Design and implementation of the Meta Casanova 3 compiler back-end

Douwe van Gijn

2016-06-29

Introduction

- Games are difficult $\longrightarrow$ Casanova language
- Compilers are difficult $\longrightarrow$ Meta Casanova language
- Compilers in two parts
  1. Front-end: parse and typecheck
  2. Back-end: generate executables

Contents

Research question

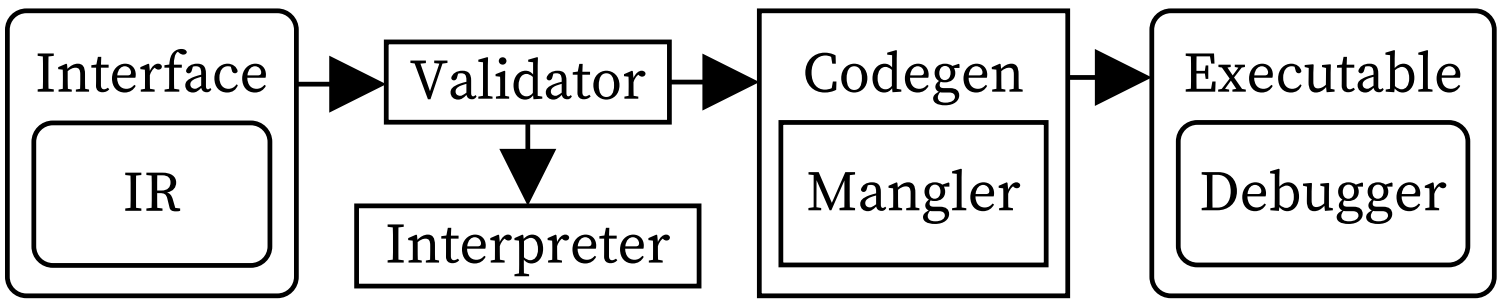
How to implement a transformation from typechecked Meta Casanova from the front-end, to executable code within the timeframe of the internship?

Requirements

- The correctness requirement
- The .NET requirement
- The multiplatform requirement
- The performance requirement

Sub-questions

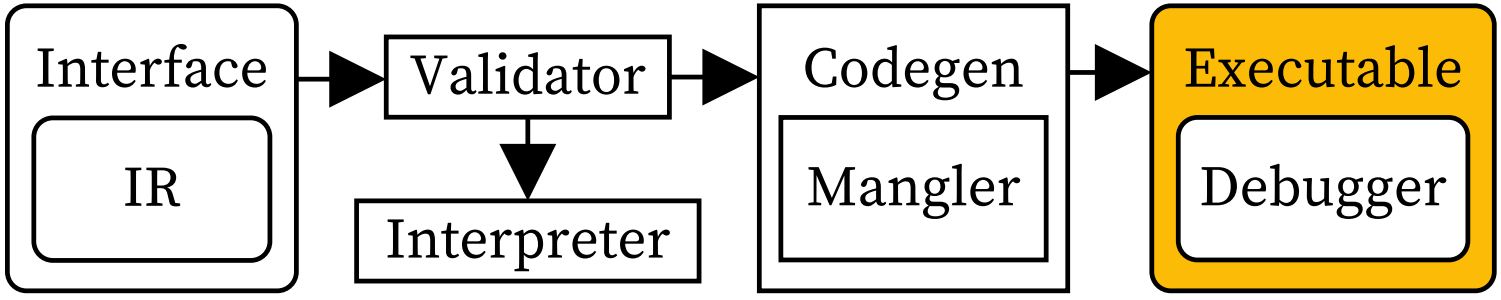
- 7 sub-questions
- Each answer implements parts of the back-end



The language question

In what language should the code generator produce its output?

