MSc Thesis: A Type System for Ensuring Safe Structured Concurrency in Scala

Student: Sebastian Willenbrink (stwi@kth.se)
Supervisor: Associate Professor Philipp Haller, EECS/TCS

Background

Concurrent programming is known to be difficult due to various hazards including data races, deadlocks, etc. In the area of programming languages, a major focus has been to design programming languages and systems that ensure the safety of concurrent/parallel code without affecting performance negatively or restricting the flexibility of programmers. A successful example of a recent language that ensures data-race freedom at compile time is Rust. Rust's approach relies on a sophisticated type system based on ownership and borrowing, concepts that have been studied during the last 25 years.

The high-level goal of this thesis project is to bring some of the ideas behind Rust and related languages to a language that focuses more on functional programming, in this case Scala. The proposed approach is based on extending Scala's type system with a notion of concurrent regions/zones which in addition to providing concurrency also restrict variable capture and references in order to ensure data-race freedom.

Recently, the type system of Scala 3 has been extended with capture checking [1], a system that statically tracks so-called capabilities, and how they are used, as part of type checking. One of the main use cases for capabilities is to improve the safety of programming with resources and effects. The reference documentation for a current prototype implementation [2] shows several examples, including a try-with-resources pattern. More generally, capabilities address the long-standing problem of effect polymorphism [1,3].

Research Question

The research question can be split into multiple sub-questions:

> Can the existing type system of Scala be extended to support safe structured concurrency where data races are prevented at compile time? Is it possible to establish important properties including type preservation and progress? How significant is the annotation overhead for developers? What is the impact on runtime performance of such an extension?

Hypothesis

The hypothesis the project tries to evaluate is the following:

> It is possible to extend Scala's existing type system with safe structured concurrency in a way that ensures data-race freedom. It is possible to leverage existing concepts, such as scoped capabilities, such that the resulting annotation overhead remains low. Due to

type erasure and few if any runtime checks, the impact on runtime performance is expected to be low.

Research Method

1. Extend the recent work on Safe Zones [4] to enable concurrent regions owned by a single async task.
2. Devise a type system extending Scoped Capabilities [1] such that concurrent regions ensure data-race safety by preventing access to potentially shared mutable state.
3. Formalize the main ideas of the type system in the context of a small-step operational semantics for a small object-oriented core language.
4. Prove important properties like progress and preservation for this small core language. Depending on the time available, the proof might be a pen-and-paper proof or a machine-checked proof.
5. Implement the extension of the type system for Scala as a prototype. This step can be skipped if there is no time or significant hurdles are encountered.
6. Demonstrate the flexibility and annotation overhead of the type system on a range of task-parallel programs written using the proposed concurrent regions. If a functional prototype is not possible, this demonstration will be based on theoretical examples.

Background of the Student:

I have taken multiple courses in the area of formal methods and programming languages, with the two most relevant being a course on programming language semantics [5] and a seminar on type systems [6]. Through these I learned how type systems and semantics are formalized and familiarized myself with concepts like simply typed lambda calculus and dependent types.

Suggested Examiner at KTH:

Suggested Supervisor at KTH:

Associate Professor Philipp Haller
I have been in contact with Philipp Haller and he has expressed interest in serving as supervisor.

Resources:

Philipp Haller is familiar with the topic. Besides that, I am not aware of any significant resources existing at the department.

Eligibility:

I have completed more than 60 ECTS on an advanced level. All the courses relevant to the thesis were completed at the TUM. I've completed the course on theory of knowledge and research methodology (DA2210).

Study Planning:

I have received grades on all courses except DH2323 as I have not yet finished the practical project of the course. Most of the work is done and only the report is missing.

References:

[1]: Martin Odersky, Aleksander Boruch-Gruszecki, Edward Lee, Jonathan Immanuel Brachthäuser, Ondrej Lhoták:
Scoped Capabilities for Polymorphic Effects. CoRR abs/2207.03402 (2022)
https://doi.org/10.48550/arXiv.2207.03402

[2]: https://docs.scala-lang.org/scala3/reference/experimental/cc.html

[3]: Lukas Rytz, Martin Odersky, Philipp Haller:
Lightweight Polymorphic Effects. ECOOP 2012: 258-282
https://doi.org/10.1007/978-3-642-31057-7_13

[4]:
https://github.com/lampepfl/dotty/pull/16517
https://github.com/scala-native/scala-native/pull/3120

[5]: Semantics course at TUM
https://campus.tum.de/tumonline/ee/ui/ca2/app/desktop/#/slc.cm.reg/student/modules/detail/light/456513/study-year/1615

[6]: Tyranny of the Types: Curse or Blessing seminar at TUM
https://www.cs.cit.tum.de/pl/lehre/sommersemester-22/seminare/the-tyranny-of-types-curse-or-blessing/