ETH zürich

ETH Library

Enhancing separation logic for object-orientation

Doctoral Thesis

Author(s):
Staden, Stephanus J. van

Publication date:
2013

Permanent link:
https://doi.org/10.3929/ethz-a-009908413

Rights / license:
In Copyright - Non-Commercial Use Permitted

This page was generated automatically upon download from the ETH Zurich Research Collection.
For more information, please consult the Terms of use.

Diss. ETH No. 21293

Enhancing Separation Logic for Object-Orientation

A dissertation submitted to
ETH ZURICH

for the degree of
Doctor of Sciences

presented by
Stephanus Johannes van Staden
Master of Science, International University in Germany

born
February 22nd, 1983

citizen of
South Africa

accepted on the recommendation of

Prof. Dr. Bertrand Meyer, examiner
Prof. Dr. Peter O'Hearn, co-examiner
Dr. Matthew Parkinson, co-examiner
Dr. Cristiano Calcagno, co-examiner

2013

Abstract

Reasoning about programs can be tricky and error-prone. Formal verification facilitates the formulation and demonstration of rigorous arguments about program correctness. To be usable and useful, a program logic, i.e. a proof system for program verification, should accommodate the principal mechanisms of the programming language and crystallise the informal reasoning of programmers.

Recently, separation logic emerged as a promising tool for reasoning about shared mutable state, which pervades mainstream programming languages such as C, Java and Eiffel. Parkinson and others proposed a proof system for Object-Oriented (OO) programs that combines separation logic with mechanisms to reason about inheritance and dynamic dispatch. This system can verify a wide range of OO programs and design patterns in a concise way. The flexibility and simplicity of the system made it an attractive target for further improvement and broader application.

The contributions of this thesis address the following problems:

1. Specifying, verifying and using relationships between state abstractions. This is especially important when reasoning about OO programs that use multiple inheritance.

2. Reasoning about executable contracts. Executable contracts are often weak and may perform side-effects. Yet they capture useful design information, are programmer-friendly and assist in debugging.

3. Refinement and correctness by construction. Instead of first writing the code and then proving it correct, these techniques make it possible to write a correct program in the first place. The program and its correctness proof grow and evolve together.

State abstraction mechanisms, such as abstract predicates, are useful for reasoning about programs with modules that encapsulate state and hide information. Parkinson adapted abstract predicates to the OO setting, and

they play a central role in his proof system. Since OO code frequently relies on relationships between state abstractions of a class or a class hierarchy, this thesis enhances Parkinson's system with mechanisms for specifying, verifying and exploiting such relationships. The extension also makes it possible to establish the logical consistency of a class hierarchy without considering implementation details, and it facilitates reasoning about multiple inheritance.

Existing OO code often contains contracts in the form of executable preconditions, postconditions and class invariants. These specifications are typically weaker than separation logic assertions, but they are more lightweight and perhaps more likely to be written by programmers. Contracts also record valuable information about program design and are useful for testing and debugging. This thesis contributes a new technique for using the separation logic assertions to verify that executable contracts will always hold at runtime and that they will not perform unwanted side-effects. As a result, verified contracts need not be monitored at runtime, and they add confidence in the correctness of the code and the separation logic specification.

Correctness by construction is an important feature of a mature engineering discipline. In the context of software engineering, it is realised by a calculus for top-down program development that features refinement as a central technique. A refinement calculus helps to construct correct code from a given specification in a series of steps. This thesis proposes freefinement – an algorithm for obtaining a sound refinement calculus from a modular program logic. The resulting refinement calculus can interoperate closely with the program logic, and it is even possible to reuse and translate proofs between them.

Many aspects of the work also apply to other settings. None of the contributions rely on a specific flavour of separation logic. The work on multiple inheritance subsumes interface inheritance, and the reasoning techniques for executable contracts generalise to non-OO languages that use explicit memory management. Finally, freefinement applies to a great variety of formal systems, including program logics for other languages and type systems.

Zusammenfassung

Beweisführung über Programme kann knifflig und fehleranfällig sein. Formale Verifikation unterstützt das Formulieren und Beweisen von präzisen Behauptungen über die Korrektheit von Programmen. Um brauchbar und nützlich zu sein, sollte eine Programmlogik, d.h. ein Beweissystem für Programmverifikation, die Hauptmechanismen der Programmiersprache unterstützen und die informelle Beweisführung der Programmierer präzisieren.

