Sana: Languages à la carte

Amanj Sherwany
sherwany.amanj@gmail.com

Joint work with: Nate Nystrom
nate.nystrom@usi.ch

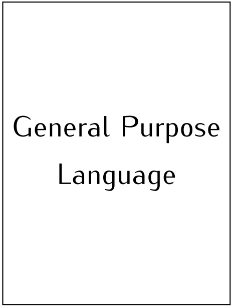
Scala-Montreal Meetup
Functional Programming Montréal Meetup

If only X programming language had this feature!

If only X programming language had this feature!

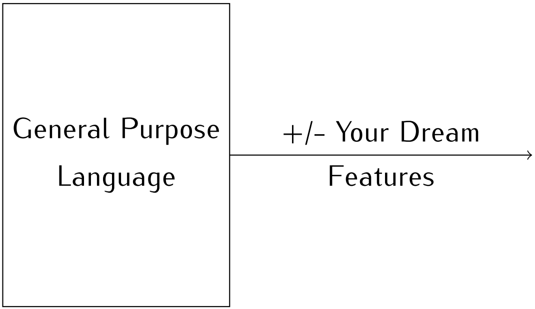
If you have ever had this wish, you are in the right place!

The "Perfect" Programming Language



General Purpose Language

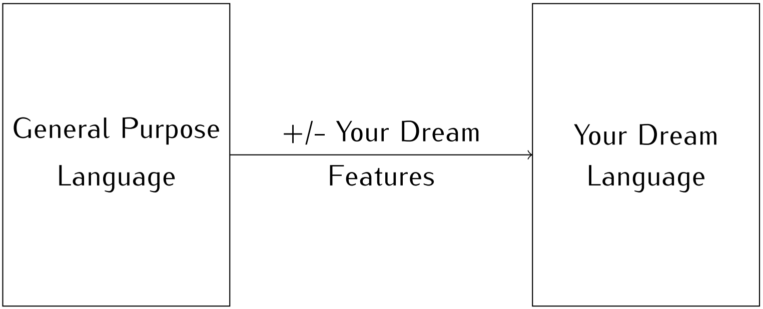
The "Perfect" Programming Language



General Purpose Language

+/- Your Dream Features

The "Perfect" Programming Language



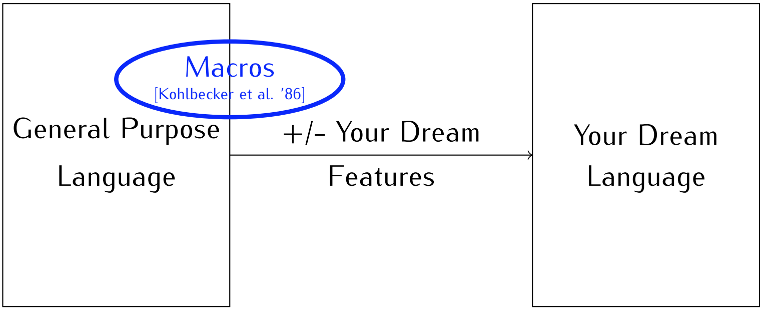
General Purpose Language

+/- Your Dream Features

Your Dream Language

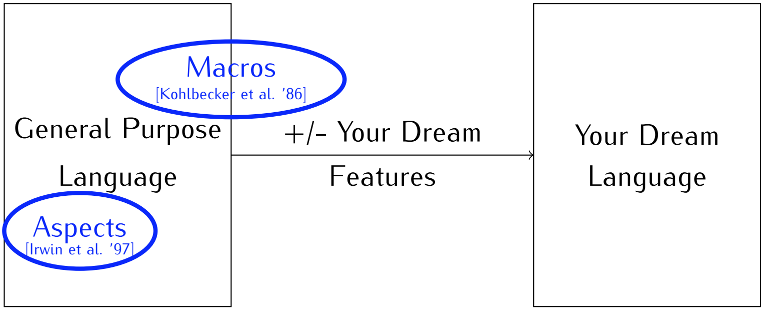
The "Perfect" Programming Language

In traditional approaches, features can be *added* using:



The "Perfect" Programming Language

In traditional approaches, features can be *added* using:



Macros
[Kohlbecker et al. '86]

