Deriving a Strongly Normalizing STG Machine

Dirk Kleeblatt
Technische Universität Berlin, Fakultät IV, Sekr. TEL 12-2
Ernst-Reuter-Platz 7, D-10587 Berlin
klee@cs.tu-berlin.de

Traditionally, compiled systems compute so called *weak head normal forms*, i.e. they perform no computations beneath λ-abstractions or case analyses. But for example in dependent type checking, *strong* normal forms are required to test for β-convertibility. This requires to deal with free variables since e. g. formal parameters of abstractions occur free in the body of the abstraction, and compiled code usually cannot deal with this free variables. Using an interpreter solves this problem, but has several disadvantages:

- Interpreted code has reduced performance compared to compiled code.
- When writing a compiler, an additional interpreter is needed just for type checking, so, two different parts of the code base have to be maintained.
- Differences between interpreter and compiler may result in computations having a different result at runtime than during type checking. This may violate type safety.

We present a strong normalization system for FUN, a simple lazy functional language, that overcomes this disadvantages by using compiled code for normalization and is well suited for dependent type checking. The generated code is efficient enough to be directly used as the final compiler output if type checking is successful. We start with a big step operational semantics, and transform it in several steps to an abstract machine executing pseudo-assembler code. The present work is a formalization of the implementation of Ulysses, a dependently typed lazy functional language informally introduced in [Kle08].

We extend the derivation of a compiled system from [dlEP03] to strong normalization. Therefore, following the strict implementation from [GL02], we extend a weak evaluator with *accumulators* representing irreducible expressions, and a read back phase.

[dlEP03] Alberto de la Encina and Ricardo Peña. Formally deriving an STG machine. In *PPDP '03: Proceedings of the 5th ACM SIGPLAN international conference on principles and practice of declaritive programming*, pages 102–112. ACM Press, 2003.

[GL02] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In *ICFP '02: Proceedings of the seventh ACM SIGPLAN international conference on Functional programming*, pages 235–246. ACM Press, 2002.

[Kle08] Dirk Kleeblatt. Checking dependent types using compiled code. In *Implementation and Application of Functional Languages, 19th International Workshop, IFL 2007. Revised Selected Papers*, volume 5083 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 2008.