Karlsruher Institut für Technologie

Institute of Information Security and Dependability (KASTEL)
Application-oriented Formal Verification
Prof. Dr. Bernhard Beckert

Praxis der Forschung

Type rule synthesis with generative AI

Background.
Pluggable type systems are a lightweight alternernative to more complex specification approaches. At our chair, we have developed *property types* for Java [1]. A property type consists of a Java type and an annotation associated with a boolean property. In the example on the right, the annotation `Interval` has the property `min <= subject <= max`.

```
@Interval(min=1, max=3) int f;
public void foo() {
    f = 3; // OK
    @Interval(min=2, max=4)
    int l = f + 1; // Error!
}
```

But defining types and type rules for all use cases is time-intensive. In our example, there is a type error on the second line even though the program is correct: Since the programmer has not manually defined a type rule that says that adding 1 to a value of type $[1, 3]$ yields a value of type $[2, 4]$, the type checker rejects this assignment as being incorrect.

While there are several approaches that use AI to infer the placement of type annotations [2] or other kinds of specifications [3,4] in the program, we want the AI to infer these kinds of type rules, along with a proof of their correctness.

Bibliography.

1. Lanzinger, Bachmeier, Ulbrich, Dietl (2023). Scalable and Precise Refinement Types for Imperative Languages. iFM 2023.
2. Jha Dietl (2024). OppropBERL: A GNN and BERT-Style Reinforcement Learning-Based Type Inference. SANER 2024.
3. Si. (2020). Code2Inv: A Deep Learning Framework for Program Verification. Computer Aided Verification 2020.
4. Wu, Barrett, Narodytska (2023). Lemur: Integrating Large Language Models in Automated Program Verification. ArXiv, abs/2310.04870.

Contact
Florian Lanzinger lanzinger@kit.edu Office 50.34R227