Static Analysis of Software

The Abstract Interpretation

Edited by
Jean-Louis Boulanger
Table of Contents

Introduction xi
Jean-Louis Boulanger

Chapter 1. Formal Techniques for Verification and Validation 1
Jean-Louis BOULANGER

1.1. Introduction 1
1.2. Realization of a software application 1
1.3. Characteristics of a software application 3
1.4. Realization cycle 4
1.4.1. Cycle in V and other realization cycles 4
1.4.2. Quality control (the impact of ISO standard 9001) 7
1.4.3. Verification and validation 9
1.5. Techniques, methods and practices 13
1.5.1. Static verification 13
1.5.2. Dynamic verification 35
1.5.3. Validation 39
1.6. New issues with verification and validation 39
1.7. Conclusion 41
1.8. Bibliography 42

Chapter 2. Airbus: Formal Verification in Avionics 45
Jean Souyris, David DELMAS and Stéphane DUPRAT

2.1. Industrial context 45
2.1.1. Avionic systems 45
2.1.2. A few examples 46
2.1.3. Regulatory framework 47
2.1.4. Avionic functions 47
2.1.5. Development of avionics levels 50
Chapter 3. Polyspace

3.1. Overview ............................................................................................................. 113
3.2. Introduction to software quality and verification procedures ...................... 114
3.3. Static analysis ..................................................................................................... 116
3.4. Dynamic tests .................................................................................................... 116
3.5. Abstract interpretation ...................................................................................... 117
3.6. Code verification ............................................................................................... 118
3.7. Robustness verification or contextual verification ............................................. 121
   3.7.1. Robustness verifications ............................................................................... 122
   3.7.2. Contextual verification ................................................................................. 122
3.8. Examples of Polyspace® results ......................................................................... 123
   3.8.1. Example of safe code .................................................................................. 123
   3.8.2. Example: dereferencing of a pointer outside its bounds ......................... 125
   3.8.3. Example: inter-procedural calls ................................................................. 126
3.9. Carrying out a code verification with Polyspace ................................................ 128
3.10. Use of Polyspace® can improve the quality of embedded software ............... 130
   3.10.1. Begin by establishing models and objectives for software quality .......... 130
   3.10.2. Example of a software quality model with objectives. ......................... 130
   3.10.3. Use of a subset of languages to satisfy coding rules. ............................. 132
   3.10.4. Use of Polyspace® to reach software quality objectives ....................... 133
3.11. Carrying out certification with Polyspace® ..................................................... 135
3.12. The creation of critical onboard software ...................................................... 135
3.13. Concrete uses of Polyspace® ............................................................................ 135
3.13.1. Automobile: Cummins engines improves the reliability of its motor's controllers 136
3.13.2. Aerospace: EADS guarantees the reliability of satellite launches 137
3.13.3. Medical devices: a code analysis leads to a recall of the device 138
3.13.4. Other examples of the use of Polyspace® 139
3.14. Conclusion 141
3.15. Bibliography 141

Chapter 4. Software Robustness with Regards to Dysfunctional Values from Static Analysis 143
Christèle FAURE, Jean-Louis BOULANGER and Samy AIT KACI

4.1. Introduction 143
4.2. Normative context 144
4.3. Elaboration of the proof of the robustness method 146
4.4. General description of the method 151
4.4.1. Required or effective value control 151
4.4.2. Computation of the required control 154
4.4.3. Verification of effective control 155
4.5. Computation of the control required 157
4.5.1. Identification of production/consumption of inputs 159
4.5.2. Computation of value domains 160
4.6. Verification of the effective control of an industrial application 161
4.6.1. Target software 161
4.6.2. Implementation 163
4.6.3. Results 169
4.7. Discussion and viewpoints 172
4.8. Conclusion 173
4.9. Bibliography 174

Chapter 5. CodePeer – Beyond Bug-finding with Static Analysis 177
Steve BAIRD, Arnaud CHARLET, Yannick MOY and Tucker TAFT

