Complete Completion using Types and Weights

Tihomir Gvero, Viktor Kuncak, Ivan Kuraj and Ruzica Piskac

Motivation

- Large APIs and libraries
  ~ 4000 classes in Java 6.0 standard library
- Using those APIs (for the first time) can be
  - Tedious
  - Time consuming
- Developers should focus on solving creative tasks
- Manual Solution
  - Read Documentation
  - Inspect Examples
- Automation = Code synthesis + Code completion

Our Solution

- **InSynth**: Interactive Synthesis of Code Snippets
- Input:
  - Scala partial program
  - Cursor point
- We automatically extract:
  - Declarations in scope (with/without statistics from corpus)
  - Desired type
- Algorithm
  - Complete
  - Efficient – output N expressions in less than T ms
  - Effective – favor useful expressions over obscure ones
  - Generates expressions with higher order functions
- Output
  - Ranked list of expressions

Sequence of Streams

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

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)
```

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))
    …
}
```

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

TreeFilter (HOF)

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

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))
```

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))
```

TreeFilter (HOF)

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

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

COMPLETION = INHABITATION

COMPLETION = INHABITATION

def m_1: T_1
…
def m_n: T_n

val a: T = ?

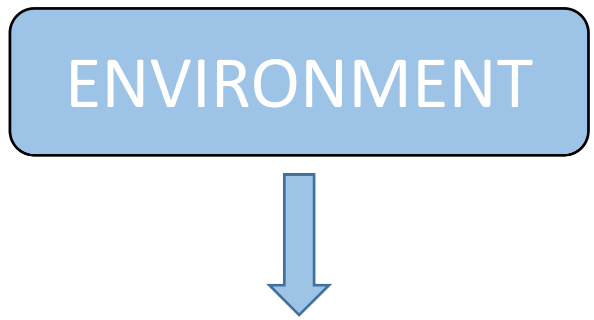
COMPLETION = INHABITATION

def m_1: T_1
…
def m_n: T_n

Γ={ m_1: T_1,…, m_n: T_n}

val a: T = ?

COMPLETION = INHABITATION



def m_1: T_1

...

def m_n: T_n

Γ={ m_1: T_1,..., 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

Simply Typed Lambda Calculus

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

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

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

Simply Typed Lambda Calculus

Simply Typed Lambda Calculus

$$\Gamma \vdash \; ? : \mathsf{T}$$

Simply Typed Lambda Calculus

Backward Search

$\Gamma \vdash \, ? : \mathrm{T}$

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

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

Infinitely many

Simply Typed Lambda Calculus

No bound on types in derivation tree(s).

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

Infinitely many

Long Normal Form

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

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

Comparison between LNF and classic APP

OLD

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

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

NEW

Comparison between LNF and classic APP

We derive EXPRESSION from Γ

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

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

Comparison between LNF and classic APP

We derive EXPRESSION from Γ

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

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

DECLARATION from Γ

Long Normal Form

Long Normal Form

$\Gamma \vdash \ ? : T$

Long Normal Form

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

Long Normal Form

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

Narrows the search space

Only one

Long Normal Form

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

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

Long Normal Form

$$\vdots$$

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

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

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

Long Normal Form

Finitely many types in derivation tree(s)

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

Algorithm

- Algorithm builds finite graph (with cycles) that
  - Represents all (infinitely many) solutions
  - Later we use it to construct expressions
- Algorithm Properties
  - Graph generation terminates
    - Type inhabitation is decidable
  - Complete - generates all solutions
  - PSPACE-complete

Subtyping

Classic

A <: B

Subtyping

Classic

A <: B

Coercion

coerc: $A \to B$

Subtyping

Classic → **Coercion**

A <: B → coerc: A $\to$ B

class FileInStr **extends** InStr {…} → coerc: FileInStr $\to$ InStr

Subtyping

Classic	**Coercion**
A <: B	coerc: A → B
class FileInStr **extends** InStr {…}	coerc: FileInStr → InStr

new SeqInStr(coerc(**new** FileInStr(sig)), coerc(**new** FileInStr(body)))

Subtyping

Classic — **Coercion**

A <: B ⇒ coerc: $A \to B$

class FileInStr **extends** InStr {…} ⇒ coerc: FileInStr $\to$ InStr

new SeqInStr(**new** FileInStr(sig), **new** FileInStr(body))

Types

Classic Types

- Simple

  Int, Bool, String, List[Int]

Types

Classic Types

- Simple

  Int, Bool, String, List[Int]

Succinct types

- Simple

  Int, Bool, String, List[Int]

Types

Classic Types

- Simple

  Int, Bool, String, List[Int]

- Function
  - Preserves argument duplicates
  - Preserves argument order

  $Int \to Int \to Bool \to Long$

Succinct types

- Simple

  Int, Bool, String, List[Int]

