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

Douwe van Gijn

2016-06-29

Contents

Introduction

- Video game industry
- Developing games is difficult $\longrightarrow$ Casanova language
- Compilers are difficult $\longrightarrow$ Meta Casanova language
- Casanova features implemented in MC
- Compilers in two parts
  1. Front-end: parse and typecheck
  2. Back-end: generate executable

Research question

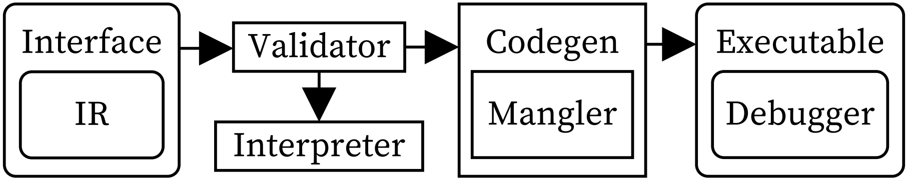
How to implement a transformation from typechecked Meta Casanova from the front-end, to executable code within the timeframe of the internship?

Requirements

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

Sub-questions

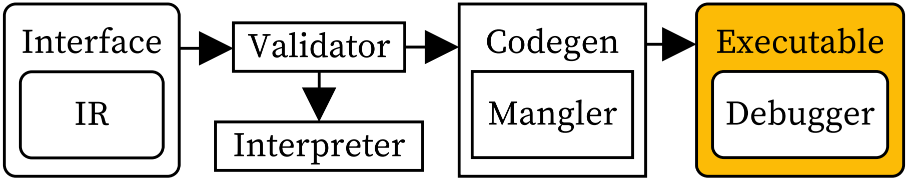
- 7 sub-questions
- Each answer implements parts of the back-end



The language question

In what language should the code generator produce its output?

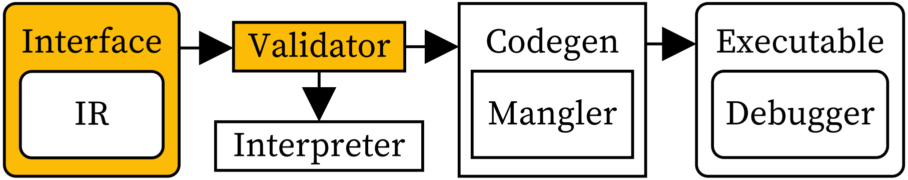
- Researched lots of langages
- Two feasable: C# and F#
- Implemented both code-models
- C# won out
    - more readable
    - easier to generate
    - faster



The interface question

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

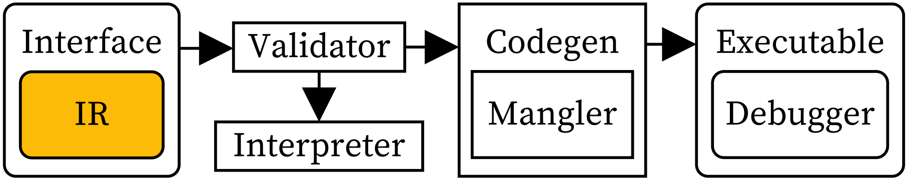
- Contains all inputs of back-end
- Attack surface
- smaller interface $\longrightarrow$ fewer representations $\longrightarrow$ fewer bugs
- Validator validates invariants
  - each identifier is defined once
  - each identifier has a type
  - no empty rules



The IR question

What should the Intermediate Representation of the functions be?

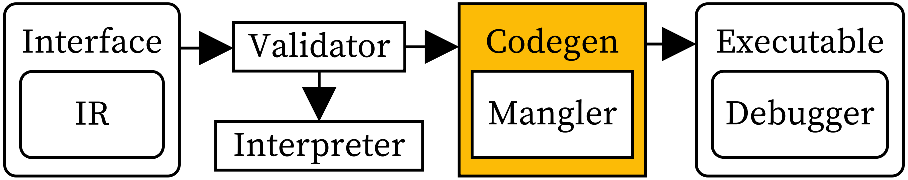
- Instruction set for MC
- Minimal and orthogonal
- 6 base instructions
- 6 .NET instructions
- Static Single Assignment (SSA) form



The codegen question

How does the interface map to the output language?

