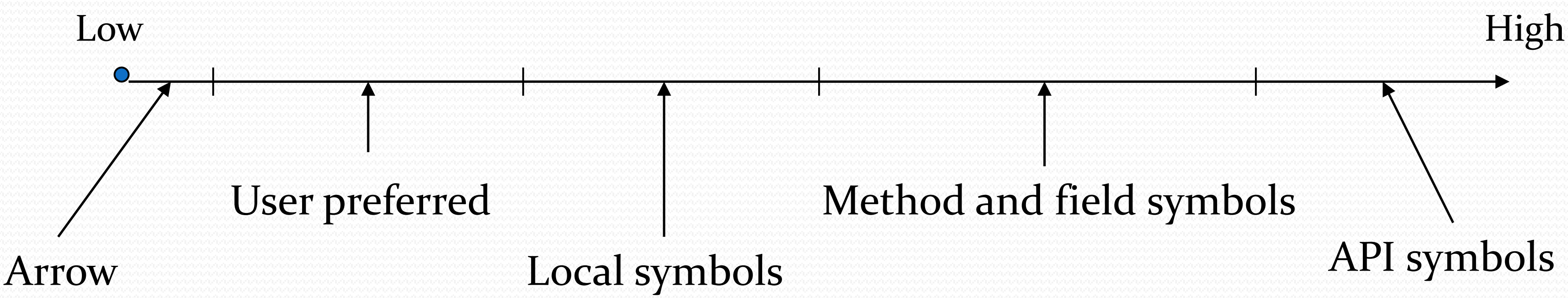
Quantitative Type Inhabitation Problem

Quantitative Type Inhabitation Problem
Given a type environment Γ, a type τ and some calculus, is there an expression *e* such that $\Gamma \vdash e : \tau$, and such that e is the “best possible solution”

- to all type assumptions we assign a weight
- lower weight indicates that term is more relevant
- weight of a term or a type is computed as the sum of the weights of all symbols

System of Weights

- Symbol weights – used for ranking solution and for directing the search
- Weight of a term is computed based on
  - precomputed term weights (obtained by analyzing a training set taken from the Web) - frequency
  - proximity to the program point where the tool is invoked



Subtyping using Coercions

- We model A <: B by introducing a coercion function
  c: A → B [Tannen et al., 1991]

```
class ArrayList[T] extends AbstractList[T] with List[T]
   with RandomAccess  with Cloneable with Serializable {…}
abstract class AbstractList[E] extends AbstractCollection[E]
   with List[E] {
   ….
   def iterator():Iterator[E] = {…}
}
```

Subtyping using Coercions

- We model A <: B by introducing a coercion function
  c: A → B [Tannen et al., 1991]

```
class ArrayList[T] extends AbstractList[T] with List[T]
   with RandomAccess  with Cloneable with Serializable {…}
abstract class AbstractList[E] extends AbstractCollection[E]
   with List[E] {
   ….
   def iterator():Iterator[E] = {…}
}
```

c1: ∀α. ArrayList[α]→ AbstractList[α]
c2: ∀β. AbstractList[β]→ AbstractCollection[β]

Subtyping Example

```
val a1: ArrayList[String] = …
…
class ArrayList[T] extends AbstractList[T] with List[T]
   with RandomAccess  with Cloneable with Serializable {…}
abstract class AbstractList[E] extends AbstractCollection[E]
   with List[E] {
   ….
   def iterator():Iterator[E] = {…}
}
…
val i1: Iterator[String] = ■
```

Subtyping Example

```
val a1: ArrayList[String] = …
…
class ArrayList[T] extends AbstractList[T] with List[T]
   with RandomAccess  with Cloneable with Serializable {…}
abstract class AbstractList[E] extends AbstractCollection[E]
   with List[E] {
   ….
   def iterator():Iterator[E] = {…}
}
…
val i1: Iterator[String] = █
```

a1: ArrayList(String)
c1: $\forall\alpha$. ArrayList[α]$\rightarrow$ AbstractList[α]
c2: $\forall\beta$. AbstractList[β]$\rightarrow$ AbstractCollection[β]
iterator: $\forall\ \gamma$. AbstractList[γ] $\rightarrow$ Iterator[γ]
??? : Iterator[String]

Subtyping Example

```
val a1: ArrayList[String] = …
…
class ArrayList[T] extends AbstractList[T] with List[T]
   with RandomAccess  with Cloneable with Serializable {…}
abstract class AbstractList[E] extends AbstractCollection[E]
   with List[E] {
   ….
   def iterator():Iterator[E] = {…}
}
…
val i1: Iterator[String] = ■
```

a1: ArrayList(String)
c1: $\forall\alpha$. ArrayList[α]$\rightarrow$ AbstractList[α]
c2: $\forall\beta$. AbstractList[β]$\rightarrow$ AbstractCollection[β]
ator: $\forall\ \gamma$. AbstractList[γ] $\rightarrow$ Iterator[γ]
: Iterator[String]

iterator(c1(a1)): Iterator[String]

