Building High Integrity Applications with SPARK

Software is pervasive in our lives. We are accustomed to dealing with the failures of much of that software – restarting an application is a very familiar solution. Such solutions are unacceptable when the software controls our cars, airplanes, and medical devices or manages our private information. These applications must run without error. SPARK provides a means, on the basis of mathematical proof, to guarantee that a program has no errors. This book provides an introduction to SPARK 2014 for students and developers wishing to master the basic concepts for building systems with SPARK.

SPARK is a formally defined programming language and a set of verification tools specifically designed to support the development of software used in high integrity applications. Using SPARK, developers can formally verify properties of their code such as information flow, freedom from runtime errors, functional correctness, security properties, and safety properties.

John W. McCormick is a professor of computer science at the University of Northern Iowa. He began his career at the State University of New York in 1979. He has served as secretary, treasurer, and chair of the Association for Computer Machinery Special Interest Group on Ada. In 1993 John was awarded the Chancellor's Award for Excellence in Teaching. He received the Special Interest Group on Ada Distinguished Service Award in 2002, as well as the Outstanding Ada Community Contributions Award in 2008. His additional awards include the Special Interest Group on Ada Best Paper and Presentation Award and the Ada Europe Best Presentation Award.

Peter C. Chapin is a professor of computer information systems at Vermont Technical College (VTC). Peter started at VTC in 1986 as an instructor in the Department of Electrical and Computer Engineering Technology teaching courses on microcontrollers and C programming. Since 2009 Peter has been Software Director of VTC's CubeSat Laboratory where he has worked with students using Ada and SPARK to program small-scale nano-satellites. VTC successfully launched a working CubeSat satellite into low Earth orbit on November 19, 2013. It is the first CubeSat programmed using SPARK.

Building High Integrity Applications with SPARK

JOHN W. McCORMICK
University of Northern Iowa

PETER C. CHAPIN
Vermont Technical College

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

32 Avenue of the Americas, New York, NY 10013-2473, USA

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107656840

© John W. McCormick and Peter C. Chapin 2015

This publication is in copyright. Subject to statutory exception
and to the provisions of relevant collective licensing agreements,
no reproduction of any part may take place without the written
permission of Cambridge University Press.

First published 2015

Printed in the United States of America

A catalog record for this publication is available from the British Library.

Library of Congress Cataloging in Publication Data
McCormick, John W., 1948–
Building high integrity applications with SPARK / John W. McCormick, University of Northern Iowa, Peter C. Chapin, Vermont Technical College.
pages cm
Includes bibliographical references and index.
ISBN 978-1-107-04073-1 (alk. paper)
1. SPARK (Computer program language) 2. Fault-tolerant computing.
I. Chapin, Peter C. II. Title.
QA76.73.S59M38 2015
004.2–dc23 2015014814

ISBN 978-1-107-04073-1 Hardback
ISBN 978-1-107-65684-0 Paperback

Additional resources for this publication at www.cambridge.org/us/academic/subjects/computer-science/programming-languages-and-applied-logic/building-high-integrity-applications-spark.

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party Internet Web sites referred to in this publication and does not guarantee that any content on such Web sites is, or will remain, accurate or appropriate.

Contents

Preface *page* ix

1 Introduction and Overview 1
1.1 Obtaining Software Quality 2
1.2 What Is SPARK? 8
1.3 SPARK Tools 10
1.4 SPARK Example 12
Summary 16
Exercises 16

2 The Basic SPARK Language 18
2.1 Control Structures 21
2.2 Subprograms 27
2.3 Data Types 32
2.4 Subprograms, More Options 57
Summary 64
Exercises 65

3 Programming in the Large 68
3.1 Definition Packages 69
3.2 Utility Packages 71
3.3 Type Packages 73
3.4 Variable Packages 83
3.5 Child Packages 87
3.6 Elaboration 93
Summary 95
Exercises 96

Cambridge University Press
978-1-107-04073-1 - Building High Integrity Applications with Spark
John W. McCormick and Peter C. Chapin
Frontmatter
More information

4 Dependency Contracts 99
4.1 Data Dependency Contracts 100
4.2 Flow Dependency Contracts 104
4.3 Managing State 110
4.4 Default Initialization 124
4.5 Synthesis of Dependency Contracts 127
Summary 130
Exercises 132

5 Mathematical Background 135
5.1 Propositional Logic 135
5.2 Logical Equivalence 139
5.3 Arguments and Inference 141
5.4 Predicate Logic 144
Summary 150
Exercises 151

