CODE GENERATION IN DRASIL

Steven Palmer

Computing and Software
Faculty of Engineering
McMaster University

January 24, 2018
Ernie Mileta Visit

OVERVIEW

1. Current code generation design
2. Results of current generator
3. Next steps

CURRENT DESIGN

CODE SPECIFICATION

- Specification:
  - Database of symbols used in the code
  - Inputs to the program
  - Outputs of the program
  - Relations between symbols
- Built from captured knowledge
- Using the code specification, Drasil finds a path from inputs to outputs through the set of relations
- Generates code using these relations to transform inputs to outputs

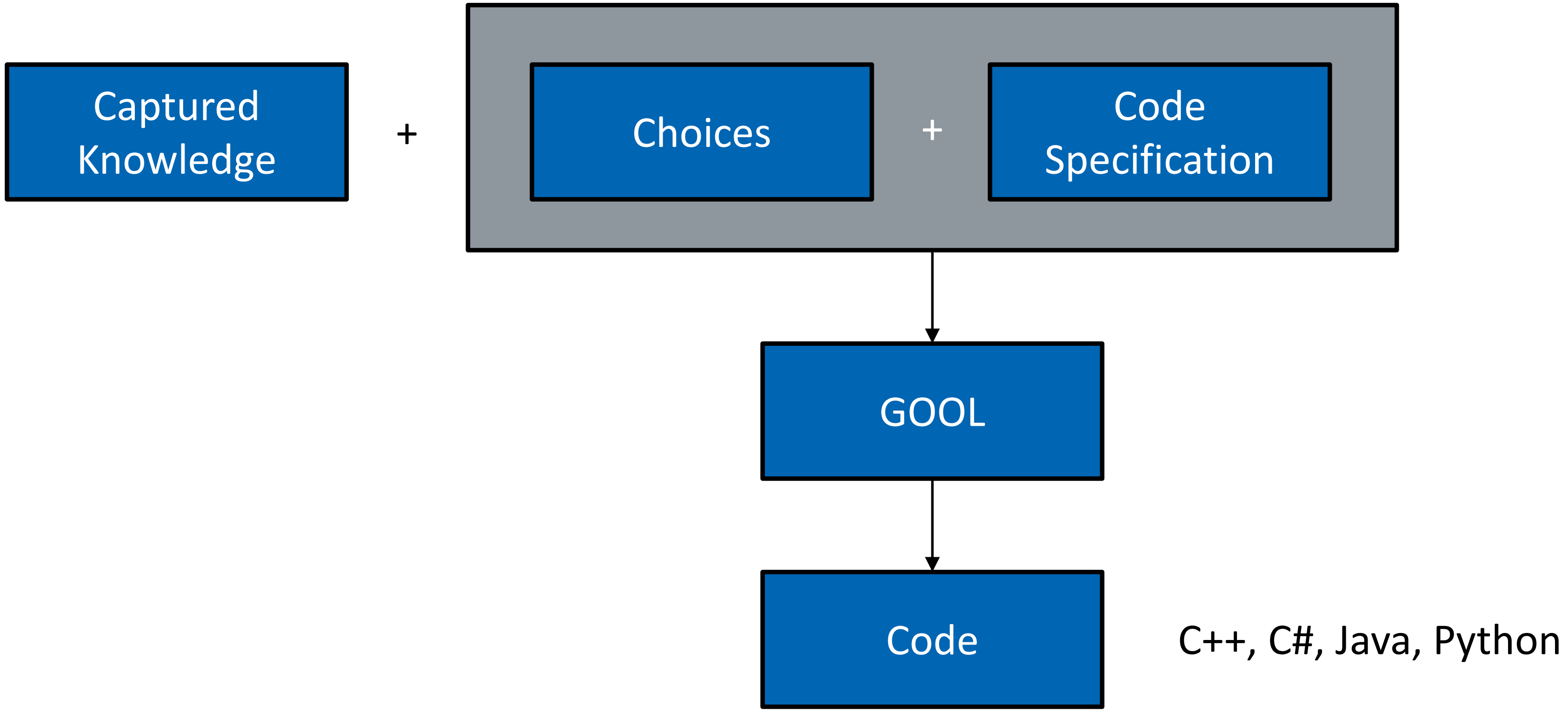
CHOICES

- Allow user to make decisions about features of the generated code
- Examples of choices:
  - Programming language?
  - Library vs program?
  - Logging?
  - Documentation in code?
  - Input/output structure?
  - Which algorithm to use?

