GOOL: A Generic Object-Oriented Language

Jacques Carette, Brooks MacLachlan, and Spencer Smith

Computing and Software Department
Faculty of Engineering
McMaster University

PEPM 2020

The Problem

OO languages:

- Structurally similar
- Differences are mostly syntactic
- Like Romance languages

The Goal

Generalize to one language:

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

- **mainstream**: Generate code in mainstream languages
- **readable**: Generared code is human-readable
- **idiomatic**: Generated code is idiomatic
- **documented**: Generated code is documented
- **patterns**: Express common OO patterns
- **expressivity**: Language succeeds in expressing real OO programs
- **common**: Commonalities abstracted

[One slide for each? —BM]

Principles

- What we can say vs. want to say vs. need to say (ex. Java introspection, C++templates)
- Readability features i.e. blocks
- Variables vs. values
- Smart constructors for common idioms

GOOL Language

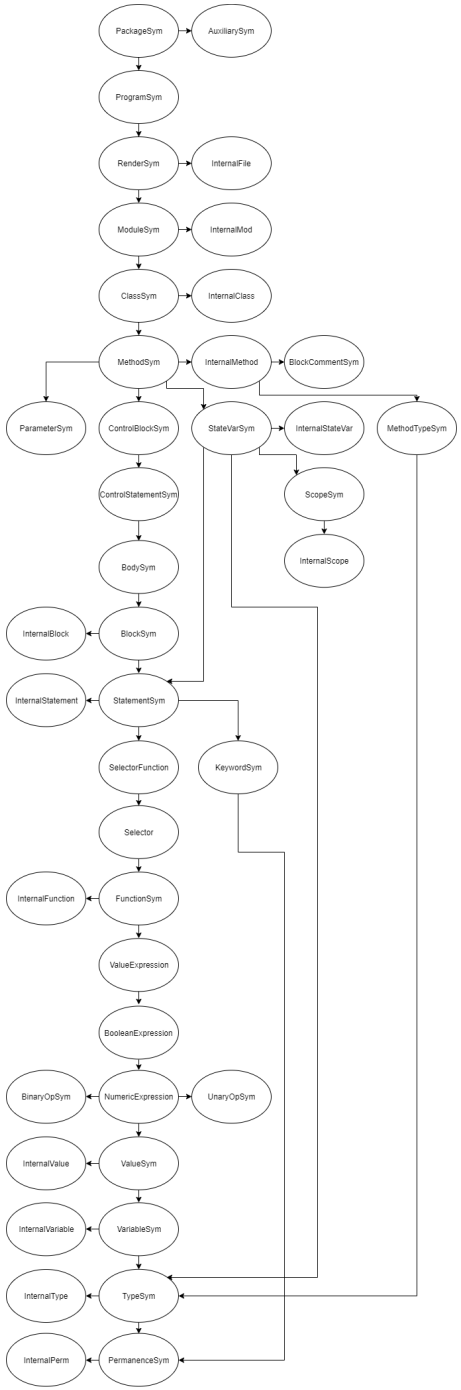
Types	`bool`, `int`, `float`, `char`, `string`, `infile` (read mode), `outfile` (write mode), `listType`, `obj`
Variables	`var`, `extVar`, `classVar`, `objVar`, `$->` (infix operator for `objVar`), `self`, [`listVar`]
Values	`valueOf` (value from variable), `litTrue`, `litFalse`, `litInt`, `litFloat`, `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

- Over target language
- Over underlying data structures

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



[Very squished to fit in frame, should we keep this? —BM]

Statistics

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

[Include the bar graph of common methods from long version of paper? —BM]

Patterns

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

[Do we want to mention all of these? A slide with sample code for each? Or maybe pick one to use as an example? GOOL code, generated code, or both? —BM]

Demo?

[Maybe show the PatternTest example like we do in the long version of the paper. Can show GOOL code and target code in all languages. Would probably want to leave the slideshow for this demo? —BM]
[I can alternatively write up a new “test” for the purpose of this demo, if you want to request some specific features to show off or if you have a specific idea —BM]

Future

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

[Split into a slide for each? Or pick a couple important ones and just do a slide for each of those? —BM]

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

[We don’t need to discuss related work in the presentation, do we? —BM]