Diss. ETH No. 12057

Mechanizing Proofs of Program Properties

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Technical Sciences

presented by
MARTIN GITSELS

born 16/06/61
citizen of Federal Republic of Germany

accepted on the recommendation of
Prof. Dr. J. Gutknecht, examiner
Prof. Dr. B. Sanders, co-examiner

1998

Kurzfassung

Die Verwendung formaler Methoden, um die Korrektheit von Programmen zu beweisen, ist eine fest etablierte Disziplin in der heutigen Informatik. In diesem Zusammenhang wird das Attribut formal in der Regel für Methoden verwendet, die eine mathematische Grundlage haben und somit wohldefiniert sind. Im Hinblick auf Korrektheitsfragen im Zusammenhang mit der Entwicklung von Programmen umfassen formale Methoden den Vorgang der Spezifikation und des Beweisens ob ein Programm seine Spezifikation erfüllt. Der Vorteil, der in der Verwendung formaler Methoden liegt, ist jedoch häufig schwer ersichtlich und daher heftig umstritten. Ein Anwendungsgebiet jedoch, wo der Einsatz formale Methoden definitive Vorteile bringt, ist die Ausbildung in der Methodik des Programmierens an den Hochschulen. Ein beliebter Ansatz ist hier die Verwendung von Prädikatenlogik, wobei Zustände von imperativen Programmen durch Prädikate beschrieben werden. Beweise über die Eigenschaften von Programmen können dann durch reine Symbolmanipulation geführt werden. Konstruktive Beweise mit der Hand zu führen ist zwar schematisch aber häufig mühsam und fehlerbehaftet. Der Einsatz von Software-Werkzeugen kann hier die Arbeit erleichtern und vor allem Fehler vermeiden.

In der vorliegenden Arbeit wurde ein Beweis-Editor von Grund auf konzipiert und implementiert, welcher Beweisaktivitäten im Zusammenhang mit der Entwicklung imperativer Programme unterstützt. Im Gegensatz zu automatischen Systemen, werden jegliche Aktivitäten unter der Kontrolle des Benutzers ausgeführt. Das System bietet somit kein automatisches Beweisen. Wir sind der Meinung, dass dies eher ein Vorteil als ein Nachteil ist, da automatische Systeme häufig die Eleganz eines Beweises vernichten und wichtige Elemente der Logik wie Quantoren nicht zulassen. Das System bietet dem Benutzer eine grafische Oberfläche bestehend aus Komponenten wie Knöpfen oder Menüs und modernen Möglichkeiten für die interaktive Eingabe. Zusätzlich kann das System vom Benutzer erweitert werden. Die Verwendung eines Typinferenz Mechanismus erhöht das Vertrauen in Beweise ohne den Benutzer mit zuvielen Details zu konfrontieren.

Die Ergebnisse der Arbeit können wie folgt zusammengefasst werden. Wir haben einen neuartigen Ansatz für die Behandlung von quantifizierten Ausdrücken entwickelt und implementiert. Der Ansatz baut konsequent auf bekannte Ergebnisse aus der Welt des symbolischen Rechnens. Die Wahl von Oberon System 3 als Plattform für das System hat sich als grosser Vorteil erwiesen. Besonders das Gadgets System bot eine gute Auswahl mächtiger und zuverlässiger Werkzeuge zum Bau einer grafischen Benutzerschnittstelle. Das abstrakte Textmodell von Oberon ermöglichte uns eine elegante und einfache Lösung für die Integration textueller und struktureller Ediermöglichkeiten. Abschliessend müssen wir jedoch sagen, dass diese Art von Beweisern wohl kaum bei der seriösen Entwicklung von Software zum Einsatz kommt. Ein Einsatz in der akademischen Ausbildung ist jedoch sehr gut vorstellbar.

Abstract

Applying formal methods to reason about the correctness of programs is a well-established discipline in today's computer science. In this context, the attribute formal is typically used for those methods that have a mathematical foundation and thus are well-defined in mathematical terms. With respect to correctness issues in program development, a formal method usually comprises the tasks of specifying a program and ensuring that the program meets its specification. As the results and benefits in the use of formal methods are highly controversially judged within the computer science community, they have been definitely used successfully for the teaching of introductory lecturers of programming in computer science. In a popular approach, programs are specified and reasoned about in a framework of classical predicate logic. That is, states of imperative programs are described by predicates. Thus, proofs about the properties of programs are calculated using equivalence transformation on a symbolic level. However, carrying out a proof by hand is often a tedious job and also tends to be an error-prone process. But various types of errors can be avoided having mechanical support by some assisting tool.

In this thesis, we describe the design and implementation of a proof-editor from scratch that mechanizes proofs about imperative programs. Using the system, proofs are carried out fully under control of the user. That is, the system does not provide any automatic reasoning. We feel that this is rather an advantage than a pitfall. Deploying automated reasoning typically diminish the elegance of the proof format or excludes quantifiers from the logic. The system provides a graphical user interface including graphical elements such as buttons and menus, but also offers a flexible user interaction with the system through mouse movements and inputs. Capabilities to extend the system make the system a general purpose tool for reasoning in predicate calculus. The incorporation of a type inference mechanism increases the confidence in proofs without introducing to many overwhelming details.

As a key result of this thesis, we have proposed and implemented a novel approach that, employing known results from mechanizing mathematics, provides an elegant and efficient solution to the problem of quantifiers in predicate logic. Choosing Oberon System 3 as platform for the system turned out to be quite fortunate, since the Gadgets toolkit provides a powerful and reliable graphical user interface builder. The abstract text model of Oberon enables an elegant solution for the integration of textual and structured editing at little costs. However, we do not expect that this kind of tool can seriously be utilized in a industrial software development process. But it could successfully be used in academic classes that teach formal program development.