GOOL

- Generic Object-Oriented Language
  - Developed by Jason Costabile as MEng project, 2012
  - Abstract syntax tree (AST) for OO languages
  - Integrated into Drasil and extended
- Allows rendering code in multiple OO languages
  - C++, C#, Java, Python

CURRENT DESIGN



RESULTS

GLASSBR CASE STUDY

- GlassBR:
  - Computes whether a given plate of glass will resist a blast force
  - Small: ~200 lines
  - Simple: Input -> Calculations -> Output
  - Good starting point for developing code generation in Drasil
- Currently able to generate complete working code for GlassBR
  - C++, C#, Java, and Python!

RISK OF FAILURE FROM GLASSBR THEORY

Number	DD1
Label	**Risk of Failure (B)**
Equation	$B = \frac{k}{(a \times b)^{m-1}}((E \times 1000)(h)^2)^m \times \text{LDF} \times e^J$
Description	B is the risk of failure m, k are the surface flaw parameters a, b are dimensions of the plate, where $(a > b)$ E is the modulus of elasticity h is the true thickness, which is based on the nominal thickness as shown in DD2 LDF is the Load Duration Factor, as defined in DD3 J is the stress distribution factor, as defined in DD4

RISK OF FAILURE: DRASIL CODE

$$B = \frac{k}{(a \times b)^{m-1}}((E \times 1000)(h)^2)^m \times \mathrm{LDF} \times e^J$$

```
risk_eq :: Expr
risk_eq = ((C sflawParamK) / (Grouping ((C plate_len) *
  (C plate_width))) :^ ((C sflawParamM) - 1) *
  (Grouping (C mod_elas * 1000) * (square (Grouping (C act_thick))))
  :^ (C sflawParamM) * (C lDurFac) * (exp (C stressDistFac)))
```

Computing and Software

McMaster University

RISK OF FAILURE: DRASIL CODE

$$B = \frac{k}{(a \times b)^{m-1}}((E \times 1000)(h)^2)^m \times \text{LDF} \times e^J$$

```
risk_eq :: Expr
risk_eq = ((C sflawParamK) / (Grouping ((C plate_len) *
   (C plate_width))) :^ ((C sflawParamM) - 1) *
   (Grouping (C mod_elas * 1000) * (square (Grouping (C act_thick))))
   :^ (C sflawParamM) * (C lDurFac) * (exp (C stressDistFac)))
```

RISK OF FAILURE: GENERATED CODE

C++:

```
double func_B(InputParameters &inParams, double J) {
    return ((((2.86 * (pow(10, -(53)))) / (pow(inParams.a * inParams.b, 7 - 1))) *
        (pow(((7.17 * (pow(10, 7))) * 1000) * (pow(inParams.h, 2)), 7))) *
        (pow(3 / 60, 7 / 16))) * (exp(J));
}
```

C#:

```
public static double func_B(InputParameters inParams, double J) {
    return ((((2.86 * (Math.Pow(10, -(53)))) /
        (Math.Pow(inParams.a * inParams.b, 7 - 1))) *
        (Math.Pow(((7.17 * (Math.Pow(10, 7))) * 1000) *
        (Math.Pow(inParams.h, 2)), 7))) * (Math.Pow(3 / 60, 7 / 16))) * (Math.Exp(J));
}
```

RISK OF FAILURE: GENERATED CODE

Python:

```
def func_B(inParams, J):
    return ((((2.86 * (10 ** (-(53)))) / ((inParams.a * inParams.b) **
        (7 - 1))) * ((((7.17 * (10 ** 7)) * 1000) *
        (inParams.h ** 2)) ** 7)) * ((3 / 60) ** (7 / 16))) *
        (math.exp(J))
```

Java:

```
public static double func_B(InputParameters inParams, double J) throws Exception {
    return ((((2.86 * (Math.pow(10, -(53)))) /
        (Math.pow(inParams.a * inParams.b, 7 - 1))) *
        (Math.pow(((7.17 * (Math.pow(10, 7))) * 1000) *
        (Math.pow(inParams.h, 2)), 7))) * (Math.pow(3 / 60, 7 / 16))) * (Math.exp(J));
}
```

CHOICE: FUNCTION COMMENTING

```
def func_B(inParams, J):
    # function 'func_B': risk of failure
    # parameter 'inParams':
    # parameter 'J': stress distribution factor (Function)

    return ((((2.86 * (10 ** (-(53)))) / ((inParams.a * inParams.b) **
      (7 - 1))) * ((((7.17 * (10 ** 7)) * 1000) *
      (inParams.h ** 2)) ** 7)) * ((3 / 60) ** (7 / 16))) *
      (math.exp(J))
```