General Purpose Language

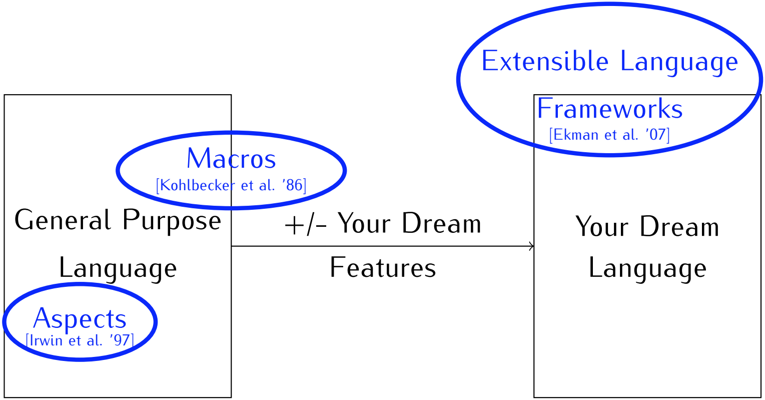
Aspects
[Irwin et al. '97]

+/- Your Dream Features

Your Dream Language

The "Perfect" Programming Language

In traditional approaches, features can be *added* using:



Extensible Language Frameworks [Ekman et al. '07]

Macros [Kohlbecker et al. '86]

General Purpose Language

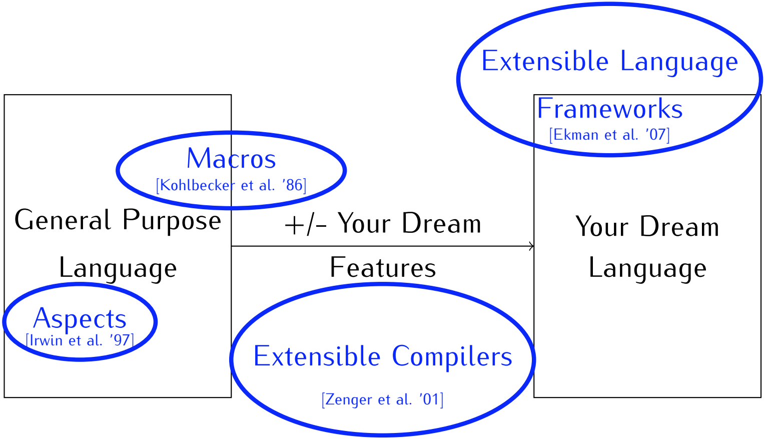
+/- Your Dream Features

Your Dream Language

Aspects [Irwin et al. '97]

The "Perfect" Programming Language

In traditional approaches, features can be *added* using:



But what if we want to remove a feature!?

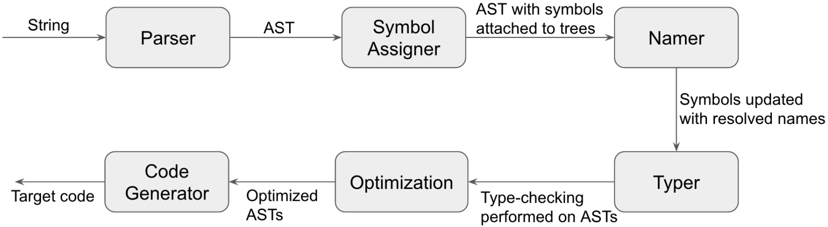
But what if we want to remove a feature!?

Up until now, we were out of luck, but not anymore!

Traditional Compilers

- Are built incrementally
- Existing features cannot be removed easily
- A massive amount of coupling
- The smallest unit is a compilation phase

Traditional Compilers


String
Parser
AST
Symbol Assigner
AST with symbols attached to trees
Namer
Symbols updated with resolved names
Typer
Type-checking performed on ASTs
Optimization
Optimized ASTs
Code Generator
Target code

Wait, can't we do better?

Sana Overview

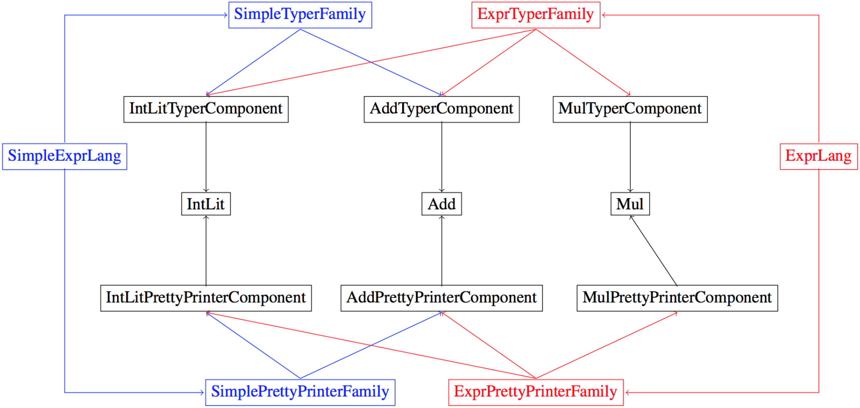
- A fully modular and extensible framework
- Provides an easy way to remove existing features
- The smallest unit is a transformation component (more on this later)

Sana Overview, *Continued*

AST node	Transformation Families (i.e. compilation phases)			
	Naming	Typing	Code Generation	Closure Conversion
lambda	X	X		X
variable	X	X	X	
method	X	X	X	X
literal			X	
class	X	X	X	X
...	...	...	...	...

