Diss. ETH No. 19851

# RUMER: A PROGRAMMING LANGUAGE AND MODULAR VERIFICATION TECHNIQUE BASED ON RELATIONSHIPS

A dissertation submitted to
**ETH Zurich**

for the degree of
**Doctor of Sciences**

presented by
**Stephanie Balzer**

accepted on the recommendation of
**Prof. Dr. Thomas R. Gross**, examiner
**Prof. Dr. Sophia Drossopoulou**, co-examiner
**Prof. Dr. Peter Müller**, co-examiner
**Dr. Alexander J. Summers**, co-examiner

2011

# Abstract

The idea of asserting a program's correctness by a mathematical proof rather than exhaustive testing has persisted in software system development. The notion of an invariant to capture the constant properties of data plays a central role in such program verification attempts. Invariants have also infiltrated object-oriented programming languages and have become a foundation for verifying object-oriented programs.

Current invariant-based, object-oriented verification techniques provide modular verification of single-object invariants. However, as soon as an invariant relates several objects, modular reasoning about the invariant is not possible without imposing certain restrictions on the program. Current solutions to the verification of such multi-objects invariants either restrict aliasing in a program or expand the visibility of an invariant beyond the boundary of the declaring class.

This thesis represents a two-part approach to program verification based on invariants. We develop a programming language, Rumer, that provides higher-level abstractions to capture not only objects but also the relationships between them. In addition, we elaborate a visible-state verification technique for Rumer that facilitates the modular verification of Rumer programs.

The Rumer programming language employs the abstraction of a relationship as a key architectural element in building a software system. Rather than employing references in objects to represent the relationships between objects, those relationships are represented explicitly by a relationship instance. This indirection gives rise to a stratified programming model. Based on this model, we formulate the Matryoshka Principle, a modularization discipline that provides strong encapsulation for Rumer types.

The verification technique is based on the Matryoshka Principle and leverages the particular features of the Rumer programming and specification language. The Matryoshka Principle provides modular reasoning about state changes and permits the adoption of a visible-state semantics for invariants. The Rumer programming and specification language encompasses techniques to encapsulate multi-object invariants in relationships and includes a rich assertion language that is based on discrete mathematics.

# Zusammenfassung

Die Idee, die Korrektheit eines Programms durch einen mathematischen Beweis anstatt durch ausgiebiges Testen sicherzustellen, hat seit jeher eine zenrale Bedeutung in der Entwicklung von Softwaresystemen eingenommen. Verifikationsansätze basieren traditionsgemäss auf Invarianten, die jene Eigenschaften von Daten spezifizieren, die zur Laufzeit des Programms konstant sind. Invarianten finden mittlerweile auch in objektorientierten Programmiersprachen Anwendung und haben sich zu einer wichtigen Grundlage für die Verifikation objektorientierter Programme entwickelt.

Gängige invariantenbasierte, objektorientierte Verifikationstechniken erlauben die modulare Verifikation von Invarianten, die sich auf ein einzelnes Objekt beziehen. Setzt eine Invariante jedoch mehrere Objekte zueinander in Beziehung, ist modulares Schliessen nur mittels Einschränkungen des Programms möglich. Gängige Lösungsansätze für die Verifikation solcher Multi-Objekt-Invarianten unterbinden entweder das Aliasing (d.h. die Existenz unterschiedlicher Referenzen auf denselben Speicherplatz) oder weiten die Sichtbarkeit einer Invariante über die Grenzen der deklarierenden Klasse hinaus aus.

Die vorliegende Dissertation präsentiert einen zweistufigen Ansatz für die Verifikation invariantenbasierter Programme. Wir beschreiben die Programmiersprache Rumer, welche Abstraktionen für die Repräsentation von Objekten und deren Beziehungen zueinander zur Verfügung stellt. Des Weiteren erarbeiten wir eine Verifikationstechnik für Rumer. Diese basiert auf sichtbaren Zuständen und ermöglicht die modulare Verifikation von Rumer Programmen.

Die Programmiersprache Rumer verwendet die Abstraktion einer Beziehung als grundlegendes Sprachkonstrukt zur Bildung eines Softwaresystems. Dabei werden die Beziehungen zwischen Objekten durch explizite Beziehungsinstanzen repräsentiert und nicht durch Referenzen in den beteiligten Objekten. Die Verwendung expliziter Beziehungsinstanzen zusätzlich zu den beteiligten Objekten führt zu einem mehrschichtigen Programmiermodell. Ausgehend von diesem Programmiermodell formulieren wir das Matrjoschka-Prinzip — ein Modularisierungsansatz, der starke Kapselungseigenschaften für Rumer Datentypen gewährleistet.

Die Verifikationstechnik basiert auf dem Matrjoschka-Prinzip und nutzt die spezifischen Eigenschaften der Programmiersprache Rumer und deren Spezifikationssprache aus. Das Matrjoschka-Prinzip ermöglicht das modulare Schliessen über Zustandswechsel und garantiert, dass Invarianten in sichtbaren Programmzuständen gelten. Die Programmierspra-

che Rumer und deren Spezifikationssprache bieten Techniken zur Kapselung von Multi-Objekt-Invarianten in Beziehungen und unterstützen eine reiche, auf diskreter Mathematik beruhende Aussagensprache.