Types

Classic Types

- Simple

  Int, Bool, String, List[Int]

- Function
  - Preserves argument duplicates
  - Preserves argument order

  Int $\rightarrow$ Int $\rightarrow$ Bool $\rightarrow$ Long

Succinct types

- Simple

  Int, Bool, String, List[Int]

- Function
  - No duplicates
  - No order

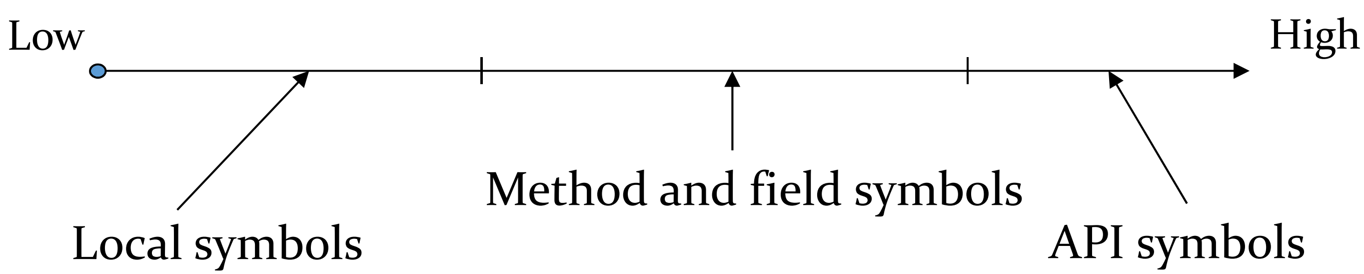
  {Int, Bool} $\rightarrow$ Long

Environment

- Classical environment
  - Declarations
  - Classic Types
- Succinct environment
  - Only succinct types
- Environment Translation
  - Shrinks environment
  - e.g. 3300 declarations to 1780 succinct types
  - We generate the graph on average in 10ms
- Reduces the search space

Weights and Corpus

- Weight of a **declaration** based on:
  - **Frequency**
    - Corpus based on 18 Scala projects (e.g. Scala compiler)
    - Over 7500 declarations, and over 90000 uses
    - Higher the frequency, lower the weight
  - **Proximity**



Algorithm with Weights

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

Algorithm with Weights

Choice based on WEIGHT

↓

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

Algorithm with Weights

Choice based on **WEIGHT**

$$\vdots$$

APP

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

ABS

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

APP

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

Ranking based on $w(f(\lambda\, x_1{:}T_1.e)) = w(f) + w(x_1) + w(e)$

Benchmarks

- 50 Java examples translated into Scala
  - Illustrate correct usage of API functions
- We generalized the import statements
  - To include more declarations
- In every example:
  1. Arbitrarily chose some expression
  2. Removed it
  3. Marked it as goal expression
  4. Measure whether InSynth can recover it

Results

- **Without weights** expected expression appears
  - Among top 10 suggestions in only 4 benchmarks (8%)
- With weights (only **proximity)**
  - Among top 10 suggestions in 48 benchmarks (**96%**)
  - As a top suggestion in 26 benchmarks (52%)
- With weights (**proximity + frequency**)
  - Among top 10 suggestions in 48 benchmarks (**96%**)
  - As a top suggestion in 32 benchmarks (**64%**)
- Average execution time **145ms**

A Sample of State of the Art

- Code completion in IDEs (Eclipse, Visual Studio, IntelliJ)
  - Mostly single declarations
  - Simple expressions
- M. Mezini et al (FSE ’09): Code recommenders
  - Suggests: **Declarations** based on API call statistics
- T. Xie et al (ASE ’07): PARSEWeb
  - Query: Source and Desired type
  - Suggests: **Code examples** based on corpus
- E. Yahav et al (OOPSLA ‘12): Prime
  - Query: Partial program
  - Suggests: **Code snippets** based on **temporal specifications**
- S. Gulwani et al (PLDI ’12): Type-directed completion of partial expressions.
  - Query: Partial Expression
  - Suggests: **Complete expressions** based on **type similarity metrics**

Conclusion

- Code Completion = Type Inhabitation
- InSynth: Interactive Synthesis of Code Snippets
- Our synthesis algorithm is:
  - Complete
  - Efficiency
  - Effective
- Eclipse plugin (part of Scala IDE EcoSystem)
- Website

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

Thank you!

Succinct Calculus

$$\text{ABS}\quad \frac{\Gamma \cup S \vdash t: T}{\Gamma \vdash S \rightarrow T}$$

$$\text{APP}\quad \frac{\{T_1,\ldots,T_n\} \rightarrow T \in \Gamma \qquad \Gamma \vdash T_1 \ \ \ldots \ \ \Gamma \vdash T_n}{\Gamma \vdash \ @\{T_1,\ldots,T_n\}:T}$$