Demo

Our Goal


SimpleTyperFamily
ExprTyperFamily
IntLitTyperComponent
AddTyperComponent
MulTyperComponent
SimpleExprLang
ExprLang
IntLit
Add
Mul
IntLitPrettyPrinterComponent
AddPrettyPrinterComponent
MulPrettyPrinterComponent
SimplePrettyPrinterFamily
ExprPrettyPrinterFamily

AST

```
// Expr is an abstract syntax component,
// and is the supertype of all expressions
trait Expr
case class IntLit(value: Int) extends Expr
case class Add(left: Expr, right: Expr) extends
   Expr
case class Mul(left: Expr, right: Expr) extends
   Expr

// The types
trait Type
case object NoType extends Type
case object IntType extends Type
case class ErrorType(error: String) extends Type
```

Open Classes

```
trait Expr {
  private var attributes: Map[String, Any] =
      Map.empty
  def getAttr[V](k: String, default: V) =
    attributes.getOrElse(k, default)
      .asInstanceOf[V]

  def setAttr[V](k: String, v: V) =
    attributes += (k -> v)
}
```

Open Classes, *Continued*

```
implicit class AugmentedExpr(e: Expr) {
  // getter and setter for expression type
  def tpe: Type = e.getAttr("type", NoType)
  def tpe_=(tpe: Type) =
    e.setAttr("type", tpe)

  // getter and setter for the position
  def pos: Position =
    e.getAttr("pos", NoPosition)
  def pos_=(pos: Position) =
    e.setAttr("pos", pos)
}
```

Transofrmation Components

Pattern matching vs. partial functions

Transofrmation Components, *Continued*

```
trait TyperComponent extends
  TransformationComponent[Expr, Expr] {
  def typed(expr: Expr): Expr
}
```

Transofrmation Components, *Continued*

```
@component
trait IntLitTyperComponent extends
    TyperComponent {
  (lit: IntLit) => {
    // translates to:
    // new AugmentedExpr(lit).tpe = IntType
    lit.tpe = IntType
    lit
  }
}
```

Transofrmation Components, *Continued*

```
@component
trait AddTyperComponent extends TyperComponent {
  (add: Add) => {
    val nl = typed(add.left)
    val nr = typed(add.right)
    val ty1 = nl.tpe
    val ty2 = nr.tpe
    if(ty1 == IntType && ty2 == IntType) {
      val r = Add(nl, nr)
      r.tpe = IntType
      r
    } else {
      val r = Add(nl, nr)
      r.tpe = ErrorType(s"type mismatch: ${ty1}
         and ${ty2}")
      r
    }}}
```

Transformation Family (*Compilation Phase*)

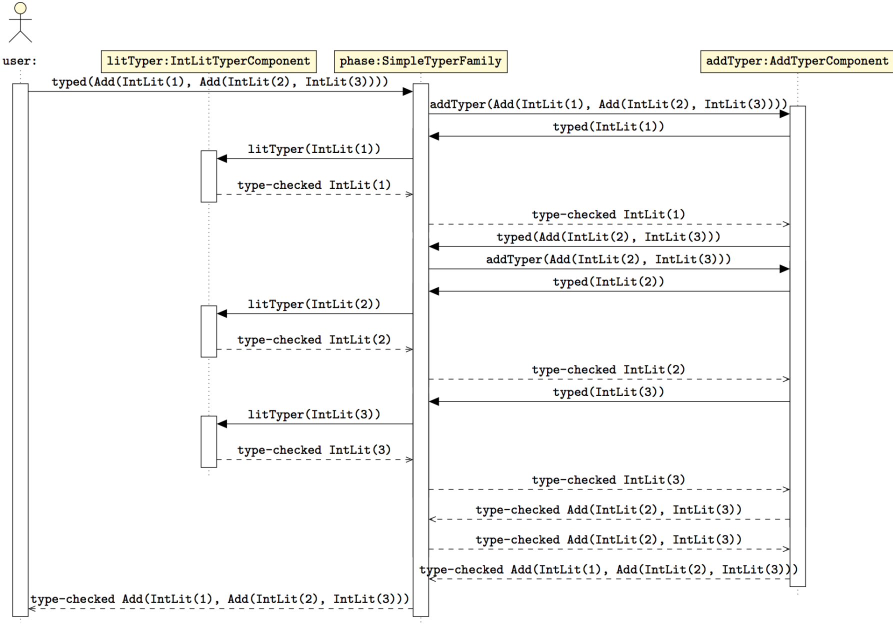
```
object TyperFamily extends TransformationFamily
    {
  def typed(expr: Expr): Expr = {
    val fun = components.reduce((x, y) => x
        orElse y)
    fun(expr)
  }

  val components: List[TyperComponent] =
    generateComponents("IntLit,Add,Mul",
        "TyperComponent" , "typed")
}
```

