Formal analysis of confidentiality conditions related to data leakage

Ibrahim Shiyam

Doctor of Philosophy

University of York
Computer Science

September 2016
This thesis is dedicated

To my parents, Zareena Ali and Ali Hassan Fulhu,
for their endless love and support throughout my life.

To my wife Fathimath Humam and daughter Aishath Zara Shiyam,
for their patience and understanding.

To my brother, Ahmed Shah Ali,
for always being there for me.
Abstract

The size of the financial risk, the social repercussions and the legal ramifications resulting from data leakage are of great concern. Some experts believe that poor system designs are to blame. The goal of this thesis is to use applied formal methods to verify that data leakage related confidentiality properties of system designs are satisfied. This thesis presents a practically applicable approach for using Banks’s confidentiality framework, instantiated using the Circus notation.

The thesis proposes a tool-chain for mechanizing the application of the framework and includes a custom tool and the Isabelle theorem prover that coordinate to verify a given system model. The practical applicability of the mechanization was evaluated by analysing a number of hand-crafted systems having literature related confidentiality requirements.

Without any reliable tool for using BCF or any Circus tool that can be extended for the same purpose, it was necessary to build a custom tool. Further, a lack of literature related descriptive case studies on confidentiality in systems compelled us to use hand-written system specifications with literature related confidentiality requirements.

The results of this study shows that the tool-chain proposed in this thesis is practically applicable in terms of time required. Further, the efficiency of the proposed tool-chain has been shown by comparing the time taken for analysing a system both using the mechanised approach as well as the manual approach.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>3</td>
</tr>
<tr>
<td>List of Tables</td>
<td>11</td>
</tr>
<tr>
<td>List of Figures</td>
<td>15</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>19</td>
</tr>
<tr>
<td>Declaration</td>
<td>21</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>23</td>
</tr>
<tr>
<td>1.1 Confidentiality</td>
<td>23</td>
</tr>
<tr>
<td>1.2 Confidentiality engineering</td>
<td>24</td>
</tr>
<tr>
<td>1.3 Data leakage through communication channels</td>
<td>28</td>
</tr>
<tr>
<td>1.4 Motivation</td>
<td>32</td>
</tr>
<tr>
<td>1.4.1 Mandatory regulation demanding secure-by-design practices</td>
<td>33</td>
</tr>
<tr>
<td>1.4.2 Cost of data leakages</td>
<td>34</td>
</tr>
<tr>
<td>1.5 Hypothesis</td>
<td>38</td>
</tr>
<tr>
<td>1.5.1 The challenges</td>
<td>39</td>
</tr>
<tr>
<td>1.5.2 Testing the hypothesis</td>
<td>40</td>
</tr>
<tr>
<td>1.6 Contributions</td>
<td>41</td>
</tr>
<tr>
<td>1.7 Thesis structure</td>
<td>42</td>
</tr>
<tr>
<td><strong>2 Background</strong></td>
<td>43</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>43</td>
</tr>
</tbody>
</table>
## Contents