6 Proof 155
6.1 Runtime Errors 155
6.2 Contracts 162
6.3 Assert and Assume 189
6.4 Loop Invariants 201
6.5 Loop Variants 211
6.6 Discriminants 216
6.7 Generics 224
6.8 Suppression of Checks 235
Summary 240
Exercises 243

7 Interfacing with SPARK 247
7.1 SPARK and Ada 247
7.2 SPARK and C 261
7.3 External Subsystems 269
Summary 282
Exercises 283

8 Software Engineering with SPARK 286
8.1 Conversion of SPARK 2005 286
8.2 Legacy Ada Software 291
8.3 Creating New Software 296

Cambridge University Press
978-1-107-04073-1 - Building High Integrity Applications with Spark
John W. McCormick and Peter C. Chapin
Frontmatter
More information

8.4 Proof and Testing 307
8.5 Case Study: Time Stamp Server 310
Summary 325

9 Advanced Techniques 326
9.1 Ghost Entities 326
9.2 Proof of Transitive Properties 330
9.3 Proof Debugging 336
9.4 SPARK Internals 347

Notes 355
References 359
Index 363

Preface

SPARK is a formally defined programming language and a set of verification tools specifically designed to support the development of high integrity software. Using SPARK, developers can formally verify properties of their code such as

- information flow,
- freedom from runtime errors,
- functional correctness,
- security policies, and
- safety policies.

SPARK meets the requirements of all high integrity software safety standards, including DO-178B/C (and the Formal Methods supplement DO-333), CENELEC 50128, IEC 61508, and DEFSTAN 00-56. SPARK can be used to support software assurance at the highest levels specified in the Common Criteria Information Technology Security Evaluation standard.

It has been twenty years since the first proof of a nontrivial system was written in SPARK (Chapman and Schanda, 2014). The 27,000 lines of SPARK code for SHOLIS, a system that assists with the safe operation of helicopters at sea, generated nearly 9,000 verification conditions (VCs). Of these VCs, 75.5% were proven automatically by the SPARK tools. The remaining VCs were proven by hand using an interactive proof assistance tool. Fast-forward to 2011 when the NATS iFACTS enroute air traffic control system went online in the United Kingdom. The 529,000 lines of SPARK code were proven to be "crash proof." The SPARK tools had improved to the point where 98.76% of the 152,927 VCs were proven automatically. Most of the remaining proofs were accomplished by the addition of user-defined rules, leaving only 200 proofs to be done "by review."

Although SPARK and other proof tools have significant successes, their use is still limited. Many software engineers presume that the intellectual challenges of proof are too high to consider using these technologies on their projects. Therefore, an important goal in the design of the latest version of SPARK, called SPARK 2014, was to provide a less demanding approach for working with proof tools. The first step toward this goal was the arrival of Ada 2012 with its new syntax for contracts. We no longer need to write SPARK assertions as special comments in the Ada code. The subset of Ada that is legal as SPARK language has grown to encompass a larger subset of Ada, giving developers a much richer set of constructs from which to develop their code.

The real power of SPARK 2014 is under the hood. The new set of SPARK tools is integrated with the front end of the GNAT compiler. This merger allows the SPARK tools to make direct use of the many code analyses performed by the GNAT compiler. Also, the new tools use an entirely new proof system based on the Why3 software verification system (Bobot et al., 2011). Why3 manages a collection of modern satisfiability modulo theory (SMT) provers such as Alt-Ergo (OCamlPro, 2014), CVC4 (New York University, 2014), YICES (Dutertre, 2014), and Z3 (Bjørner, 2012) that complete the actual proving of the contracts in our program. These underlying proof tools can handle far more situations than the original SPARK prover. Do not be put off by this high-powered mathematical foundation; you do not need knowledge of these low-level proof tools to use SPARK.

Another significant improvement in SPARK 2014 is the integration of proof and testing techniques. The Ada 2012 assertions in a SPARK program can be checked dynamically by running test cases. Alternatively, these assertions can be proven correct by the SPARK proof tools. Such a mixed verification approach allows us to incrementally increase the level of formality in our programs. Having the ability to combine testing and proof also allows us to more easily verify programs written in a mixture of languages.

It is useful to distinguish between SPARK as a programming language and the SPARK tools that perform the analysis and proof of program properties. The removal of difficult-to-analyze features such as access types and exceptions makes the SPARK 2014 language a subset of Ada 2012. Yet, the SPARK 2014 language also extends Ada 2012 with additional pragmas and aspects. The SPARK language is described in the SPARK *2014 Reference Manual* (SPARK Team, 2014a) and could potentially be implemented by many Ada compiler vendors.

