Extracting LLVM IR from Agda

Jochem Broekhoff June 27, 2022 Project Group 7

1. *AGDA* IN SHORT

- Functional programming language
- Dependently typed (Π, Σ)
- *Total:* - finite-time termination
  - extensive matching

2. MOTIVATION

- Broader adoption of proven correctness
- Leverage LLVM's optimization passes
- Link with popular system libraries

3. IMPLEMENTATION

- Pipeline demonstrated in figure 1
- Boehm GC as memory manager
- Small supporting runtime library
- No foreign-function interface, so no linking with system libraries

LAZINESS vs. STRICTNESS

- Strict: *everything* computed directly
- Lazy: values *only* computed when otherwise the program cannot be executed further
- Trickier at runtime: using thunks (see right)
- Making lazy stuff strict is easy, not vice-versa

4. BENCHMARK ANALYSIS

Three aspects:
- LLVM vs. MAlonzo vs. Scheme
- Lazy vs. strict evaluation in LLVM
- Optimization: static memory allocation

Three test cases: consuming a large number, executing quicksort, and summing all triples of natural number coordinates in a 3D sphere.

Selection of each combination is plotted in figure 3 (bottom right).

5. CONCLUSIONS

- Basic code extraction is relatively simple
- No improved asymptotic performance
- Straightforward strict evalution is no good.
- Simple optimizations can have a significant effect on the running time
- More indirect translation paths seem promising, via dedicated frameworks

So: is LLVM a *practical* backend?
Yes, but it definitely needs some work.
Other approaches might be better.

THUNKS: *Delay & Force*

Agda2LLVM's *thunks* are visualized in figure 2. This shows a visual representation of what it means to *force* a thunk:

1) A thunk starts off being a box with only a recipe for the value that it represents. This is the **delayed** state.

2) The program passess references to the thunk as pointers, *without* looking into the box.

3) **Forcing:** As soon as *somewhere* the value is needed, the recipe is executed and the result is **replaced in the same box**, such that the result is immediately usable *program-wide*.

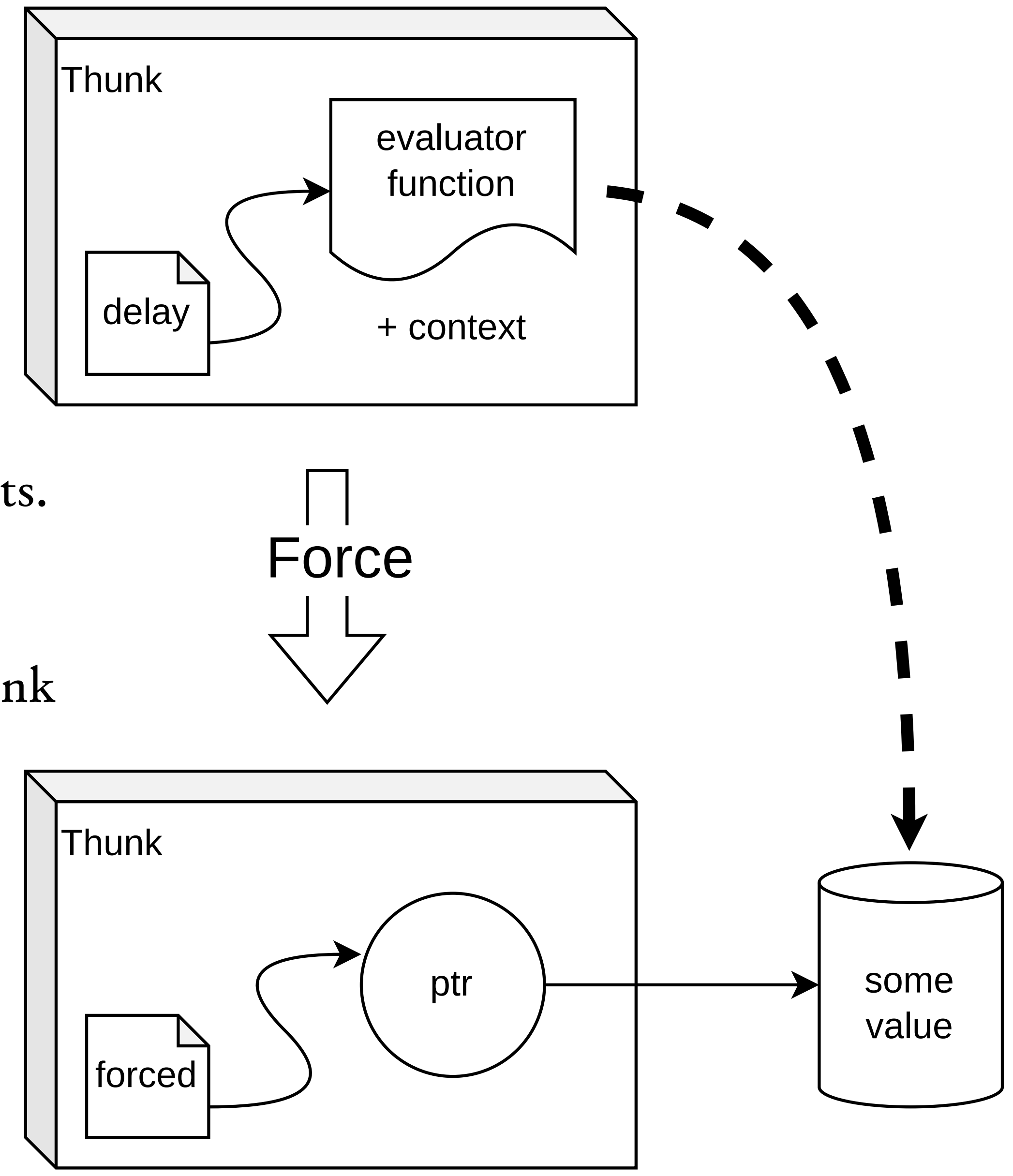

Figure 2: simplified inner workings of thunks and their lazy evaluation.