1. Analyzing systems with a confidentiality requirement ........................................... 44
   1.1 Abuse case ......................................................................................... 44
   1.2 PROMELA and the SPIN model checker .............................................. 46
   1.3 Integrating process design and SecureUML ........................................... 48
   1.4 Secure Tropos - Secure requirements engineering with reasoning ............... 48
   1.5 ConAn tool - Automated confidentiality analysis ..................................... 49
   1.6 InDico - Automated analysis of business processes for confidentiality .......... 50
   1.7 CONCHITA framework ...................................................................... 50
   1.8 TEES confidentiality model .................................................................... 51
   1.9 Confidentiality Properties and the B Method .......................................... 51
   1.10 Banks’s confidentiality framework (BCF) .............................................. 53
       1.10.1 The conceptual basis for BCF ..................................................... 54
       1.10.2 Advantages and limitations of BCF ............................................ 56
   1.11 Unifying Theories of Programming (UTP) ............................................ 57
       1.11.1 UTP theories .............................................................................. 58
       1.11.2 Program correctness ................................................................... 64
       1.11.3 Refinement ................................................................................ 67
       1.11.4 BCF in UTP .............................................................................. 68
       1.11.5 Possible twin state space .............................................................. 69
   1.12 Circus: a formal specification language .................................................... 71
       1.12.1 Advantages of Circus .................................................................. 71
       1.12.2 Uses of Circus .......................................................................... 72
       1.12.3 Challenges of using the Circus notation ....................................... 73
   1.13 BCF using Circus .................................................................................. 74
       1.13.1 User inference through observation .............................................. 74
       1.13.2 Formalising a confidentiality requirement ...................................... 80
   1.14 Analysing confidentiality requirements using BCF in Circus ....................... 81
       1.14.1 Back propagation laws .................................................................. 82
   1.15 Limitations of BCF in Circus ............................................................... 84

6
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9</td>
<td>Summary</td>
<td>85</td>
</tr>
<tr>
<td>3</td>
<td>Mechanisation of BCF</td>
<td>87</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>87</td>
</tr>
<tr>
<td>3.2</td>
<td>Practicality</td>
<td>89</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Rationale for a custom tool for mechanising BCF in Circus</td>
<td>89</td>
</tr>
<tr>
<td>3.2.2</td>
<td>The proposed mechanisation of BCF in Circus</td>
<td>90</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Design decisions</td>
<td>94</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Requirements of the major components in the architecture</td>
<td>99</td>
</tr>
<tr>
<td>3.2.5</td>
<td>The mechanised analysis process</td>
<td>104</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Interpreting the result of a mechanised analysis</td>
<td>106</td>
</tr>
<tr>
<td>3.3</td>
<td>Fixing the input prefix law</td>
<td>107</td>
</tr>
<tr>
<td>3.4</td>
<td>Suitability</td>
<td>110</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Types of data leakage supported by BCF in Circus</td>
<td>111</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Analysing data leakage through indirect communication using BCF in Circus</td>
<td>114</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Confidentiality violation through recursion</td>
<td>117</td>
</tr>
<tr>
<td>3.5</td>
<td>Efficiency</td>
<td>117</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Comparison of efficiency between the manual and the mechanized analysis</td>
<td>118</td>
</tr>
<tr>
<td>3.6</td>
<td>Summary</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>An approach for evaluating the mechanisation of BCF</td>
<td>121</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>121</td>
</tr>
<tr>
<td>4.2</td>
<td>The advantage of mechanisation over a manual approach</td>
<td>122</td>
</tr>
<tr>
<td>4.3</td>
<td>Value of the mechanisation</td>
<td>134</td>
</tr>
<tr>
<td>4.4</td>
<td>Benchmark for evaluation</td>
<td>134</td>
</tr>
<tr>
<td>4.5</td>
<td>Limitations of the catalogue approach for evaluation</td>
<td>135</td>
</tr>
<tr>
<td>4.6</td>
<td>Evaluation plan</td>
<td>137</td>
</tr>
<tr>
<td>4.7</td>
<td>Summary</td>
<td>141</td>
</tr>
</tbody>
</table>
# A systematic literature search for case study material

## 5.1 Introduction

## 5.2 Systematic literature search for case study material

### 5.2.1 Research question

### 5.2.2 Identification of indexing services

### 5.2.3 Inclusion criteria

### 5.2.4 Search keywords

### 5.2.5 Literature selection

## 5.3 Patterns of confidentiality requirements

### 5.3.1 Deriving patterns of confidentiality requirements

## 5.4 Subtleties in formalizing generic patterns of confidentiality requirements

### 5.4.1 Scenarios where different subtleties with inequality between two sets may satisfy a confidentiality requirement