At the time of this writing, Altran/AdaCore's implementation of SPARK is the only one that exists. Details of their implementation are described in the SPARK *2014 Toolset User's Guide* (SPARK Team, 2014b). Because there is only one implementation of SPARK, it is easy to assume that SPARK is really the union

Cambridge University Press
978-1-107-04073-1 - Building High Integrity Applications with Spark
John W. McCormick and Peter C. Chapin
Frontmatter
More information

of behaviors described in the reference manual and user's guide. However, it is possible that another implementation of SPARK may arise that implements the language in the reference manual while providing a different user experience.

As a convenience to the reader, in this book we have at times conflated SPARK the language and SPARK as implemented by Altran/AdaCore. With only one implementation of SPARK available, this approach seems reasonable, and it has the advantage of streamlining the presentation. For example, `SPARK_Mode`, described in Section 7.1.1, provides a way of identifying which parts of a program are SPARK. Technically, `SPARK_Mode` is a feature of Altran/AdaCore's implementation and not part of the SPARK language itself. Another implementation of SPARK could conceivably use a different mechanism for identifying SPARK code.

It is also important to understand that while the SPARK language is relatively static, the tools are rapidly evolving. As the tools mature they are able to automatically complete more proofs faster than before. You may find that recent versions of the tools do not require as many hints and assertions as older versions. In particular, some examples in this book may be provable by newer tools with fewer assertions required than we use here.

The SPARK language includes the Ada 2012 constructs necessary for object-oriented programming: tagged types, type extensions, dispatching operations, abstract types, and interface types. Contract notations are provided for ensuring that any operations applied to a superclass instance are also valid for instances of a subclass (the Liskov Substitution Principle). We have elected not to cover the object-oriented aspects of SPARK programming in this book.

Chapter Synopses

Chapter 1 provides an overview of high integrity software and some approaches commonly used to create high-quality software. The SPARK language and tool set are described in the context of reducing defect rates.

Chapter 2 introduces the basic subset of Ada 2012 that constitutes the SPARK language. SPARK's decision and loop structures will be familiar to all programmers. Subprograms come in two forms: functions and procedures. A significant portion of this chapter is devoted to types. Ada allows us to define our own simple and complex types. Using these types, we can create accurate models of the real world and provide valuable information to the SPARK tools so we can identify errors before the program is executed.

Chapter 3 is about the package. Packages facilitate the construction of large programs. We use packages to support separation of concerns, encapsulation,

Cambridge University Press
978-1-107-04073-1 - Building High Integrity Applications with Spark
John W. McCormick and Peter C. Chapin
Frontmatter
More information

information hiding, and more. A SPARK program consists of a main subprogram that uses services provided by packages.

Chapter 4 provides the first look at contracts – assertions about the program's behavior that must be implemented correctly by the developer. Dependency contracts provide the means to verify data dependencies and information flow dependencies in our programs. Incorrect implementation of data dependencies or flow of information can lead to security violations. The SPARK tools can check that the implementation conforms to the requirements of these contracts. This analysis offers two major services. First, it verifies that uninitialized data is never used. Second, it verifies that all results computed by the program participate in some way in the program's eventual output – that is, all computations are effective.

Chapter 5 provides a review of basic discrete mathematics useful in reading and writing contracts. Propositional and predicate logic provide the fundamental notions needed for expressing the assertions in contracts that specify functional behavior. The existential (there exists) and universal (for all) quantifiers of predicate logic are crucial in stating assertions about collections. Although not necessary to use the SPARK tools, we give a basic introduction to arguments and their validity. The verification conditions (VCs) generated by the SPARK proof tools are arguments that must be proven valid to ensure that our implementation fulfills the contracts. We leave it to the SPARK tools to do the actual proofs.

Chapter 6 describes how to use the SPARK tools to prove behavioral properties of a program. The first step is the proof that our program is free of runtime errors – that is, no exceptions can ever be raised. By including contracts such as preconditions and postconditions with each subprogram, we state the desired functionality. We can use the SPARK tools to show that these contracts are always honored, and thus, our code implements that functionality. In an ideal world, the tools would need only our code to verify it is free of runtime errors and meets all of its contracts. In reality, the tools are not yet smart enough to accomplish that verification alone. When a proof fails in a situation where we believe our code to be correct, we need to give the tool some additional information it can use to complete its proof. This information comes in the form of additional assertions.