Language Module (*Compiler*)

```
trait ExprLang extends
    LanguageModule[Expr, String] {
  def compile = TyperFamily.typed join
                PrettyPrinterFamily.pprint
}
```

How This Works


user:
litTyper:IntLitTyperComponent
phase:SimpleTyperFamily
addTyper:AddTyperComponent
typed(Add(IntLit(1), Add(IntLit(2), IntLit(3))))
addTyper(Add(IntLit(1), Add(IntLit(2), IntLit(3))))
typed(IntLit(1))
litTyper(IntLit(1))
type-checked IntLit(1)
type-checked IntLit(1)
typed(Add(IntLit(2), IntLit(3)))
addTyper(Add(IntLit(2), IntLit(3)))
typed(IntLit(2))
litTyper(IntLit(2))
type-checked IntLit(2)
type-checked IntLit(2)
typed(IntLit(3))
litTyper(IntLit(3))
type-checked IntLit(3)
type-checked IntLit(3)
type-checked Add(IntLit(2), IntLit(3))
type-checked Add(IntLit(2), IntLit(3))
type-checked Add(IntLit(1), Add(IntLit(2), IntLit(3)))
type-checked Add(IntLit(1), Add(IntLit(2), IntLit(3)))

What Does Sana Provide?

- A core language module, called tiny
- Macros to eliminate boilerplate (generateComponents, @component)
- A skeleton for compilers (symbol table, a base AST, a base type, error reporting facilities and others)

Heavily Used Scala Features

- Partial Functions (components are partial functions)
- Function composition
- Macros
- Implicits
- And a little bit of monads

Evaluation 1: Java 1.0

Modules	Description	LOC
tiny	A small module with no Java specific components	773
calcj	Arithmetic calculator	939
primj	Primitive features of Java	2033
brokenj	break, continue, labels and switch statements	899
ooj	Packages, classes, interfaces and other OO features	6073
arrayj	Arrays, this builds on top of BrokenJ	813
arrooj	Combines OOJ and ArrayJ	786
roobustj	Exception handling	1803
dynj	Cast and instanceof	136
ppj	synchronized and volatile	446
modulej	import and class loaders	2232
bytecodej	JVM bytecode generation	2694
Total		19627

Evaluation 2: Oberon-0

- Oberon does not have classes, but it has records.
- In Oberon-0 the size of arrays is part of the type.
- Oberon-0 has type-aliasing, but Java does not.
- Simple type inference is performed for constant variables in Oberon-0.
- Oberon-0 has structural subtyping for records, while Java has nominal subtyping. There is no common supertype like `Object` in Oberon-0.
- Methods in Java can be overloaded; Oberon-0 procedures cannot.
- Only 1121 LOC!

Evaluation 3: DCCT

- Like Oberon-0, DCCT is dramatically different from Java.
- Has dictionaries but not arrays.
- Has records with constructors but not classes.
- The primitives are completely different from the ones found in Java.
- Only 783 LOC!

Evaluation 4: Performance

- We used our Java compiler to compile the standard library of Java 1.0 (which is, 14053 lines of code).
- Our experiments were run on a 2.3 GHz Intel Core i7 machine (MacBook Pro 15-inch retina display) with 16 gigabytes of RAM, running OS X 10.9.5.
- We used Scala version 2.11.7 and JVM 1.8.0_51 64-bit.
- Our compiler finished compilation and emitting the bytecode in an average of 16.25 seconds (over 5 runs), while Oracle's Java compiler could finish it in 2.5 seconds.
- Given that our compiler is an unoptimized prototype, performance is reasonable.

Source Code

Available at:

`http://github.com/amanjpro/languages-a-la-carte`

Thanks!

Questions?