Detecting synchronization defects in multi-threaded object-oriented programs

Author(s):
Praun, Christoph von

Publication Date:
2004

Permanent Link:

https://doi.org/10.3929/ethz-a-004756402 →

Diss. ETH No. 15524

Detecting Synchronization Defects in Multi-Threaded Object-Oriented Programs

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

for the degree of
Doctor of Technical Sciences

presented by
Christoph von Praun
Dipl.-Inf. TU-München
born January 20, 1972
citizen of Germany

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

2004

Abstract

This dissertation describes an efficient and automated approach to determine synchronization defects in multi-threaded object-oriented programs. The approach is based on the key observation that object-oriented programs provide guarantees about data confinement and encapsulation that can be leveraged by the design of a static analysis and a runtime checker. To be practical, the techniques are demonstrated on the Java programming language.

The static analysis operates on an abstract model of threads and data, and simulates the execution of a parallel program on these abstract domains. Thereby, precise information about locking, thread activity, object access, and object escape is recorded in a context-sensitive manner. This symbolic execution provides a general platform to assess properties of parallel programs. The focus of this dissertation is on the detection of three possible sources of errors: data races, methods that may not execute atomically, and deadlock. The static analysis of object-oriented languages is generally limited by the effects of aliasing and the resulting difficulty to disambiguate dynamically allocated data and locks. While approximations of the static analysis reduce the accuracy of the results, we have found the reports of potential synchronization defects to be highly useful in practice: Overreporting may occur, however at a rate that is amenable to manual inspection. True defects may be overlooked but such underreporting can be limited to certain cases that we observed rarely in practice.

Two alternative software mechanisms are developed to assess concurrency and locking at runtime: First, object race detection checks if access to shared objects follows a locking discipline. Second, object consistency guarantees that threads behave so that access to individual objects is serializable and happens without harmful interference. Both mechanisms are implemented as a sparse program instrumentation that is guided by the static analysis and optimized with standard compiler techniques. The runtime overhead is very low (on average 44% for object race checking and 25% for object consistency) and well spent in the light of the benefits.

The trend towards thread-level parallelism and multi-threaded computer systems make precise information about concurrency and synchronization indispensable for correct program translation, optimization, and execution. The techniques presented in this dissertation are promising steps towards providing this information and making the detection of synchronization defects a default option for compilers and runtime systems of multi-threaded object-oriented programs.

Kurzfassung

Die vorliegende Arbeit beschreibt einen effizienten und automatisierten Ansatz zur Erkennung von Synchronisationsfehlern in nebenläufigen objektorientierten Programmen. Der Ansatz basiert auf der Beobachtung, dass objektorientierte Programme gewisse Garantien über die Erreichbarkeit und Kapselung von Daten festlegen; solche Garantien können die Effizienz und Genauigkeit von statischen und dynamischen Verfahren zur Erkennung von Synchronisationsfehlern verbessern. Die in dieser Dissertation entwickelten Techniken werden an der Programmiersprache Java beispielhaft vorgestellt und evaluiert.

Die statische Analyse basiert auf einem symbolischen, d.h. simulierten, Programmablauf mit abstraktem Thread- und Datenmodell. Die Simulation registriert die Verwendung von Locks, Objektzugriffe und die Erreichbarkeit von Objekten durch nebenläufige Threads. Die Simulation ist kontext-sensitiv, d.h. bei der Analyse einzelner Methodenaufrufe wird der Programm- und Datenkontext, in welchem eine Methode zur Ausführung gelangt, berücksichtigt. Die symbolische Programmausführung ist die Grundlage für weitere Analysen. Der Schwerpunkt in der vorliegenden Arbeit liegt auf Analysen zur Erkennung von drei möglichen Fehlerquellen: Data Races, Methoden deren Ausführung nicht atomar sein könnte und Deadlock. Die Analyse von objektorientierten Sprachen wird generell durch Aliasing und durch die daraus resultierende Ungenauigkeit, dynamisch allozierte Objekte und Locks zu unterscheiden, erschwert. Obgleich die notwendigen Abstraktions- und Näherungsverfahren die Präzision reduzieren, sind die Ergebnisse der statischen Erkennung von Synchronisationsfehlern äußerst nützlich in der Praxis. Einige Warnungen haben keine Entsprechung in tatsächlichen Programmausführungen (Overreporting); solche Warnungen sind typisch, sie tauchen jedoch nur in relativ geringer Anzahl auf und können mit moderatem Aufwand vom Benutzer erkannt und selektiert werden. Es ist andererseits möglich, dass die entwickelten Analyseverfahren keine Warnung generieren für Programme, die tatsächlich Synchronisationsfehler aufweisen (Underreporting). Dieses Phänomen tritt jedoch nur in Situationen auf, die wir genau bestimmen können, die aber nur selten in der Praxis vorkommen.

Neben den statischen Analysen werden zwei software-basierte Techniken vorgestellt, welche Informationen über aktuelle Locks und Nebenläufigkeit zur Laufzeit verfügbar machen. Der erste Mechanismus, genannt Object Race Detection, überprüft, ob nebenläufige Threads beim Zugriff auf gemeinsame Objekte eine bestimmte Lockdisziplin einhalten. Der zweite Mechanismus, genannt Object Consistency, garantiert, dass die Effekte von Objektzugriffen serialisierbar sind und keine Interferenz von Threads auftaucht, die zu Inkonsistenz von Daten führen könnte. Beide Mechanismen sind in Form einer Programminstrumentierung implementiert, die durch die Ergebnisse der statischen Analyse gesteuert und mit gängigen Compilertechniken optimiert wird; Ziel ist es möglichst wenige Objektzugriffe zu instrumentieren. Die Laufzeitkosten dieser Instrumentierungsvarianten sind sehr gering (im Mittel 44% bei Ob-

ject Race Detection und 25% bei Object Consistency) und lohnenswert angesichts der Vorteile, die die Erkennung von Synchronisationsfehlern mit sich bringt.

Der Trend zu Multithreading in Software und Hardware Systemen erfordert die Verfügbarkeit von genauer Information über Nebenläufigkeit und Synchronisation zur korrekten Übersetzung, Optimierung und Ausführung eines Programms. Die Techniken, welche in dieser Dissertation vorgestellt werden, sind erste Schritte zur Bereitstellung dieser Information und zur Etablierung von Mechanismen zur Erkennung von Synchronisationsfehlern in Compiler- und Laufzeitsystemen für objektorientierte nebenläufige Programme.