

Contents

	Page
Editorial	3
<hr/>	
ARTICLES	
<hr/>	
Securing Java with Local Policies <i>Massimo Bartoletti, Gabriele Costa, Pierpaolo Degano, Fabio Martinelli, and Roberto Zunino</i>	5
<p>We propose an extension to the security model of Java, that allows for specifying, analysing and enforcing history-based usage policies. Policies are defined by usage automata, that recognize the forbidden execution histories. Programmers can sandbox an untrusted piece of code with a policy, which is enforced at run-time through its local scope. A static analysis allows for optimizing the execution monitor: only the policies not guaranteed to be always obeyed will be enforced at run-time.</p>	
An Operational Semantics including „Volatile“ for Safe Concurrency <i>John Boyland</i>	33
<p>In this paper, we define a novel “write-key” operational semantics for a kernel language with fork-join parallelism, synchronization and “volatile” fields. We prove that programs that never suffer write-key errors are exactly those that are “data race free” and also those that are “correctly synchronized” in the Java memory model. This 3-way equivalence is proved using Twelf.</p>	
Resource Usage Protocols for Iterators <i>Christian Haack, and Clément Hurlin</i>	55
<p>We discuss usage protocols for iterator objects that prevent concurrent modifications of the underlying collection while iterators are in progress. We formalize these protocols in Java-like object interfaces, enriched with separation logic contracts. We present examples of iterator clients and proofs that they adhere to the iterator protocol, as well as examples of iterator implementations and proofs that they implement the iterator interface.</p>	

Universe-Type-Based Verification Techniques for Mutable Static Fields and Methods

85

A. J. Summers, S. Drossopoulou, and P. Müller

We present three novel techniques for the verification of invariants in the setting of Java-like languages including static fields and methods. Our techniques structure the heap through universe types, and extend the Visibility Technique of Müller et al.

In order to cater for mutable static fields, we extend the classical universe types heap topology with multiple trees, where each tree is rooted in a class. Thus classes may naturally own objects as static fields.

We present a basic version of our approach, which allows trees to be visited at the top and then navigated “downwards”, and which avoids dangerous call-backs through effects which track static method calls. As well as the usual kinds of proof obligations defining that certain invariants must hold at a given state, we employ a second kind of obligation to show that certain other invariants are preserved between two states (i.e., if they hold in the former state then they will still hold in the latter). This allows us to deal with invariants whose expected truth-value cannot always be determined statically in a modular way.

We then present two extensions of our basic technique, aimed at improving usability. Firstly, we introduce a new universe annotation to allow safe callbacks between trees, whereby trees may be visited not at the top, but at the point where a previous visit “had left off”.

Secondly, we refine our heap topology with a notion of ‘levels’, which stratify the heap and provide modularity with regard to library classes and the required effects annotations.

OUTLOOK

A brief outlook to the next issue

127