GOOL: A Generic Object-Oriented Language

Jacques Carette, Brooks MacLachlan, Spencer Smith

Computing and Software Department
Faculty of Engineering
McMaster University

PEPM 2020

Observations

OO languages:

- Structurally similar

Observations

OO languages:

- Structurally similar
- Mainly *shallow* syntaxtic differences

Observations

OO languages:

- Structurally similar
- Mainly *shallow* syntaxtic differences
- Like Romance languages

Observations

OO languages:

- Structurally similar
- Mainly *shallow* syntaxtic differences
- Like Romance languages
- We tend to say similar things in all of them

The Goal

One language to express them all.

- Is it possible?
- Capture the meaning of OO programs
- DSL for domain of OO programs
- Currently targets Java, Python, C#, C++

- **mainstream**: Most potential users

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience
- **idiomatic**: For readability, understandability

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience
- **idiomatic**: For readability, understandability
- **documented**: For readability, understandability

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience
- **idiomatic**: For readability, understandability
- **documented**: For readability, understandability
- **patterns**: More efficient coding

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience
- **idiomatic**: For readability, understandability
- **documented**: For readability, understandability
- **patterns**: More efficient coding
- **expressivity**: Works for real examples

- **mainstream**: Most potential users
- **readable**: Human beings are a target audience
- **idiomatic**: For readability, understandability
- **documented**: For readability, understandability
- **patterns**: More efficient coding
- **expressivity**: Works for real examples
- **common**: Reduce code duplication

McMaster University

Approach

Start from real OO programs

McMaster University

Approach

Start from real OO programs

What we can say vs. want to say vs. need to say

Readability Features

Example: Blocks

- Semantically meaningless
- Reflect how people write programs

```
# Initialize dimensions
h = input('What is the height? ')
w = input('What is the width? ')
d = input('What is the depth? ')

# Calculate volume and surface area
v = h * w * d
sa = 2 * (h * w + w * d + h * d)
```

Some ingredients

- Variables distinct from values (viz use/mention)

Some ingredients

- Variables distinct from values (viz use/mention)
- Smart constructors for common idioms

GOOL Language

Types	`bool`, `int`, `float`, `char`, `string`, `infile` (read mode), `outfile` (write mode), `listType`, `obj`
Variables	`var`, `extVar`, `classVar`, `objVar`, `$->` (infix operator for `objVar`), `self`, [`listVar`]
Values	`valueOf` (value from variable), `litTrue`, `litFalse`, `litInt`, `litFloat`, `litChar`, `litString`, `?!`, `?&&`, `?<`, `?<=`, `?>`, `?>=`, `?==`, `?!=`, `#~`, `#/^`, `#—`, `#+`, `#-`, `#*`, `#/`, `#^`, `inlineIf`, `funcApp`, `extFuncApp`, `newObj`, `objMethodCall`, [`selfFuncApp`, `objMethodCallNoParams`]
Statements	`varDec`, `varDecDef`, `assign`, `&=`, `&+=`, `&-=`, `&++`, `&~-`, `break`, `continue`, `returnState`, `throw`, `free`, `comment`, `ifCond`, `ifNoElse`, `switch`, `for`, `forRange`, `forEach`, `while`, `tryCatch`, `block`, `body` [`bodyStatements` (single-block body), `oneLiner` (single-statement body)]
List API	`listAccess`, `at` (same as `listAccess`), `listSet`, `listAppend`, `listIndexExists`, `indexOf`, `listSlice`
Scope	`public`, `private`
Binding	`static_`, `dynamic_`
Functions	`function`, `method`, `param`, `pointerParam`, `mainFunction`, `docFunc`, [`pubMethod`, `privMethod`]
State Variables	`stateVar`, `constVar`, [`privMVar`, `pubMVar` (dynamic), `pubGVar` (static)]
Classes	`buildClass`, `docClass`, [`pubClass`, `privClass`]
Packages	`buildModule`, `fileDoc`, `docMod`, `prog`, `package`, `doxConfig`, `makefile`

