Type providers for Rust

December 2016

candidate	Araz Abishov
supervisors	Martin Steffen and Volker Stolz
group	PMA
type	60 ECTS
study program	computer science
planned date of completion	May 2018

Short description

The task is to design and implement a advanced type-based concepts for Rust, especially type providers which have proven a useful concept for diminishing the amount of boilerplate code when working with APIs.

Background and motivation

Rust [1] is a quite recent programming language, with a first stable release 2015, which is gaining wide attraction. It has been embraced by the open-source community; for example Mozilla's parallel prowser engine project *Servo* [2], a modern, high-performance browser engine designed for both application and embedded use, is implemented in Rust.

The stated goal of the language is to enable fast, efficient, and memory safe *systems programming*, stressing safe code even for low-level, hardware-close applications and in the presence of *concurrency*. It combines a number of advanced and/or novel features, for example sophisticated memory management, but *not* based on carbage collection but based on ownership, a type system based on traits, and integration with modern package management.

Problem setting

Type providers, most prominently known from F#, are a type-based abstraction which provides type safe integration of external resources. The Rust language currently does not support type providers. In the thesis, we will extend the compiler with the knowledge about such external data sources: given a data definition in the form of a schema, the typing environment of a given Rust program is extended by those externally defined types. Furthermore, expressions of such types are made known to the compiler infrastructure. During compilation, for data accesses to such types (that is, reads and writes), corresponding wrappers for deserialization/serialization are generated. The developed framework should be well-documented and easy to extend, especially with respect to different external type sources (SQL schema, XML schema, JSON,...), and the actual data sources (SQL query, XML files, ...). Software development should be done in accordance with best practices in software development, e.g. through accompanying unit tests.

This approach would be validated by comparing a hand-written case study in Rust without the extension against a version using the new language extension, and possibly against a solution in a programming language which already offers similar features (e.g. F#, above).

Keywords: type providers. compiler, code generation, compiler plugin, Rust

References

[1] Rust programming language. `https://www.rust-lang.org/`, Dec. 2016.

[2] Servo, the parallel browser engine project. `https://servo.org/`, Dec. 2016.