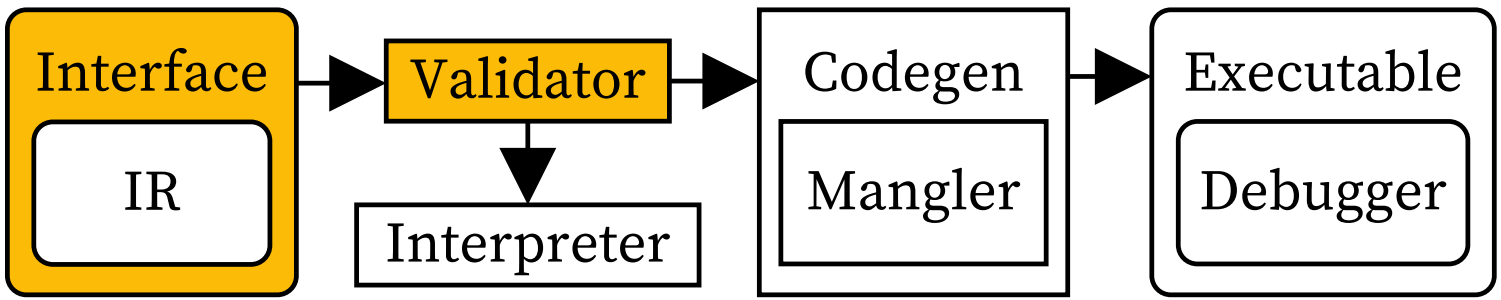
- Researched lots of langages
- Two feasable: C# or F#
- Implemented both code-models
- C# won out
  - more readable
  - easier to generate
  - faster



The interface question

What should the interface be between the front-end and the back-end?

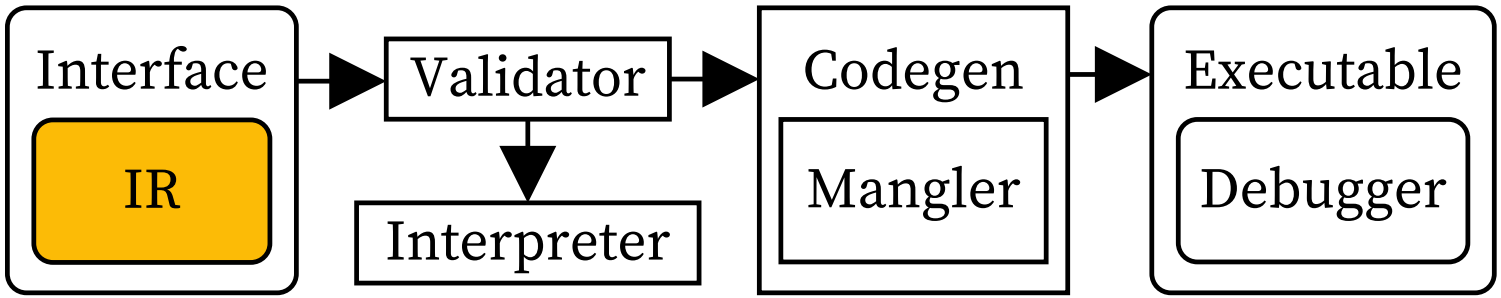
- Contains all inputs of back-end
- Validator validates invariants



The IR question

What should the Intermediate Representation of the function be?

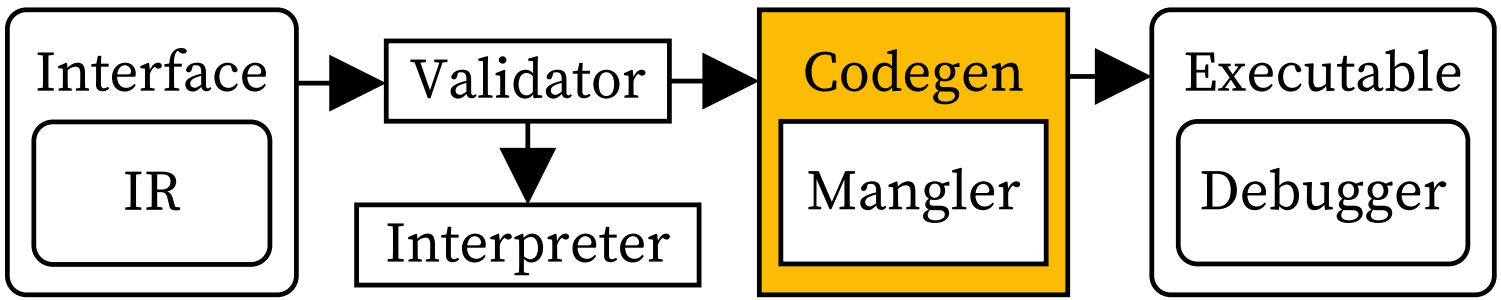
- Instruction set for MC
- Minimal and orthogonal
- Only 6 base instructions



The codegen question

How does the interface map to the output language?

