Synchronous Programming with Refinement Types
Honors Capstone – Fall 2023

José Luiz Vargas de Mendonça
April 23\textsuperscript{rd}, 2023
Nice to meet you!

• I’m José Vargas
• Hometown: Manaus, Brazil
• Major: Aerospace 🚀 and Computer Engineering 📱
• Hobbies: Learning languages (spoken and programming)
Table of Contents

- Research
  1. Background
  2. Motivation
  3. Zelus Compiler Architecture
  4. Code Example
  5. Future Steps

- Honors Experience
  6. Starting the Project
  7. Focus Area Courses
  8. Alternative Paths
Research
1. Background

- **Cyber-physical systems (CPSs)** are composed of software that interacts with the environment.
- Unit testing complex software might not cover all scenarios.
- Formal verification provides rigorous tools to prove **software safety**.
- **Refinement types** can add constraints to base types on programming languages.

\[
\begin{align*}
\text{let } (x : \text{int}) &= 4 \\
\text{vs} \\
\text{let } (x : \{v : \text{int} | v \geq 0\}) &= 4
\end{align*}
\]
2. Motivation

- **Synchronous languages** are used to program **Cyber-Physical Systems** (CPSs)

- **Refinement types** are used for formal verification
2. Motivation

- Unit tests are **not enough** to show that software is safe!
- **MARVeLus** tells you what you need to know to check that a critical software follows its **specifications**.
3. Zelus Compiler Architecture

Figure 3: Compiler architecture

3.1 Introducing MARVeLus: a refinement type verifier for Zelus
3.2 MARVeLus workflow
4. Refinement variable declaration

- Refinement type requires a base type and a type predicate
- Once the variable is declared, rhs expression is checked against predicate
- Predicate can depend on previously defined variables

\[
\text{let } \var : \{ \nu : t \mid \phi(\nu) \} = e
\]
4.1 Refinement variable check

Verification condition: \( \neg (\bigwedge^E \phi_i \rightarrow \phi(var)) \)

where

\( E \), is the program environment

\( \phi_i \), are constraints associated with previously defined variables

\( \phi(var) \), is the constraint we want to satisfy
4.2 Refinement variable declaration example

let pi = 3.14159
let w = 2 .* pi
let y0 : {v : float | v >= pi} = 4.0

Verification condition:
\[ \neg((pi = 3.14159) \land (w = 2pi) \land (y0 = 4.0) \rightarrow (y0 \geq pi)) \]

- The constraints are added to the program environment as variables are declared
- Once verification condition is checked to be true, \( \phi(var) \) is added to the environment
4.2 Refinement variable declaration example (continued)

```plaintext
let pi = 3.14159
let w = 2 .* pi
let y0 : {v : float | v >= pi} = 4.0
let y1 : {v : float | v >= y0 * 2} = 10.0
```

If constraint is satisfiable, display passed message

```
Proving constraint: (let ((a11 (=> (and (= y0 10.0) (= y0 pi)) (= y0 4.0)) (= w (* 2.0 pi)) (= pl (/ 3.14159.0 100000.0))) (y1 (* y0 2.0))))
```

If constraint is not satisfiable, display a warning and provide counter-examples

```
Proving constraint: (let ((a11 (=> (and (= y0 (- 4.0)) (= w (* 2.0 pi)) (= y0 (* pi 3.14159 0 100000.0))) (= y0 pi)))
```

Counter-example

```
Counterexample found:
  w: 6.28318
  pi: 3.14159
  y0: -4
Could not prove: (=> y0 pi)
```
5. Current work and next steps

- Preliminary work published at FTSCS 2022
  

- Include support for the continuous part of CPSs
- Refactor compiler code for maintainability
Honors Experience
6. Starting the Project

AERO 495: Fundamentals of Aerospace Computing
Fall 2020
(Currently AERO 350)

Part I: Fundamentals of Computer Science

Part II: Fundamentals of Computational Science

Part III: Introduction to Embedded Systems
7. Focus Area Courses

- **EECS 483 – Compiler Construction** (FA22, Prof. Max New)
  - Learned how to convert a text file into assembly code
  - Runtime definitions
  - Data structures encoding

- **EECS 590 – Advanced Programming Languages** (FA22, Prof. Jean-Baptiste Jeannin)
  - Formal introduction to PL theory
  - Proofs by induction
  - Operational semantics and typing rules

- **AERO 490 – Directed study** (FA21, Prof. Jean-Baptiste Jeannin)
  - Develop the code base for the MARVeLus Compiler

\[\begin{align*}
0 & := \lambda f.\lambda x.x \\
1 & := \lambda f.\lambda x.f\ x \\
2 & := \lambda f.\lambda x.f\ (f\ x)
\end{align*}\]
8. Alternative Paths

- Explored CVC5 SMT Solver instead of Z3
- Start a new programming language from scratch
9. Acknowledgements

• **My family** for continuous support that allowed me to complete undergraduate education in the US

• **Professor Jean-Baptiste Jeannin** for the continuous support and mentoring over the past 2.5 years

• **Jiawei Chen** for mentoring and time dedicated to answer questions

• The **MARVL research group** for support with presentations, articles and class scheduling

• **Honors Program** for academic support. Shoutout to **Rachel Armstrong** for helping me with scholarship applications!

• The **Aerospace and EECS Department** for all the challenging classes and projects

• **Dr. John Callewaert** for academic support since freshman year