## 5.5 Identifying and formalizing generic patterns of confidentiality requirements

## 5.6 Generalized patterns of confidentiality requirements

## 5.7 Confidentiality requirement patterns in literature

## 5.8 Patterns in software engineering

## 5.9 Limitations of the study

## 5.10 Summary

# Evaluation of mechanisation

## 6.1 Introducing

## 6.2 Mechanised analysis of confidentiality patterns

### 6.2.1 Mechanised analysis of a system having a confidentiality property that reflects pattern CP1

### 6.2.2 System requirement specification - Bank information system

### 6.2.3 Formal specification - Bank information system

### 6.2.4 Formalising the confidentiality requirement
Contents

6.2.5 Structure of the Circus specifications used in the mechanised analysis ........................................... 209
6.2.6 Using the mechanised tool to analyse the system .............................................................................. 211
6.2.7 Strengthening a weak specification ............................................................................................. 212
6.2.8 An example of strengthening a weak specification ........................................................................ 215
6.2.9 Strengthening the specification ........................................................................................................ 217
6.2.10 Results of the analysis .................................................................................................................... 224
6.2.11 Negative testing .............................................................................................................................. 225
6.2.12 Analysing other confidentiality patterns .......................................................................................... 227
   Analysing the confidentiality pattern CP2 ............................................................................................. 227
   Analysing the confidentiality pattern CP3 ............................................................................................. 234
   Analysing the confidentiality pattern CP4 ............................................................................................. 243
   Analysing the confidentiality pattern CP5 ............................................................................................. 252
6.2.13 A comparison of results of the mechanised analysis ......................................................................... 255

6.3 Summary .................................................................................................................................................. 258

7 Evaluation .................................................................................................................................................. 259
7.1 Introduction ............................................................................................................................................. 259
7.2 Factors that could have influenced the quality of the analysis ............................................................. 260
7.3 Benefits derived from the mechanisation .............................................................................................. 261
7.4 Contributions and Limitations .............................................................................................................. 263
7.5 Mechanization vs. manual back propagation ....................................................................................... 267
7.6 A critical analysis of the adopted mechanisation approach ................................................................. 267
7.7 Critical factors that would have altered the direction of this research ................................................... 270
7.8 Further work .......................................................................................................................................... 271

A Appendix .................................................................................................................................................. 279
A.1 Modelling a system using the Circus notation ...................................................................................... 279
   A.1.1 Defining the data types and state variables ...................................................................................... 279
   A.1.2 Establishing the communication channels ....................................................................................... 283
   A.1.3 Defining the system operations ........................................................................................................ 284
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1.4</td>
<td>Defining the overall behaviour of the system</td>
<td>285</td>
</tr>
<tr>
<td>A.1.5</td>
<td>Recursion</td>
<td>287</td>
</tr>
<tr>
<td>A.2</td>
<td>A comparison of the tools that provide any form of support for specifying systems in the Circus notation</td>
<td>288</td>
</tr>
<tr>
<td>A.3</td>
<td>Decisions regarding the development of a custom tool for BCF application</td>
<td>289</td>
</tr>
<tr>
<td>A.4</td>
<td>Translating CFAT notation to HOL</td>
<td>291</td>
</tr>
<tr>
<td>A.5</td>
<td>Description and formal specification of systems</td>
<td>295</td>
</tr>
<tr>
<td>A.5.1</td>
<td>Case study - Phone book system</td>
<td>295</td>
</tr>
<tr>
<td>A.5.2</td>
<td>Case study - Secure electronic examination system</td>
<td>302</td>
</tr>
<tr>
<td>A.5.3</td>
<td>Case study - ePurse system</td>
<td>318</td>
</tr>
</tbody>
</table>