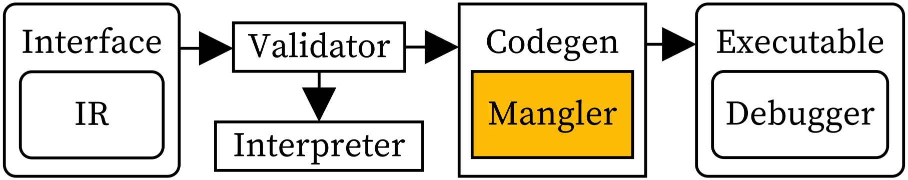
- By generating datastructures
- By generating the program structure
- Translating the IR to linear stream of C# instructions



The mangle question

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

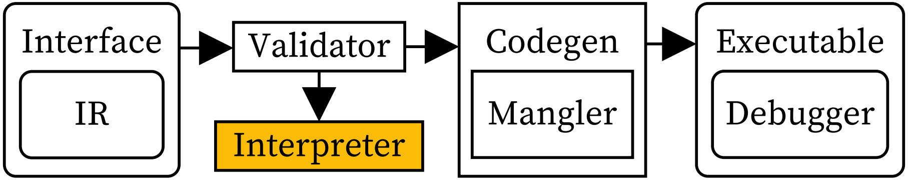
- MC identifiers $\longrightarrow$ C# identifiers
- MC identifiers: nearly all printable ASCII
- C# identifiers: Only alphanumeric
- No name conflicts
- Escaping with underscore
- Embedding type information



The validation question

How to validate the code-gen?

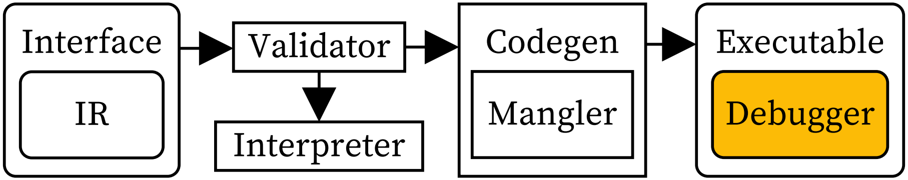
- **Not** with the validator
- With the interpreter
- Slower and simpler
- Compare results of interpreter and codegen



The debug question

How to validate the test programs?

- Interactive debugger
- Embedded in executable
- Program view with breakpoints
- Watch window



The correctness & .NET requirement

- Wrote test programs
- Tested every instruction
- Compared with interpreter
- Validated with debugger

The multiplatform requirement

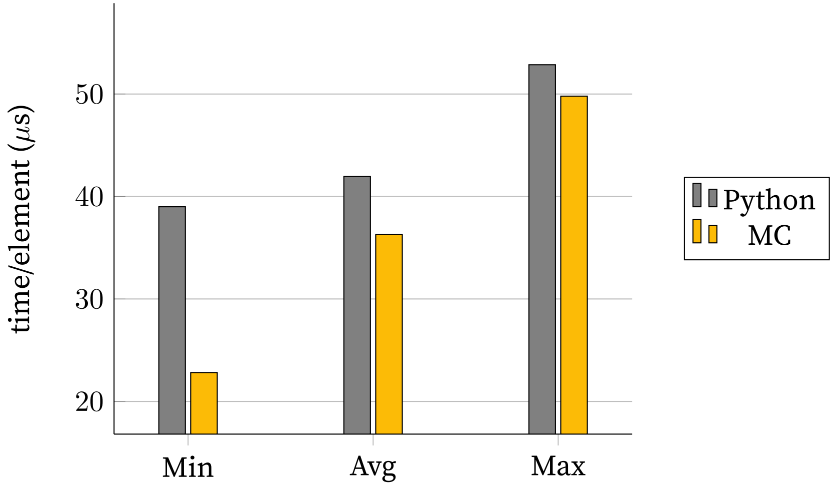
Microsoft .NET Compiler for windows
Mono everywhere else

- Linux
- Mac OS X, iOS, tvOS, watchOS
- Sun Solaris
- BSD - OpenBSD, FreeBSD, NetBSD
- Microsoft Windows
- Nintendo Wii
- Sony PlayStation 3
- Sony PlayStation 4

The performance requirement

- Benchmark
- 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
- Documented in thesis
- Helps the research team
- Video game industry

Demo time!

Defence