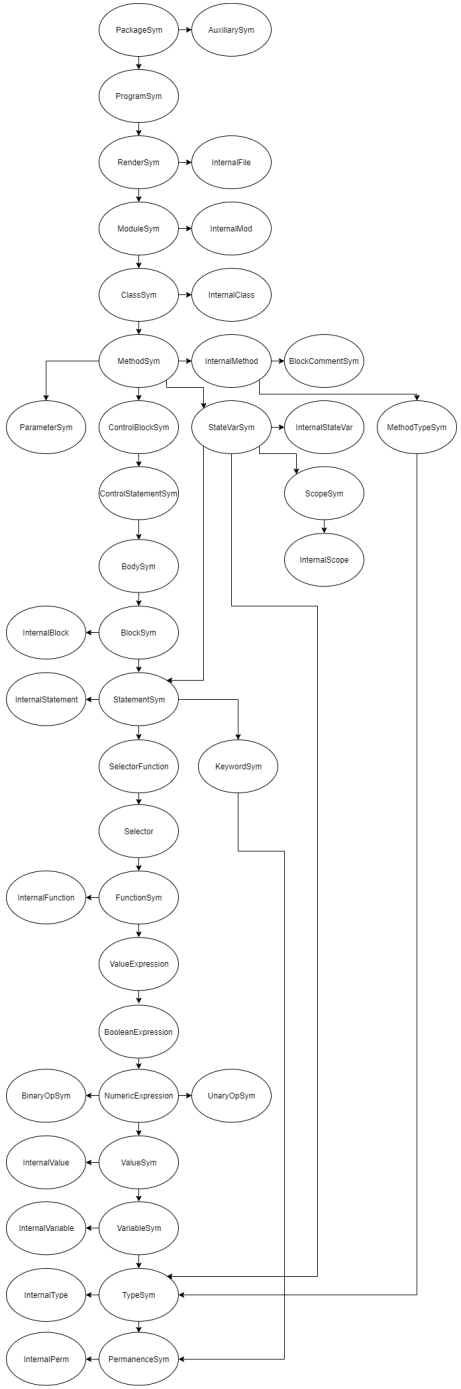
Encoding

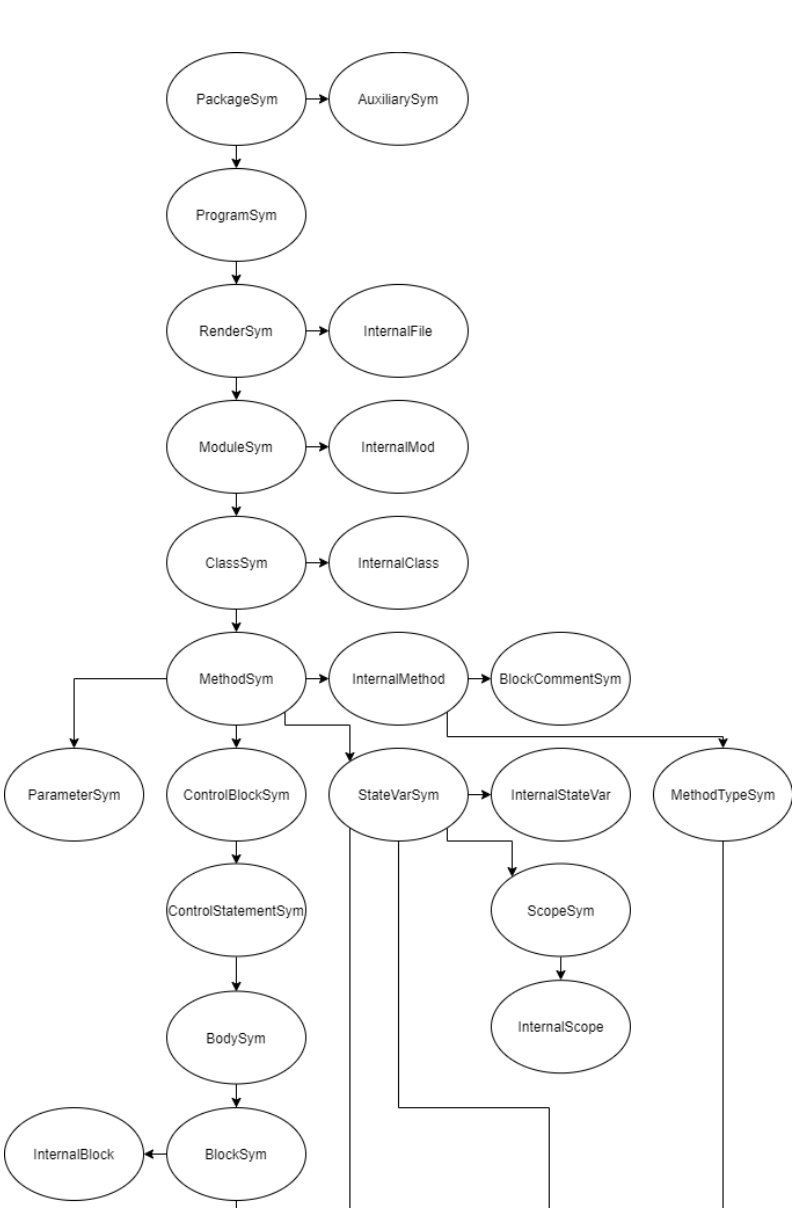
Tagless with type families – 2 Layers of abstraction

