Code Extraction From a Dependently Typed Language - "How Effectively Does Forth Serve as a Target Language for Extraction From Agda?"

Louis Milliken – Lmilliken@student.tudelft.nl
Supervised by Jesper Cockx, Lucas Escot

1. What Is Agda?

- Pure functional language
- Dependently typed
  - A dependent type are type whose definition depends on another value.
  - E.g. (*Vec 5*)could represent an array of length 5

```
data Vec (A : Set) : Nat → Set where
  []    : Vec A zero
  _::_  : {n : Nat} → A →
          Vec A n → Vec A (suc n)
```

  - Used to provide guarantees about code and write proofs
- Lazy evaluation
- Some code is unused after compilation

2. What Is Forth?

- Stack based, imperative language
- Untyped
- Values are pushed onto the stack instead of being passed as arguments
- Words (functions) pop values off the stack to use them

```
5 6  ok
.s <2> 5 6  ok
+  ok
.s <1> 11  ok
```

- Strict evaluation

3. The Problem

- How to translate Agda to Forth?
- How will it perform compared to other compilers? (Haskell, Scheme)
- Is it even possible to implement every feature of Agda in Forth
  - If not, how much can be?

4. Results

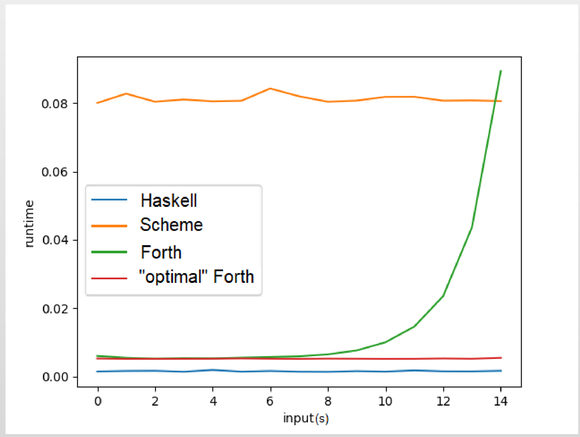

Fig. 1: Runtimes for consume function with inputs 0 - 15

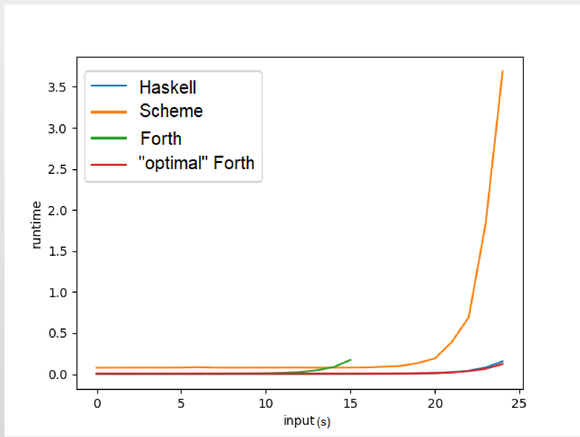

Fig. 2: Runtimes for consume function with inputs 0 - 24

5. Conclusions

- Forth is not a very good target language
- Lack support for too many features of Agda
- Improvements to my compiler could greatly improve performance
  - Improve thunk handling
  - Consider different Forth implementations

[1] Agda GHC Backend: https://agda.readthedocs.io/en/v2.6.2.1/tools/compilers.html#ghc-backend

6. Example Translations

```
consume : Nat → Nat
consume zero = zero
consume (suc n) = consume n
```

Fig. 3: consume function in Agda

```
defer consume
...
ENDFLAG :noname { a }
a 0 0 pointer ! obj=
if
    wildcards 0 pushArrayN { }
    0
else
    1 a sub pass pass { b } b consume pass
then
;
create XTconsume fillHere
:noname XTconsume foldThunks ; is consume
```

Fig. 4: consume function in Forth

```
ite : {A : Set} → Bool → A → A → A
ite true x y = x
ite false x y = y
```

Fig. 5: ite (if, then else) function in Agda

```
defer ite
...
ENDFLAG :noname { d c b a }
b if c else d then ;
:noname { c b a e }
c b a e pass pass pass ;
:noname { b a f }
b a f pass pass ;
:noname { a g }
a g pass ;
create XTite fillHere
:noname XTite foldThunks ; is ite
```

Fig. 6: ite (if, then else) function in Forth