Type safety of C# and .Net CLR

Doctoral Thesis ETH No. 17003

Type Safety of C♯ and .NET CLR

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH
(ETH ZÜRICH)

for the degree of
Doctor of Technical Sciences

presented by
Nicu Georgian Fruja
Dipl. Math.–University of Bucharest
born January 13, 1977
citizen of Romania

accepted on the recommendation of
Prof. Dr. Thomas R. Gross, examiner
Prof. Dr. Robert Stärk, co-examiner

2007

Abstract

Type safety plays a crucial role in the security enforcement of any typed programming language. This thesis presents a formal proof of C♯'s type safety. For this purpose, we develop an abstract framework for C♯, comprising formal specifications of the language's grammar, of the statically correct programs, and of the static and operational semantics. Using this framework, we prove that C♯ is type-safe, by showing that the execution of statically correct C♯ programs does not lead to type errors.

The bytecode resulting from compiling C♯ programs is executed on the Common Language Runtime (CLR), the managed execution environment of the .NET Framework. As the bytecode may be transmitted over a network and get corrupted as a result, the bytecode execution may perform malicious operations. Therefore, to guarantee type safety, the CLR employs a bytecode verification, *i.e.*, a static analysis that performs several consistency checks before the bytecode is run. We show type safety of the bytecode language by proving the soundness of the bytecode verification. The abstract framework developed for the proof encompasses formal specifications of the bytecode language's abstract syntax, of the well-typedness constraints ensured by the bytecode verification, and of the bytecode language's static and dynamic semantics. Finally, we demonstrate that legal and well-typed bytecode programs do not lead to any type error when executed.

This thesis reveals an important number of relevant ambiguities in the official specifications of C♯ and CLR and, in certain cases, even the absence of a specification at all. Thus, we identify and fill several gaps in the two official documents. We also point out a series of inconsistencies between different implementations of C♯ and CLR and their official specifications.

Zusammenfassung

Typsicherheit nimmt einen entscheidenden Platz in der Gewährleistung der Sicherheit einer typisierten Programmiersprache ein. Diese Doktorarbeit bietet einen formalen Beweis der Typsicherheit von C♯. Hierfür erarbeiten wir ein abstraktes System für C♯, bestehend aus den formalen Spezifikationen der Grammatik der Sprache, der statischen Korrektheit von Programmen und der statischen und operativen Semantik. Basierend auf diesem System beweisen wir die Typsicherheit von C♯, indem wir aufzeigen, dass die Ausführung eines statisch korrekten C♯ Programms nicht zu Typfehlern führt.

Das kompilierte Ergebnis eines C♯ Programms wird als Bytecode auf einer verwalteten Umgebung namens Common Language Runtime (CLR) ausgeführt. Da Bytecode über ein Netzwerk übertragen werden und folglich korrumpiert werden kann, kann die Ausführung von Bytecode zu böswilligen Operationen führen. Um die Typsicherheit zu garantieren, setzt CLR folglich eine Bytecode-Verifikation ein, das heisst eine statische Analyse, welche mehrere Konsistenzüberprüfungen vornimmt bevor der Bytecode ausgeführt wird. Wir zeigen, dass der Bytecode typsicher ist, indem wir die Korrektheit der Bytecode-Verifikation beweisen. Das abstrakte System, welches wir für den Beweis entwickelt haben, umfasst die Spezifikationen der abstrakten Syntax der Bytecode-Sprache, der Wohl-Typisiertheits-Bedingungen, welche durch die Bytecode-Verifikation sichergestellt werden müssen, und der statischen und dynamischen Semantik der Bytecode-Sprache. Abschliessend legen wir dar, dass legale, wohl-typisierte Bytecode Programme während ihrer Ausführung zu keinerlei Typffehlern führen.

Diese Doktorarbeit zeigt eine bedeutende Anzahl von relevanten Zweideutigkeiten in der offiziellen Spezifikationen von C♯ und CLR auf und in gewissen Fällen sogar das Fehlen einer Spezifikation überhaupt. In diesem Sinne identifizieren wir und schliessen wir einige Lücken in den zwei offiziellen Dokumenten. Des weiteren zeigen wir mehrere Inkonsistenzen zwischen verschiedenen Implementationen von C♯ und CLR und deren offiziellen Spezifikationen auf.