Chapter 7 explores the issues around building programs that are not completely written in SPARK. It is often infeasible or even undesirable to write an entire program in SPARK. Some portions of the program may need to be in full Ada to take advantage of Ada features that are not available in SPARK such as access types and exceptions. It may be necessary for SPARK programs to call third-party libraries written in full Ada or in some other programming

Cambridge University Press
978-1-107-04073-1 - Building High Integrity Applications with Spark
John W. McCormick and Peter C. Chapin
Frontmatter
More information

language such as C. Of course, SPARK's assurances of correctness cannot be formally guaranteed when the execution of a program flows into the non-SPARK components.

Chapter 8 provides an overview of SPARK in the context of a software engineering process. We describe three common usages: conversion of SPARK 2005 programs to SPARK 2014, analysis or conversion of existing Ada programs, and development of new SPARK programs from scratch. We introduce the INFORMED design method for SPARK and discuss how testing and proof may be used in combination. Finally, we present a case study of developing an application using the INFORMED process.

In Chapter 9 we examine some advanced techniques for proving properties of SPARK programs including ghost entities and proof of transitive properties. We discuss the approaches we found useful for debugging proofs and provide a number of guidelines for completing difficult proofs. Finally, we give a brief tour of the internal workings of SPARK and suggestions for learning more.

Tools

Currently, the partnership of Altran and AdaCore provides the only implementation of SPARK 2014. They provide two versions. SPARK GPL with the corresponding GNAT GPL Ada compiler is available for free to software developers and students at `http://www.libre.adacore.com`. SPARK Pro with the corresponding GNAT Pro Ada compiler is intended for industrial, military, and commercial developers. Information on subscriptions to these professional tools is available at `http://www.adacore.com`.

Web Resources

A Web site with the complete source code for all of the examples and some of the exercises in the book may be found at `http://www.cambridge.org/us/academic/subjects/computer-science/programming-languages-and-applied-logic/building-high-integrity-applications-spark`.

The *Ada 2012 Language Reference Manual* and the *Rationale for Ada 2012* are available at `http://www.ada-auth.org/standards/12rm/html/RM-TTL.html` and `http://www.ada-auth.org/standards/12rat/html/Rat12-TTL.html`.

The *GNAT Reference Manual* and the *GNAT User's Guide* are available at `http://docs.adacore.com/gnat_rm-docs/html/gnat_rm.html` and `http://docs.adacore.com/gnat_ugn-docs/html/gnat_ugn.html`.

CAMBRIDGE

Cambridge University Press
978-1-107-04073-1 - Building High Integrity Applications with Spark
John W. McCormick and Peter C. Chapin
Frontmatter
More information

The SPARK *2014 Reference Manual* and the SPARK *2014 Toolset User's Guide* are available at `http://docs.adacore.com/spark2014-docs/html/lrm` and `http://docs.adacore.com/spark2014-docs/html/ug`.

Additional resources for Ada 2012 and SPARK may be found at `http://university.adacore.com`, `https://www.linkedin.com/groups/Ada-Programming-Language-114211/about`, and `https://www.linkedin.com/groups/SPARK-User-Community-2082712/about`

You can keep up with the latest SPARK developments at `http://www.spark-2014.org`.

Acknowledgments

We would like to thank the many SPARK 2014 developers at AdaCore and Altran who helped us with all of the nuances of the language and tools. We appreciate their work in reading and commenting on several drafts of this book. This book is much better as a result of their efforts. We also thank those individuals in industry who use SPARK on real projects for their feedback. Their comments, corrections, and suggestions have enormously improved and enriched this book. We are grateful to (in alphabetical order) Stefan Berghofer, Roderick Chapman, Arnaud Charlet, Robert Dorn, Claire Dross, Pavlos Efstathopoulos, Johannes Kanig, David Lesens, Stuart Matthews, Yannick Moy, Florian Schanda, and Tucker Taft.

Anyone who has written a textbook can appreciate the amount of time and effort involved and anyone related to a textbook author can tell you at whose expense that time is spent. John thanks his wife Naomi for her support and understanding. Peter thanks his wife Sharon for her patience and his students for their interest in SPARK.

John W. McCormick
University of Northern Iowa
`mccormick@cs.uni.edu`

Peter C. Chapin
Vermont Technical College
`PChapin@vtc.vsc.edu`