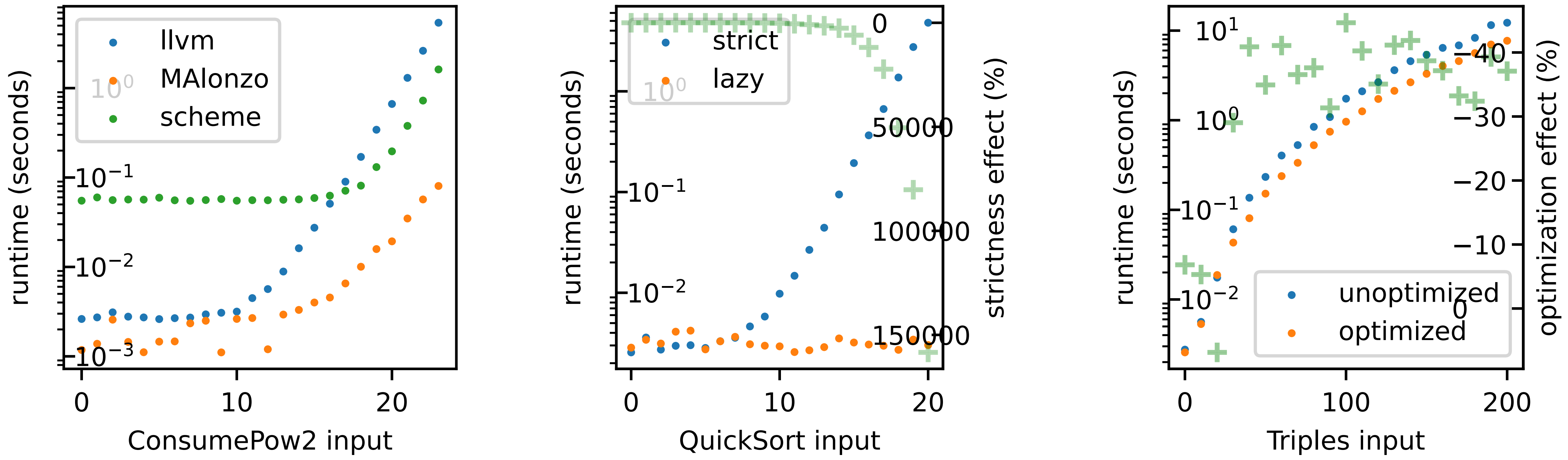

Figure 3: a sample of the benchmark results.

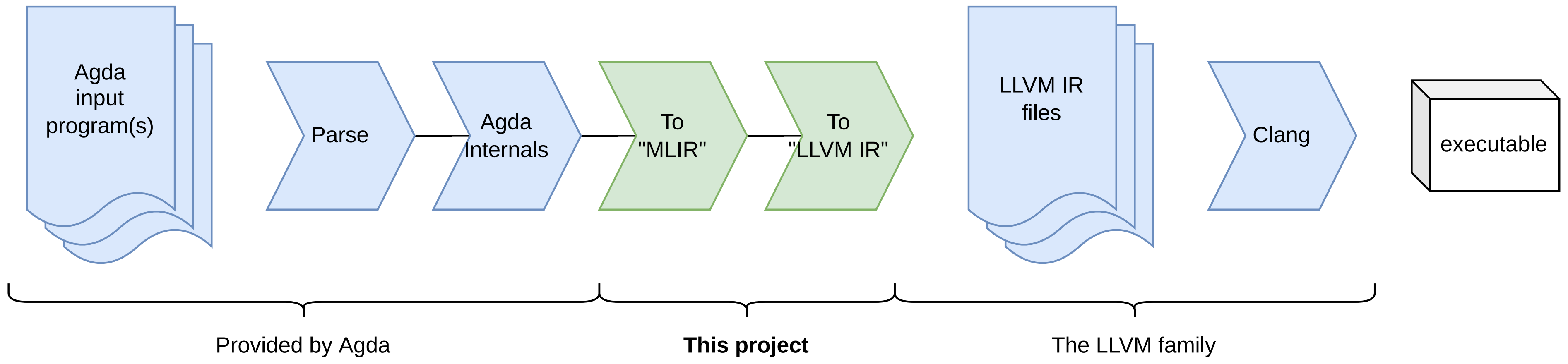

Figure 1: summarized pipeline of the Agda2LLVM implementation.

Supervisors: Jesper Cockx
Lucas Escot

CSE3000 - Research Project
BSc. CSE - EEMCS

TUDelft