Yin-Yang: Concealing the Deep Embedding of DSLs

Vojin Jovanovic and Amir Shaikhha

Ecole Polythechnique Federale de Lausanne, EPFL
{first}.{last}@epfl.ch

Abstract. Deeply embedded domain-specific languages (EDSLs) inherently compromise programmer experience for improved program performance. Shallow EDSLs complement them by trading program performance for good programmer experience. We present Yin-Yang, a framework for DSL embedding that uses Scala reflection to reliably translate shallow EDSL programs to the corresponding deep EDSL programs. The translation allows program prototyping and debugging in the user friendly shallow embedding, while the corresponding deep embedding is used in production—where performance is important. The reliability of the translation completely conceals the deep embedding from the DSL user. For the DSL author, Yin-Yang generates the deep DSL embeddings from their shallow counterparts by reusing the core translation. This obviates the need for code duplication and assures that the implementations of the two embeddings are always synchronised.

Keywords: Embedded Domain-Specific Languages, Macros, Deep Embedding, Shallow Embedding, Compile-Time Meta-Programming

1 Introduction

External Domain-specific languages (DSLs) are languages with a custom compiler specialized to a particular application domain. Knowledge about the domain and language specialization give the compiler possibilities for optimization that do not exist in general purpose languages. The restricted language makes it easy for *DSL users* to learn the language while the optimizations can yield execution times close to hand-optimized programs [2].

Development of external DSLs includes building a parser and type checker as well as a large tool-chain (i.e., debuggers, IDEs, and documentation tools). An appealing alternative are *embedded DSLs* (EDSLs) [1] that reuse the parser, type checker and the tool-chain of the general-purpose *host language* to minimize required development effort. We can classify DSL embeddings into two categories:

– *Shallow embeddings* where values in the embedded language are *directly* represented by the values in the host language. A special sub-category of shallow embeddings are *direct embeddings* where language constructs and term types are exactly the same as in the host language.

– *Deep embeddings* where values in the embedded language are symbolically (with data structures) represented in the host language.

Direct embeddings are typically friendly for the *DSL user* as they *i) linguistically match* the host language and *ii)* can be easily debugged since values of the embedded language directly correspond to the values of the host language. However, since programs can not be introspected, the number of possible domain-specific optimizations is limited.

Deep embeddings reify the DSL programs into an *intermediate representation* IR. The reification hinders usability as it often relies on complex type system constructs that create a linguistic mismatch with the host language. Furthermore, the IR construction prevents term introspection during debugging and makes the problem even harder. However, the domain knowledge about the IR opens opportunities for optimizations.

2 Demonstration: A DSL for In-Memory Queries

Our demonstration shows how Yin-Yang is used for: *i)* the DSL author and *ii)* the DSL user. For the purpose of the demonstration we use an existing DSL for writing in-memory queries. The demonstration shows how the DSL user can be completely unaware of the concept of the deep embedding. And how the DSL author can program the deep embeddings and program transformations as regular Scala code.

The DSL author writes a naïve Scala implementation of common query operators.

The DSL user can immediately write concise queries in the DSL. Although the query performance is satisfactory for prototyping, in production, the overhead is unacceptably large.

To improve on the situation the DSL author decides to provide a deep embedding for the query engine. All he needs to do is place a single Scala annotation per query operator he implemented. The annotation marks that the for the operator the author would like to have a deep embedding.

All the DSL user needs to do now is put the queries in a DSL scope. After putting queries into the DSL scopes the user is still faced with the same type errors as before. The program execution errors can still be debugged in a standard debugger. After placing queries into scopes all standard compiler optimizations are applied to the DSL yielding significant performance improvements.

The DSL author then decides to further improve performance by introducing the domain specific optimizations in the query language. The DSL author declares simple rewrite rules that the DSL compiler will later apply.

The DSL author then measures performance of his DSL and realizes that the generated code performs on par with the hand-optimized version of the query.

References

1. Paul Hudak. Building domain-specific embedded languages. *ACM Comput. Surv.*, 28(4es):196, 1996.
2. Tiark Rompf, Arvind K. Sujeeth, Nada Amin, Kevin J. Brown, Vojin Jovanovic, HyoukJoong Lee, Manohar Jonnalagedda, Kunle Olukotun, and Martin Odersky. Optimizing data structures in high-level programs: New directions for extensible compilers based on staging. In *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, page 497–510, 2013.