CHOICE: FUNCTION LOGGING

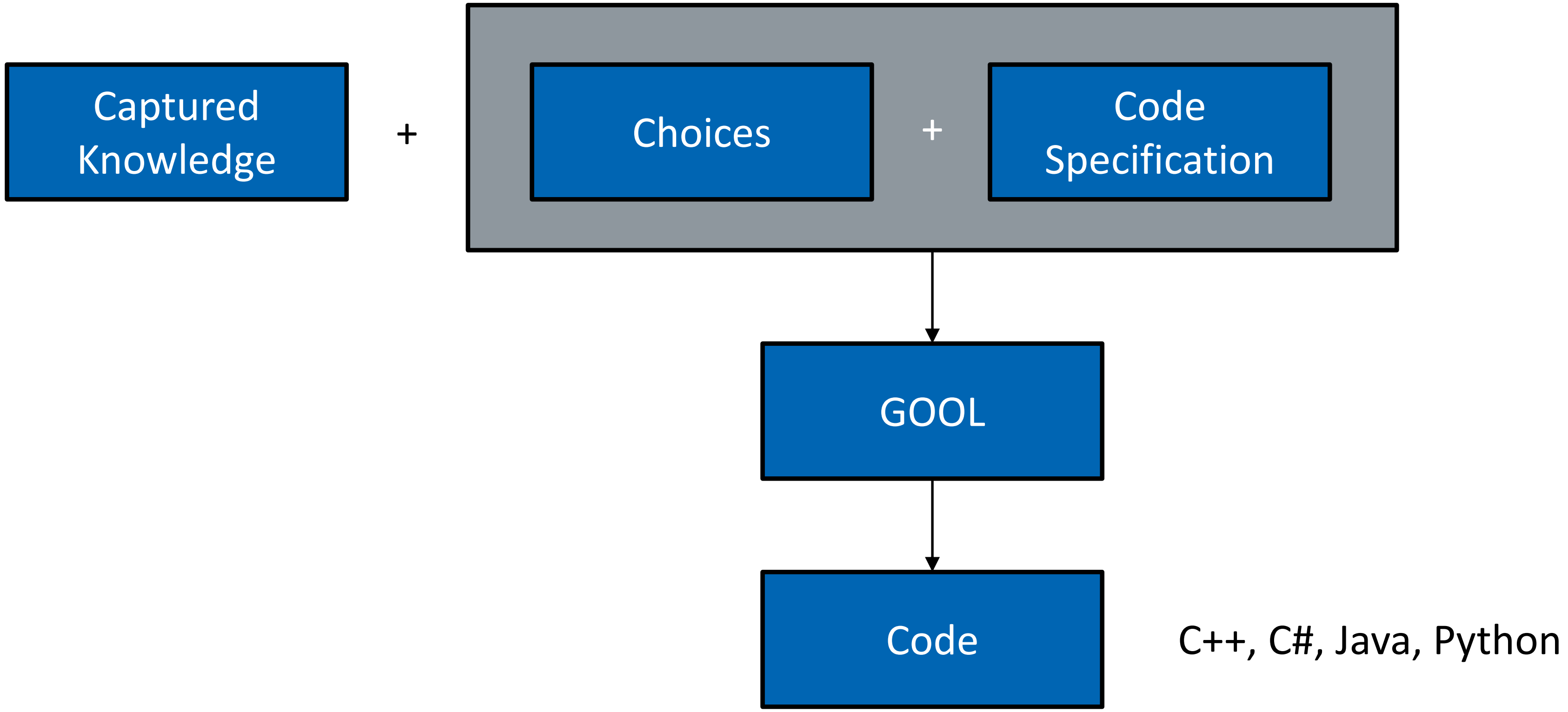
```
def func_B(inParams, J):
    # function 'func_B': risk of failure
    # parameter 'inParams':
    # parameter 'J': stress distribution factor (Function)

    outfile = open("log.txt", "w")
    print("function func_B(", end='', file=outfile)
    print(inParams, end='', file=outfile)
    print(", ", end='', file=outfile)
    print(J, end='', file=outfile)
    print(") called", file=outfile)
    outfile.close()

    return ((((2.86 * (10 ** (-(53)))) / ((inParams.a * inParams.b) **
      (7 - 1))) * ((((7.17 * (10 ** 7)) * 1000) *
      (inParams.h ** 2)) ** 7)) * ((3 / 60) ** (7 / 16))) *
      (math.exp(J))
```

