Safe Pointers and Verifiable Automatic Heap Management without Garbage Collection

Submitted by Katie Dey on Sat, 03/10/2018 - 1:59pm. Contributor:
S. Tucker Taft

BIO
S. Tucker Taft is VP and Director of Language Research at AdaCore, and Product Architect for AdaCore's new "QGen" Model-Based Development toolsuite. Tucker led the Ada 9X language design team, culminating in the February 1995 approval of Ada 95 as the first ISO standardized object-oriented programming language. His specialties include programming language design, compiler implementation, high-integrity software, integrated development environments, real-time systems, parallel programming, and model-based development. Since 2001, Tucker has been a member of the ISO Rapporteur Group that developed Ada 2005 and Ada 2012. Most recently Tucker has been designing and implementing the parallel programming language ParaSail, and working on parallel programming extensions for Ada. Prior to joining AdaCore, Tucker was Founder and CTO of SofCheck, Inc., providing tools and technology to enhance software development quality and productivity. Prior to that Mr. Taft was a Chief Scientist at Intermetrics, Inc. and its follow-ons for 22 years. Tucker received an A.B. Summa Cum Laude degree from Harvard University, where he has more recently taught compiler construction and programming language design.

ABSTRACT
SPARK 2014 is a language intended for formal verification of software. It is based on a large subset of Ada 2012, a language that itself includes standard syntax for specifying pre and postconditions, subtype predicates, and type invariants. SPARK augments Ada 2012 with annotations for specifying global variable usage, dependencies between inputs and outputs, responsibilities for initialization and synchronization, loop variants and invariants, etc. These annotations enable various levels of proof, from simple absence of run-time errors, all the way to full functional correctness.

SPARK 2014 supports most features of Ada 2012, including Ada's object-oriented features, a subset of Ada's multitasking features including Ada's data-oriented synchronization mechanism known as protected types, Ada's generic templates, etc. One significant feature not currently supported in SPARK 2014 is the ability to create heap-based data structures linked together using pointers (called "access types" in Ada). This is in large part due to the challenges of formally verifying pointer-heavy code and manual heap management. Garbage collection eliminates the complexity of verifying manual heap management, but introduces the complexity of verifying the garbage collector, while introducing real-time and resource limitation concerns. Even without the complexities of verifying the heap management, the "aliasing" inherent in most use of pointers makes proofs that much harder.

Program verification in general depends heavily on having full understanding of any potential "aliasing" between distinct names that might refer to the same underlying memory, so that when an update is performed, appropriate (sound) conclusions can be drawn. SPARK enforces strict non-aliasing between two parameters passed to the same subprogram, or between a parameter and a global variable used by the subprogram, if updating is permitted of either. This means that when inside a subprogram, program proofs need not consider the possibility of aliasing between parameters and/or global variables. When considering adding pointers to SPARK, we felt it was essential to preserve these restrictions on aliasing. The notion of "pointer ownership," which restricts the number of pointers through which an object can be updated, provides the key ingredient to reduce aliasing, and to simplify heap management.

Program verification in general depends heavily on having full understanding of any potential "aliasing" between distinct names that might refer to the same underlying memory, so that when an update is performed, appropriate (sound) conclusions can be drawn. SPARK enforces strict non-aliasing between two parameters passed to the same subprogram, or between a parameter and a global variable used by the subprogram, if updating is permitted of either. This means that when inside a subprogram, program proofs need not consider the possibility of aliasing between parameters and/or global variables. When considering adding pointers to SPARK, we felt it was essential to preserve these restrictions on aliasing. The notion of "pointer ownership," which restricts the number of pointers through which an object can be updated, provides the key ingredient to reduce aliasing, and to simplify heap management.

This presentation will describe the approach taken to adding "owning" pointers and automatic heap management to SPARK, and the proof rules necessary to allow the formal verification of pointer-heavy SPARK programs.

For full information on the SPARK language, see:
* SPARK 2014 Overview:
* SPARK 2014 Reference Manual:
* Book on SPARK 2014:
J. McCormick, P. Chapin, "Building High Integrity Applications with SPARK, Cambridge University Press, 2015

<table>
<thead>
<tr>
<th>S. Tucker Taft</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>License:</strong> Creative Commons 2.5</td>
</tr>
</tbody>
</table>

Other available formats:

**Safe Pointers and Verifiable Automatic Heap Management without Garbage Collection**

Switch to normal viewer
Switch to experimental viewer

Presentation HCSS 2018