Subtyping Example

```
val a1: ArrayList[String] = …
…
class ArrayList[T] extends AbstractList[T] with List[T]
   with RandomAccess  with Cloneable with Serializable {…}
abstract class AbstractList[E] extends AbstractCollection[E]
   with List[E] {
   ….
   def iterator():Iterator[E] = {…}
}
…
val i1: Iterator[String] = 
```

a1.Iterator

iterator(c1(a1)): Iterator[String]

Evaluation

Benchmarks	Size	#Initial	Rank	Total	Rank	Prove	Recon	Total
AWTPermissionStringname	2/2	5615	>10	101	1	8	125	133
BufferedInputStreamFileInputStream	3/2	3364	>10	45	1	7	46	53
BufferedOutputStream	3/2	3367	>10	18	1	7	11	19
BufferedReaderFileReaderfileReader	4/2	3364	>10	69	1	7	43	50
BufferedReaderInputStreamReader	4/2	3364	>10	66	1	7	42	49
BufferedReaderReaderin	5/4	4094	>10	4760	6	7	237	244
ByteArrayInputStreambytebuf	4/4	3366	>10	94	>10	4	18	22
ByteArrayOutputStreamintsize	2/2	3363	>10	51	2	8	63	70
DatagramSocket	1/1	3246	>10	74	1	7	80	88
DataInputStreamFileInput	3/2	3364	>10	20	1	6	46	52
DataOutputStreamFileOutput	3/2	3364	>10	29	1	7	38	45
DefaultBoundedRangeModel	1/1	6673	>10	220	1	10	257	266
DisplayModeintwidthintheightintbit	2/2	4999	>10	136	1	6	147	154
FileInputStreamFileDescriptorfdObj	2/2	3366	>10	24	2	6	17	23
FileInputStreamStringname	2/2	3363	>10	125	1	9	100	109
FileOutputStreamFilefile	2/2	3364	>10	86	1	8	51	60
FileReaderFilefile	2/2	3365	>10	37	2	7	13	20
FileStringname	2/2	3363	>10	169	1	7	155	163
FileWriterFilefile	2/2	3366	>10	40	1	8	28	36
FileWriterLPT1	2/2	3363	6	139	1	7	89	96
GridBagConstraints	1/1	8402	>10	3241	1	19	323	342
GridBagLayout	1/1	8401	>10	1	1	0	1	1
GroupLayoutContainerhost	4/2	6436	>10	24	1	10	26	36
ImageIconStringfilename	2/2	8277	>10	495	1	13	154	167
InputStreamReaderInputStreamin	3/3	3363	>10	90	4	7	177	184
JButtonStringtext	2/2	6434	>10	117	1	9	85	95
JCheckBoxStringtext	2/2	8401	>10	134	2	18	50	68
JformattedTextFieldAbstractFormatter	3/2	10700	>10	2048	4	21	101	122
JFormattedTextFieldFormatterformatter	2/2	9783	>10	67	2	15	85	100
JTableObjectnameObjectdata	3/3	8280	>10	109	2	13	129	142

Evaluation

Benchmarks	Size	#Deriv.	Rank	Total	Rank	Prove [ms]	Recon [ms]	Total [ms]
AWTPermissionStringname	2/2	5615	>10	101	1	8	125	133
BufferedInputStreamFileInputStream	3/2	3364	>10	45	1	7	46	53
BufferedOutputStream	3/2	3367	>10	18	1	7	11	19
BufferedReaderFileReaderfileReader	4/2	3364	>10	69	1	7	43	50
BufferedReaderInputStreamReader	4/2	3364	>10	66	1	7	42	49
BufferedReaderReaderin	5/4	4094	>10	4760	6	7	237	244
ByteArrayInputStreambytebuf	4/4	3366	>10	94	>10	4	18	22
ByteArrayOutputStreamintsize	2/2	3363	>10	51	2	8	63	70
DatagramSocket	1/1	3246	>10	74	1	7	80	88
DataInputStreamFileInput	3/2	3364	>10	20	1	6	46	52

Evaluation

Benchmarks	Size	#Deriv.	Rank	Total	Rank	Prove [ms]	Recon [ms]	Total [ms]
AWTPermissionStringname	2/2	5615	>10	101	1	8	125	133
BufferedInputStreamFileInputStream	3/2	3364	>10	45	1	7	46	53
BufferedOutputStream	3/2	3367	>10	18	1	7	11	19
BufferedReaderFileReaderfileReader	4/2	3364	>10	69	1	7	43	50
BufferedReaderInputStreamReader	4/2	3364	>10	66	1	7	42	49
BufferedReaderReaderin	5/4	4094	>10	4760	6	7	237	244
ByteArrayInputStreambytebuf	4/4	3366	>10	94	>10	4	18	22
ByteArrayOutputStreamintsize	2/2	3363	>10	51	2	8	63	70
DatagramSocket	1/1	3246	>10	74	1	7	80	88
DataInputStreamFileInput	3/2	3364	>10	20	1	6	46	52