5.1. Positioning of CodePeer 177
5.1.1. Mixing static checking and code understanding 177
5.1.2. Generating contracts by abstract interpretation 179
5.2. A tour of CodePeer capabilities 182
5.2.1. Find defects in code 182
5.2.2. Using annotations for code reviews 184
5.2.3. Categorization of messages 186
5.2.4. Help writing run-time tests 187
5.2.5. Different kinds of output 188
<table>
<thead>
<tr>
<th>Chapter 6. Formal Methods and Compliance to the DO-178C/ED-12C Standard in Aeronautics</th>
<th>207</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emmanuel LEDINOT and Dillon PARIENTE</td>
<td></td>
</tr>
<tr>
<td>6.1. Introduction</td>
<td>207</td>
</tr>
<tr>
<td>6.2. Principles of the DO-178/ED-12 standard</td>
<td>208</td>
</tr>
<tr>
<td>6.2.1. Inputs of the software development process</td>
<td>208</td>
</tr>
<tr>
<td>6.2.2. Prescription of objectives</td>
<td>209</td>
</tr>
<tr>
<td>6.3. Verification process</td>
<td>212</td>
</tr>
<tr>
<td>6.4. The formal methods technical supplement</td>
<td>218</td>
</tr>
<tr>
<td>6.4.1. Classes of formal methods</td>
<td>219</td>
</tr>
<tr>
<td>6.4.2. Benefits of formal methods to meet DO-178C/ED-12C objectives</td>
<td>221</td>
</tr>
<tr>
<td>6.4.3. Verification of the executable code at the source level</td>
<td>223</td>
</tr>
<tr>
<td>6.4.4. Revision of the role of structural coverage</td>
<td>225</td>
</tr>
<tr>
<td>6.4.5. Verification of the completeness of requirements and detection of unintended functions</td>
<td>227</td>
</tr>
<tr>
<td>6.5. LLR verification by model-checking</td>
<td>229</td>
</tr>
<tr>
<td>6.6. Contribution to the verification of robustness properties with Frama-C</td>
<td>234</td>
</tr>
<tr>
<td>6.6.1. Introduction to Frama-C</td>
<td>234</td>
</tr>
<tr>
<td>6.6.2. Presentation of the case study</td>
<td>241</td>
</tr>
<tr>
<td>6.6.3. Analysis process of the case study</td>
<td>243</td>
</tr>
<tr>
<td>6.6.4. Conclusion on the case study</td>
<td>252</td>
</tr>
<tr>
<td>6.7. Static analysis and preservation of properties</td>
<td>252</td>
</tr>
<tr>
<td>6.8. Conclusion and perspectives</td>
<td>256</td>
</tr>
<tr>
<td>6.9. Appendices</td>
<td>258</td>
</tr>
<tr>
<td>6.9.1. Automatically annotating a source code</td>
<td>258</td>
</tr>
<tr>
<td>6.9.2. Automatically subdividing input intervals</td>
<td>259</td>
</tr>
<tr>
<td>6.9.3. Introducing cut strategies for deductive verification</td>
<td>261</td>
</tr>
<tr>
<td>6.9.4. Combining abstract interpretation, deductive verification and functions which can be evaluated in assertions</td>
<td>263</td>
</tr>
<tr>
<td>6.9.5. Validating ACSL lemmas by formal calculus</td>
<td>265</td>
</tr>
<tr>
<td>6.9.6. Combining static and dynamic analysis</td>
<td>266</td>
</tr>
</tbody>
</table>
Chapter 7. Efficient Method Developed by Thales for Safety Evaluation of Real-to-Integer Discretization and Overflows in SIL4 Software

Anthony BALOTTO, Fateh KAOKAI, Rafael MARCANO and Daniel DRAGO

7.1. Introduction 273
7.2. Discretization errors in the embedded code production chain 274
  7.2.1. Presentation of the issue 274
  7.2.2. Objective of the analysis of the real-to-integer discretization 278
7.3. Modeling of the creation and propagation of uncertainties 280
  7.3.1. Creation of uncertainties 280
  7.3.2. Propagation of uncertainties 287
7.4. Good practice of an analysis of real-to-integer discretization 294
  7.4.1. Code extraction 294
  7.4.2. Functional code reorganisation 294
  7.4.3. Algorithmic breakdown in basic arithmetic relations 295
  7.4.4. Computation of uncertainties 295
7.5. Arithmetic overflow and division by zero 297
  7.5.1. Analysis of arithmetic overflow risk 297
  7.5.2. Analysis of the risk of division by zero 298
7.6. Application to a rail signalling example 299
  7.6.1. General presentation of the communication-based train controller system 299
  7.6.2. Example of analysis of the behavior of speed control 300
  7.6.3. Industrial scale view: a few numbers 306
7.7. Conclusion 307
7.8. Annex: proof supplements 308
  7.8.1. Proof 1: existence and unicity of integer division 308
  7.8.2. Proof 2: framing the error of integer division 312
  7.8.3. Proof 3: rules of the arithmetic of uncertainty intervals 314
  7.8.4. Proof 4: framing of uncertainties from a product 314
7.9. Bibliography 317

Conclusion and viewpoints 319
Jean-Louis BOULANGER

Glossary 323

List of Authors 327

Index 329