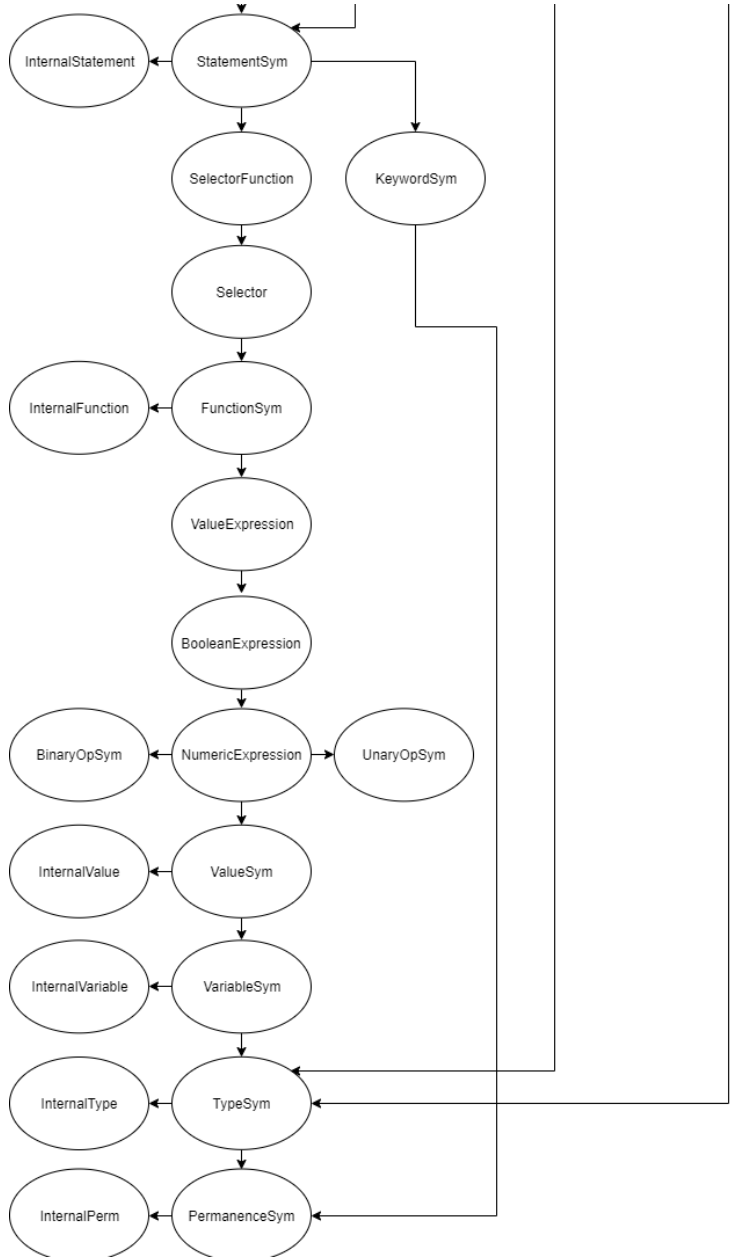
1. Over target language
2. Over language-specific representational data structures

```
class (TypeSym repr) => VariableSym repr where
  type Variable repr
  var  ::  Label -> repr (Type repr)
    -> repr (Variable repr)
```

