Verifying Concurrent List–Manipulating Programs by LTL Model Checking

Joost–Pieter Katoen and Thomas Noll and Stefan Rieger

RWTH Aachen University
Software Modeling and Verification Group
52056 Aachen, Germany
{katoen,noll,rieger}@cs.rwth-aachen.de

Abstract:

Techniques for the verification of elementary properties of concurrent pointer programs are indispensable. Programming with pointers is error–prone with potential pitfalls such as dereferencing null pointers and the creation of memory leaks. Pointer programming becomes even more vulnerable in a concurrent setting where data structures such as linked lists and trees are manipulated and inspected by several threads.

We present a model–checking approach to the verification of concurrent programs that manipulate singly–linked lists.

Our approach is illustrated by considering a simple concurrent programming language that besides the usual control structures offers primitives for pointer manipulation, cell creation and destruction, and (guarded) atomic regions that allow concurrency control constructs such as test–and–set primitives and monitors. An operational semantics is provided in terms of labeled transition systems in which states are equipped with a graph structure representing the current list configuration. List abstraction exploits a variant of summary nodes [→ Sagiv et al.] that represent more than M chained list cells where constant M is directly obtained from the formula to be checked. Each configuration is shown to have a canonical representation (up to isomorphism). The abstract semantics of any concurrent program in our language is finite, obtained in a fully mechanized manner, and keeps the minimal "distance" between program variables and summary nodes invariant. Over–approximation occurs in a very controlled manner; only assignments may yield nondeterminism as variables may get "too close" to summary nodes.

Properties are expressed in a first–order linear–time temporal logic (LTL) that is enriched with assertions on singly–linked lists such as reachability of cells, aliasing, and freshness of cells. Our logic is similar in spirit to NTL [→ Distefano et al.] and ETL [→ Sagiv et al.]. Opposed to NTL, we avoid the use of temporal operators inside quantification. In this way, involved mechanisms to keep track of the identities of individual cells are not needed. As a result, standard LTL model checking algorithms can be employed. The differences with ETL are more of a technical nature. ETL has a three–valued interpretation, whereas our logical interpretation is a standard binary one. Moreover, ETL–formulas are translated in first–order logic with transitive closure for the evaluation on a trace, whereas in our case traces are generated by labeled transition systems and used in standard LTL model checking.