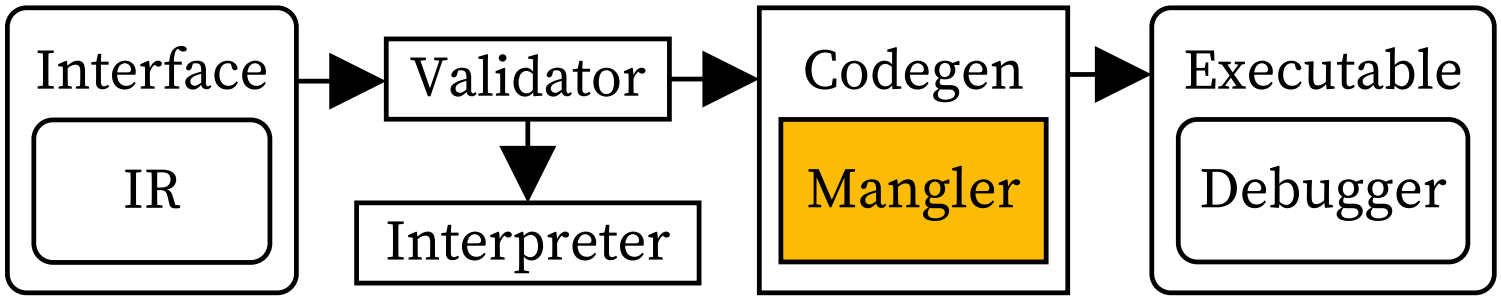
- Generates program structure
- Translates IR to C# instructions



The mangle question

How to generate names so they comply with the output language?

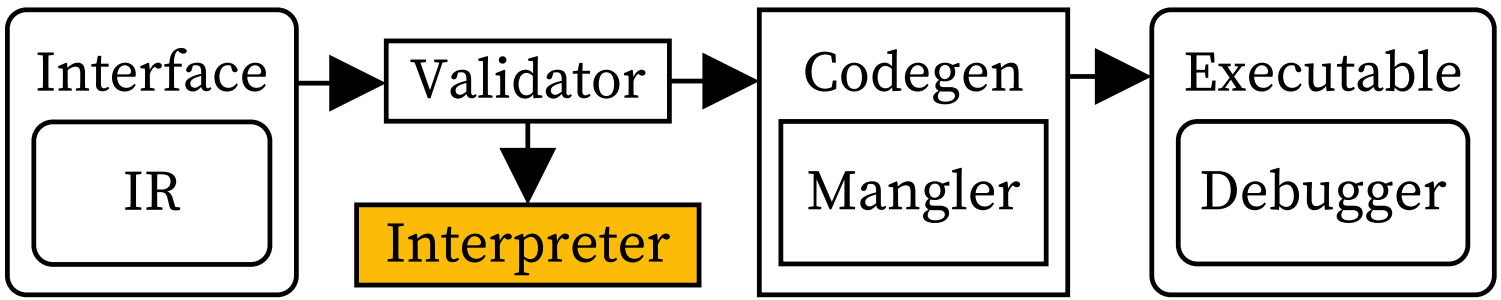
- MC identifiers $\longrightarrow$ C# identifiers
- No name conflicts



The validation question

How to validate the code-gen?

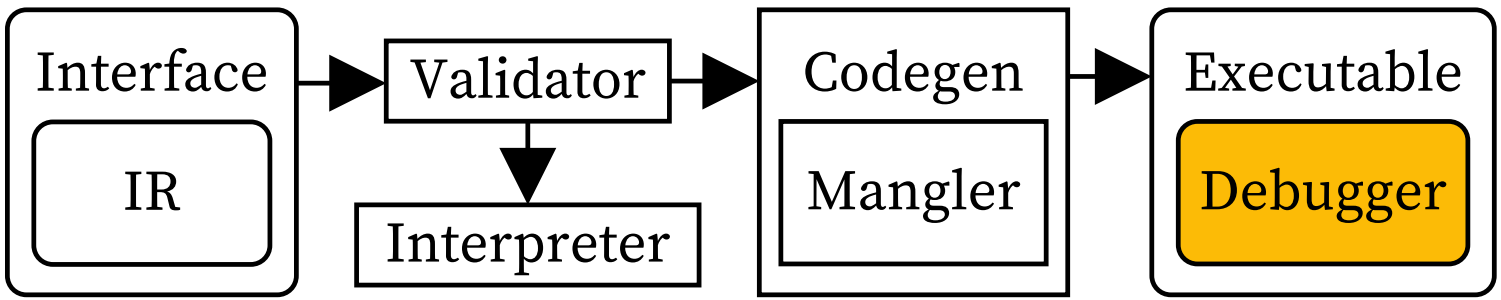
- Built an interpreter!
- Compare results of interpreter and codegen



The debug question

How to validate the test-programs?

