Agda2Rust: A Study on an Alternative Backend for the Agda Compiler

Hector Peeters - h.peeters@student.tudelft.nl

Supervisors: J.G.H. Cockx
L.F.B. Escot

1. Agda

- Dependently typed functional programming language [1]
- Proof assistant
- Compiles to a different language
- Lazily evaluated
- Backends for Haskell and JavaScript

2. Rust

- General purpose systems language [2]
- Statically typed
- No garbage collection
- Compiled using an LLVM backend
- Collection of modern libraries

3. Motivation

- Support for FFI with more languages
- Increase Agda's industry adoption
- Haskell backend uses a lot of unsafe type casts
- Potential performance improvement
- Make use of Rust's library ecosystem

4. Implementation

```
1  data Nat : Set where
2      zero : Nat
3      suc  : Nat → Nat
4
5  plus : Nat → Nat → Nat
6  plus zero    n = n
7  plus (suc m) n = suc (plus m n)
```

Figure 1: Natural number and plus definition in Agda

```
1  #[derive(Debug, Clone)]
2  enum Nat {
3      zero(),
4      suc(Box<Nat>),
5  }
6
7  fn zero() -> Nat {
8      Nat::zero()
9  }
10
11 type suc0 = impl FnOnce(Nat) -> Nat;
12 fn suc() -> suc0 {
13     move |a| Nat::suc(Box::new(a))
14 }
15
16 type plus0 = impl FnOnce(Nat) -> Nat;
17 type plus1 = impl FnOnce(Nat) -> plus0;
18 fn plus() -> plus1 {
19     move |a| move |b| match a {
20         Nat::zero() => b,
21         Nat::suc(c) => suc()(plus()(*c)(b)),
22     }
23 }
```

Figure 2: Generated strict Rust code of the natural number example

5. Limitations

- No higher-order functions
- Laziness implementation is incomplete
- Incorrect code generation for non-erased dependent types

6. Results

- Does not outperform the current Haskell backend
- Lazy evaluation adds significant performance overhead
- Rust and Rust-optimal are identical in performance so for strict evaluation: agda2rust generates almost optimal code

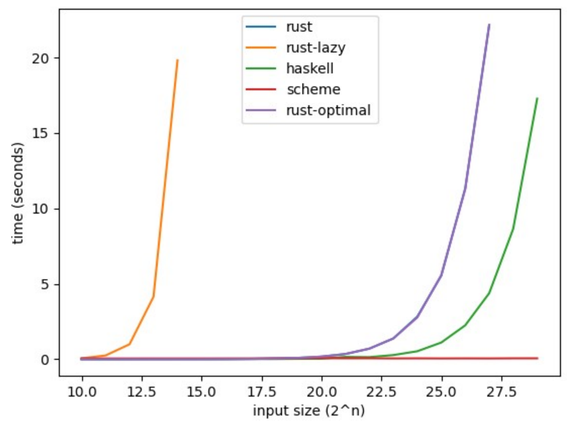

Figure 3: Consume function benchmark

7. Conclusion

- Rust is not a suitable target language when performance is important
- Rust's type system makes code generation unnecessarily difficult
- The current generated code integrates very nicely into an existing Rust code base
- A redesign is required to make the backend feature-complete

8. Future Work

- Implement a more optimised thunk
- Add support for more built-in functions and data types
- Implement additional optimisations (removing erased types, built-in booleans, tail-call optimisation, ...)
- Redesign data type representation to allow full dependent type support
- Consider other systems languages like Zig or Nim

References:
[1]: The Agda Team. 2022. The Agda Documentation. https://agda.readthedocs.io/en/v2.6.2.1/
[2]: The Rust Team. 2022. Rust Programming Language Website. https://www.rust-lang.org/

TU Delft