NEXT STEPS

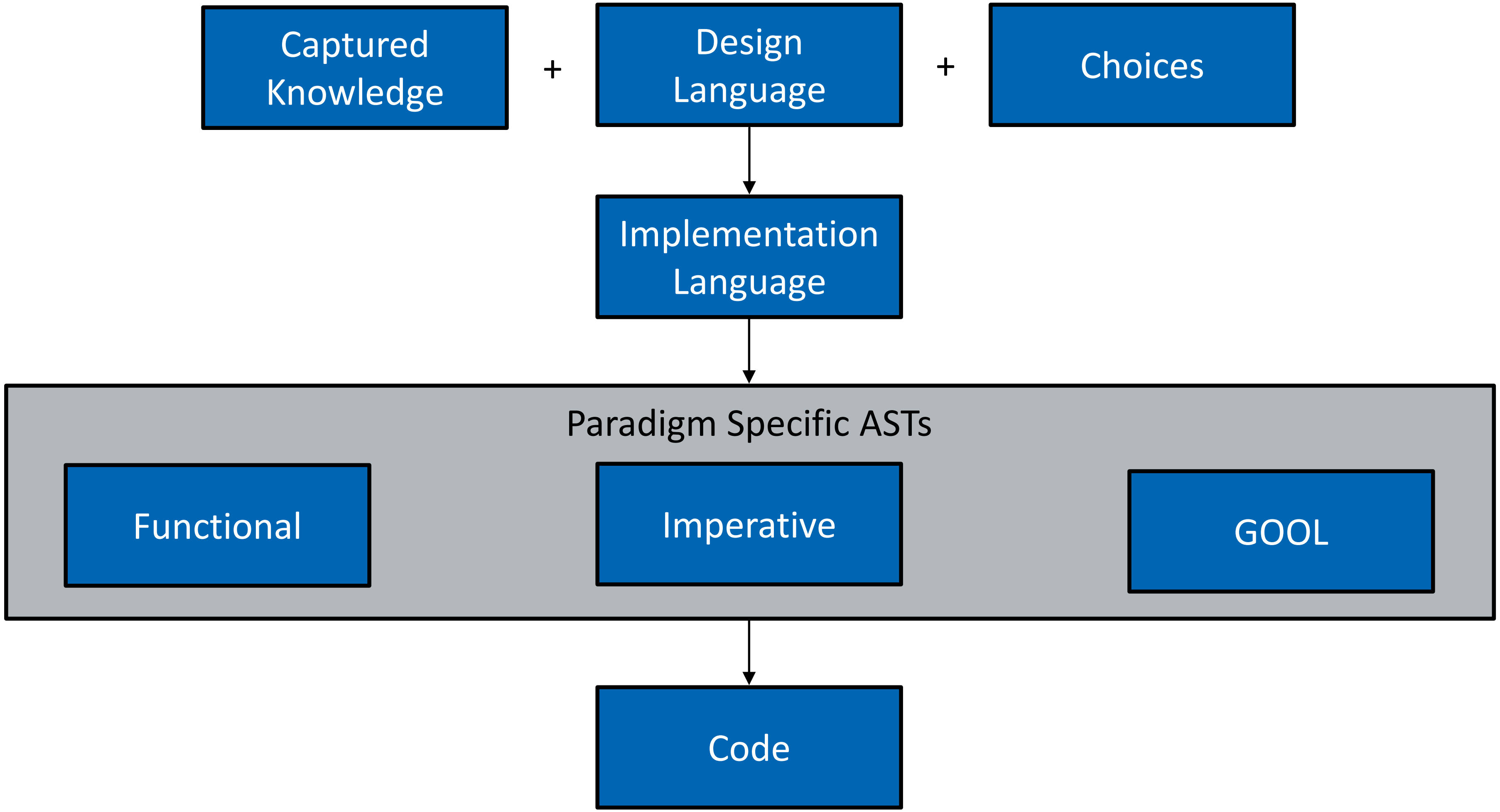
RECALL: CURRENT DESIGN



LIMITATIONS OF CURRENT DESIGN

- Works only for simple program structures:
  - Read inputs, do some (serial) computations, write outputs
- Lacks expressiveness for the user:
  - The current design is rigid
  - Tries to do too much in an automated way
  - Need a way for the user of Drasil to specify the design of the code
  - Design language

FUTURE DESIGN



THANK YOU