McMaster University

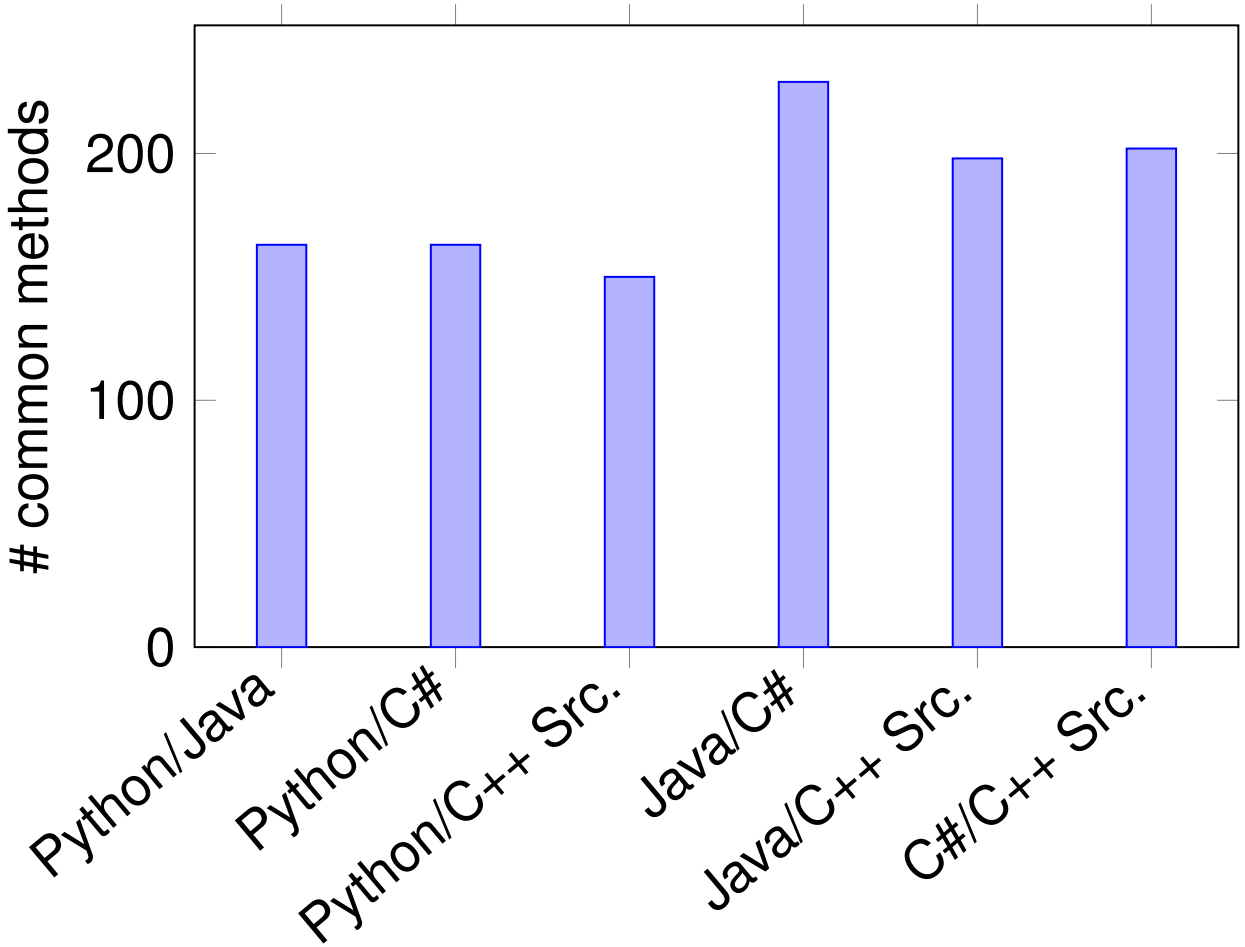






Statistics

- 43 classes, 328 methods
- 300 functions that abstract over commonalities
- 40% more common methods between Java and C# than Java and Python



Things we need/want to say

- Command line arguments
- Lists
- I/O
- Procedures with Input/Output/Both parameters
- Getters and setters
- Design patterns

Example: List Slicing

GOOL: *hideous, we know*

```
listSlice sliced (valueOf old) (Just $ litInt 1)
  (Just $ litInt 3) Nothing
```

Python:

```
sliced = old [1:3:]
```

Example: List Slicing

GOOL: *hideous, we know*

```
listSlice sliced (valueOf old) (Just $ litInt 1)
  (Just $ litInt 3) Nothing
```

Java:

```
ArrayList<Double> temp = new ArrayList<Double>(0);
for (int i_temp = 1; i_temp < 3; i_temp++) {
    temp.add(old.get(i_temp));
}
sliced = temp;
```

Example: List Slicing

GOOL: *hideous, we know*

```
listSlice sliced (valueOf old) (Just $ litInt 1)
  (Just $ litInt 3) Nothing
```

C#:

```
List<double> temp = new List<double>(0);
for (int i_temp = 1; i_temp < 3; i_temp++) {
    temp.Add(old[i_temp]);
}
sliced = temp;
```

Example: List Slicing

GOOL: *hideous, we know*

```
listSlice sliced (valueOf old) (Just $ litInt 1)
  (Just $ litInt 3) Nothing
```

C++:

```
vector<double> temp(0);
for (int i_temp = 1; i_temp < 3; i_temp++) {