### Abbreviations

**Abbreviations**  327

### List of References

**List of References**  331
List of Tables

1.1 Financial risk ratio against net asset value of two companies where more than 1 million records were compromised .................................................. 35
1.2 Data leakages between 2011-2015 where more than 1 million records were compromised ................................................................. 37
2.1 Features supported by the various approaches for analysing confidentiality requirements ................................................................. 53
2.2 A set of basic signatures common to a range of programming language theories ................................................................. 59
2.3 Naming convention used for Circus specifications .................................................. 75
2.4 A subset of back propagation laws of BCF in Circus by Banks (2012) ........ 83
3.1 The input and output format of each major component of the architecture of the mechanisation of BCF in Circus .................................................. 99
3.2 Analysing data leakage through indirect communication .................................................. 116
3.3 Time taken for analysing Figure 4.1 using BCF in Circus .................................................. 119
4.1 Descriptions of the constructs of the Patient details component of the Health Information System .................................................. 123
5.1 Relevant catalogues for software engineering research .................................................. 147
5.2 Patterns of confidentiality requirements .................................................. 150
5.3 A collection of confidentiality requirements from literature .................................................. 152
5.4 Possible ways of specifying an inequality between two sets $S$ and $\tilde{S}$ .................................................. 170
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5</td>
<td>A comparison of equality in function maps from Figure 5.2</td>
<td>172</td>
</tr>
<tr>
<td>5.6</td>
<td>How confidentiality properties are addressed by subtleties in set inequality.</td>
<td>173</td>
</tr>
<tr>
<td>5.7</td>
<td>Possible ways of specifying an inequality between two sets $S$ and $S'$</td>
<td>178</td>
</tr>
<tr>
<td>5.8</td>
<td>Bit representations of bank balances between 500 and 503</td>
<td>179</td>
</tr>
<tr>
<td>5.9</td>
<td>A catalogue of generalized patterns of confidentiality requirements</td>
<td>181</td>
</tr>
<tr>
<td>6.1</td>
<td>Mapping a system requirement specification to structures in a Circus</td>
<td>192</td>
</tr>
<tr>
<td>6.2</td>
<td>Roles and Permissions Matrix of the Bank information system</td>
<td>194</td>
</tr>
<tr>
<td>6.3</td>
<td>Bank information system - Description of the Circus actions</td>
<td>196</td>
</tr>
<tr>
<td>6.4</td>
<td>Bank information system - Description of the user roles</td>
<td>197</td>
</tr>
<tr>
<td>6.5</td>
<td>Bank information system - Description of the basic types</td>
<td>198</td>
</tr>
<tr>
<td>6.6</td>
<td>Bank information system - Description of the state variables</td>
<td>198</td>
</tr>
<tr>
<td>6.7</td>
<td>Bank information system - Description of the state invariants</td>
<td>199</td>
</tr>
<tr>
<td>6.8</td>
<td>Results of the mechanised analysis of the Bank information system</td>
<td>224</td>
</tr>
<tr>
<td>6.9</td>
<td>Result of the mechanised evaluation of the Bank information system</td>
<td>227</td>
</tr>
<tr>
<td>6.10</td>
<td>Roles and Permissions Matrix of the Phone book system</td>
<td>230</td>
</tr>
<tr>
<td>6.11</td>
<td>Results of the mechanised analysis of the Phone book system</td>
<td>234</td>
</tr>
<tr>
<td>6.12</td>
<td>Roles and Permissions Matrix of the Secure electronic examination system</td>
<td>238</td>
</tr>
<tr>
<td>6.13</td>
<td>Formal specification of conditions required to satisfy the confidentiality requirement CR21</td>
<td>241</td>
</tr>
<tr>
<td>6.14</td>
<td>Results of the mechanised evaluation of the Secure electronic examination system</td>
<td>243</td>
</tr>
<tr>
<td>6.15</td>
<td>Roles and Permissions Matrix of the ePurse system</td>
<td>246</td>
</tr>
<tr>
<td>6.16</td>