In letzter Zeit hat sich Separation Logic als nützliches Werkzeug erwiesen, um Beweise über einen gemeinsam genutzten, veränderlichen Zustandsraum zu führen, der in populären Programmiersprachen wie z.B. C, Java und Eiffel allgegenwärtig ist. Parkinson et al. schlugen ein Beweissystem für objektorientierte (OO) Programme vor, das Separation Logic mit Mechanismen für die Beweisführung über Vererbung und dynamischer Bindung kombiniert. Dieses System kann ein breites Spektrum von OO-Programmen und Entwurfsmustern knapp und präzise verifizieren. Die Flexibilität und Einfachheit des Systems haben es zu einem attraktiven Zielobjekt für weitere Verbesserungen und Einsatzmöglichkeiten gemacht.

Die Beiträge dieser Dissertation befassen sich mit den folgenden Problemen:

1. Spezifikation, Verifikation und Verwendung von Beziehungen zwischen Zustandsabstraktionen. Dies ist speziell dann wichtig, wenn Beweise über OO-Programme geführt werden, die Mehrfachvererbung verwenden.

2. Beweisführung über ausführbare Spezifikationen. Ausführbare Spezifikationen sind oft schwach und können mit Nebeneffekten behaftet sein. Dennoch erfassen sie nützliche Designinformationen, sind programmiererfreundlich und helfen bei der Fehlersuche.

3. Verfeinerung und konstruktionsbedingte Korrektheit. Anstatt zuerst den Programmtext zu schreiben und dann zu verifizieren, erlauben diese Techniken es, das Programm von Anfang an korrekt zu schreiben. Das

Programm und der Beweis seiner Korrektheit wachsen und entwickeln sich gemeinsam.

Mechanismen für die Zustandsabstraktion, wie z.B. abstrakte Prädikate, sind nützlich für die Beweisführung über Programme mit Modulen, die Zustand und Informationen kapseln. Parkinson adaptierte abstrakte Prädikate an das OO-Umfeld und sie spielen eine zentrale Rolle in seinem Beweissystem. Da OO-Programme sich oft auf Beziehungen zwischen Zustandsabstraktionen einer Klasse oder Klassenhierarchie stützen, erweitert diese Dissertation Parkinsons System mit Mechanismen für die Spezifikation, Verifikation und Instrumentalisierung dieser Beziehungen. Die Erweiterung ermöglicht es auch, die logischen Konsistenz einer Klassenhierarchie zu begründen ohne Implementationsdetails zu berücksichtigen, und sie unterstützt die Beweisführung über Mehrfachvererbung.

Bereits existierende OO-Programme enthalten oft Spezifikationen in der Form von ausführbaren Vor-, Nachbedingungen und Klasseninvarianten. Diese Spezifikationen sind typischerweise schwächer als Zusicherungen in Separation Logic, aber sie sind auch schlanker und werden vielleicht eher von Programmierern geschrieben. Auch enthalten sie wertvolle Informationen über den Programmaufbau und sind nützlich für Tests und die Fehlerbehebung. Ein Beitrag dieser Dissertation ist eine neue Technik für die Verwendung von Zusicherungen in Separation Logic, die sicherstellt, dass ausführbare Spezifikationen zur Laufzeit immer eingehalten werden und keine ungewollten Nebeneffekte haben. Dadurch müssen ausführbare Spezifikationen zur Laufzeit nicht mehr überprüft werden und sie erhöhen das Vertrauen in die Korrektheit des Programms und der Spezifikationen in Separation Logic.

Konstruktionsbedingte Korrektheit (correctness-by-construction) ist eine wichtige Eigenschaft ausgereifter Ingenieursdisziplinen. Im Kontext von Software Engineering wird es durch einen Kalkül für Top-Down-Programmentwicklung realisiert, der Verfeinerung als zentrale Technik aufweist. Ein Verfeinerungskalkül hilft bei der Konstruktion korrekter Programme in einer Serie von Schritten, ausgehend von einer gegebenen Spezifikation. Diese Dissertation schlägt Freefinement vor – ein Algorithmus um einen korrekten Verfeinerungskalkül aus einer modularen Programmlogik zu erhalten. Der resultierende Verfeinerungskalkül ist in der Lage, mit der Programmlogik eng zu interagieren, und es ist sogar möglich, Beweise zwischen diesen beiden wiederzuverwenden und zu übersetzen.

Viele Aspekte dieser Arbeit sind auch auf andere Umgebungen anwendbar, keiner der Beiträge hängt von spezifischen Ausprägungen von Separation Logic ab. Die Arbeit an Mehrfachvererbungen umfasst auch Schnittstellenvererbung, und die Beweisführungstechnik für ausführbare Spezifikationen kann

auf Nicht-OO-Sprachen mit explizitem Speichermanagement verallgemeinert werden. Überdies betrifft Freefinement eine grosse Vielfalt von formalen Systemen, inklusive Programmlogiken für andere Sprachen und Typsysteme.