On-going Work: Repairing

- In addition to finding the best expression, use InSynth to repair the existing expression

Sequence of Streams

```
def main(args:Array[String]) = {
  var body:String = "email.txt"
  var sig:String = "signature.txt"
  var inStream:SeqInStr = new SeqInStr( sig,  body)
    …
}
```

Type Mismatch

// **new** SeqInStr: FileInStr → FileInStr → SeqInStr

Sequence of Streams

```
def main(args:Array[String]) = {
  var body:String = "email.txt"
  var sig:String = "signature.txt"
  var inStream:SeqInStr = new SeqInStr( sig,  body)
    ...
}
```

Type Mismatch

We propose polynomial algorithm that finds the best repair

```
// new SeqInStr: FileInStr → FileInStr → SeqInStr
```

Sequence of Streams

```
def main(args:Array[String]) = {
  var body:String = "email.txt"
  var sig:String = "signature.txt"
  var inStream:SeqInStr = new SeqInStr( sig,  body)
    ...
}
```

Backbone Expression

// **new** SeqInStr: FileInStr → FileInStr → SeqInStr

Sequence of Streams

```
def main(args:Array[String]) = {
  var body:String = "email.txt"
  var sig:String = "signature.txt“
  var inStream:SeqInStr =  (?)  new SeqInStr( (?) sig,  (?) body)
    …
}
```

```
// new SeqInStr: FileInStr → FileInStr → SeqInStr
```

Sequence of Streams

```
def main(args:Array[String]) = {
   var body:String = "email.txt"
   var sig:String = "signature.txt"
   var inStream:SeqInStr =  (?)  new SeqInStr((λx. new FileInStr(x)) sig,  (?) 
body)
      ...
}
```

Synthesize

// **new** SeqInStr: FileInStr → FileInStr → SeqInStr

Sequence of Streams

```
def main(args:Array[String]) = {
   var body:String = "email.txt"
   var sig:String = "signature.txt“
   var inStream:SeqInStr =  (?)  new SeqInStr((λx. new FileInStr(x)) sig,  (?)
body)
     …
}
```

Use Γ to synthesize function:

$(\lambda x{:}\text{String. } \textbf{new} \text{ FileInStr}(x)) : \text{String} \rightarrow \text{FileInStr}$

Constraint: Function “body” must contain exactly one variable “x”

// **new** SeqInStr: FileInStr → FileInStr → SeqInStr

Sequence of Streams

```
def main(args:Array[String]) = {
  var body:String = "email.txt"
  var sig:String = "signature.txt“
  var inStream:SeqInStr =  (?)  new SeqInStr(new FileInStr(sig),  (?) body)
    …
}
```

Use Γ to synthesize function:

$$(\lambda x:\text{String}.\ \textbf{new}\ \text{FileInStr}(x)) : \text{String} \rightarrow \text{FileInStr}$$

Constraint: Function “body” must contain exactly one variable “x”

// **new** SeqInStr: FileInStr → FileInStr → SeqInStr

Sequence of Streams

```
def main(args:Array[String]) = {
  var body:String = "email.txt"
  var sig:String = "signature.txt“
  var inStream:SeqInStr =  (?)  new SeqInStr(new FileInStr(sig),
                                        (λx. new FileInStr(x)) body)
    …
}
```

// **new** SeqInStr: FileInStr → FileInStr → SeqInStr

Sequence of Streams

```
def main(args:Array[String]) = {
  var body:String = "email.txt"
  var sig:String = "signature.txt"
  var inStream:SeqInStr =  (?)  new SeqInStr(new FileInStr(sig), new FileInStr(body))
    …
}
```

// **new** SeqInStr: FileInStr → FileInStr → SeqInStr

Sequence of Streams

```
def main(args:Array[String]) = {
  var body:String = "email.txt"
  var sig:String = "signature.txt"
  var inStream:SeqInStr = (λx. x) new SeqInStr(new FileInStr(sig), new FileInStr(body))
    …
}
```

// **new** SeqInStr: FileInStr → FileInStr → SeqInStr

Sequence of Streams

```
def main(args:Array[String]) = {
  var body:String = "email.txt"
  var sig:String = "signature.txt"
  var inStream:SeqInStr = new SeqInStr(new FileInStr(sig), new FileInStr(body))
    ...
}
```

```
// new SeqInStr: FileInStr → FileInStr → SeqInStr
```

Conclusion

- Code Completion = Type Inhabitation
- InSynth: Interactive Synthesis of Code Snippets
- Eclipse plugin (part of Scala IDE EcoSystem)
- Website

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

- Repairing Code:
  - Polynomial Algorithm the finds the best solution