    temp.push_back(old.at(i_temp));
}
sliced = temp;
```

Example: Setters

GOOL:

```
setMethod "FooClass" foo
```

Python:

```
def setFoo(self, foo):
    self.foo = foo
```

Example: Setters

GOOL:

```
setMethod "FooClass" foo
```

Java:

```
public void setFoo(int foo)  throws Exception {
    this.foo = foo;
}
```

Example: Setters

GOOL:

setMethod ”FooClass” foo

C#:

```
public void setFoo(int foo) {
    this.foo = foo;
}
```

Example: Setters

GOOL:

setMethod ”FooClass” foo

C++:

```
void FooClass::setFoo(int foo) {
    this->foo = foo;
}
```

Currently working on

- More types
- Smarter generation - ex. import statements
- Interfacing with external libraries
- User-decisions - ex. which type to use for lists?
- More patterns

Language of Design

Drasil project - http://github.com/JacquesCarette/Drasil

- Generate scientific software
- Design language lets users guide design
- GOOL is the backend

Complete Example

Projectile program

Design 1

- Documented
- Bundled inputs

Design 2

- Logging
- More modular

Conclusion

We currently use GOOL to generate some examples of scientific software (glass breakage, projectile simulation)

Together new:

- Idiomatic code generation
- Human-readable, documented code generation
- Coding patterns are language idioms

With respect to “The Goal” — It is possible