- Interactive debugger
- Embedded in executable



Results

- The correctness requirement
- The .NET requirement
- The multiplatform requirement
- The performance requirement

The correctness & .NET requirement

Test programs

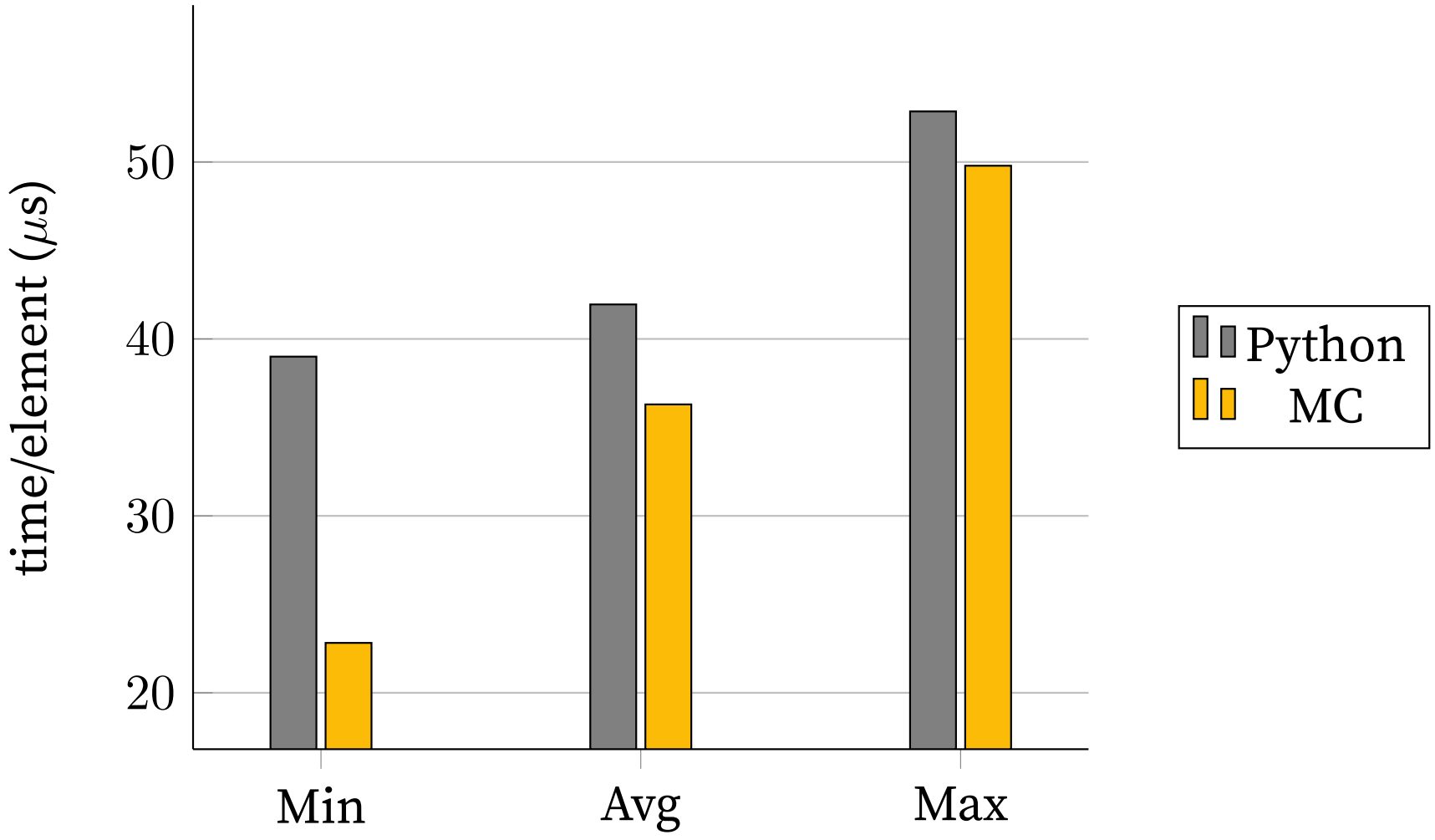
The multiplatform requirement

Microsoft .NET Compiler for windows
Mono everywhere else

The performance requirement

- Length of list
- Inductive list in MC
- Library list in Python
- 1000 lists of 1 000 000 elements

The performance requirement



Conclusion

- All requirements are met
- Working back-end within the allocated time
- Demo time!

Defence