<td>Results of the mechanised analysis of the ePurse system</td>
<td>249</td>
</tr>
<tr>
<td>6.17</td>
<td>Results of the mechanised analysis of the ePurse system</td>
<td>254</td>
</tr>
<tr>
<td>6.18</td>
<td>A comparison of the average analysis times for different systems</td>
<td>257</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Mapping a system requirement specification to structures in a <em>Circus</em> specification</td>
<td>276</td>
</tr>
<tr>
<td>A.1</td>
<td>Online expenditure tracker - Description of the state variables</td>
<td>281</td>
</tr>
<tr>
<td>A.2</td>
<td>Online expenditure tracker - Description of the state invariants</td>
<td>281</td>
</tr>
<tr>
<td>A.3</td>
<td>Operators in HiVe for the Z mathematical constructs</td>
<td>294</td>
</tr>
<tr>
<td>A.5</td>
<td>Phone book system - Description of the <em>Circus</em> actions</td>
<td>296</td>
</tr>
<tr>
<td>A.6</td>
<td>Phone book system - Description of the basic types</td>
<td>297</td>
</tr>
<tr>
<td>A.7</td>
<td>Phone book system - Description of the state variables</td>
<td>297</td>
</tr>
<tr>
<td>A.8</td>
<td>Phone book system - Description of the state invariants</td>
<td>298</td>
</tr>
<tr>
<td>A.10</td>
<td>Secure electronic examination system - Description of the basic types</td>
<td>304</td>
</tr>
<tr>
<td>A.11</td>
<td>Secure electronic examination system - Description of the free types</td>
<td>304</td>
</tr>
<tr>
<td>A.12</td>
<td>Secure electronic examination system - Description of the state variables</td>
<td>305</td>
</tr>
<tr>
<td>A.13</td>
<td>Secure electronic examination system - Description of the state invariants</td>
<td>307</td>
</tr>
<tr>
<td>A.14</td>
<td>Secure electronic examination system - Description of the <em>Circus</em> actions</td>
<td>310</td>
</tr>
<tr>
<td>A.16</td>
<td>ePurse system - Description of the <em>Circus</em> actions</td>
<td>319</td>
</tr>
<tr>
<td>A.17</td>
<td>ePurse system - Description of the basic types</td>
<td>319</td>
</tr>
<tr>
<td>A.18</td>
<td>ePurse system - Description of the free types</td>
<td>320</td>
</tr>
<tr>
<td>A.19</td>
<td>ePurse system - Description of the state variables</td>
<td>320</td>
</tr>
<tr>
<td>A.20</td>
<td>ePurse system - Description of the state invariants</td>
<td>321</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 How the work in this thesis is related to the rest of the information system security research ........................................ 29
1.2 Aspects of a data leakage .................................................. 31

2.1 Approaches for eliciting, analysing, transforming and formally verifying confidentiality properties in systems ........................................ 45
2.2 UTP theories, healthiness conditions and the theory of Circus ................................................................. 62
2.3 Weakest pre-condition and post condition of an ePurse payment operation ................................................. 67
2.4 Circus specification of the secret number system ................................................. 76
2.5 Inferences of Alice and Bob from system observations ................................................. 76

3.1 Value added from mechanising BCF in Circus ................................................. 88
3.2 The architecture and the flow of the mechanisation of BCF in Circus for analysing systems with a confidentiality requirement ................................................. 92
3.3 The process for evaluating a system with a confidentiality requirement using the manual approach as well as using the Confidentiality Framework Application Tool ................................................. 95
3.4 The CFAT editor window ................................................. 102
3.5 How the predicate \( \text{loginUser} \notin (\text{cashiers} \cup \text{managers}) \) is parsed using the CFAT tool ................................................. 103
3.6 The mechanised analysis process ................................................. 105
3.7 Specification of Secret Highest Bidder - code block 1 of 1 ................................................. 115
List of Figures

4.1 Specification of the patient details component of a Patient information system ........................................... 124
5.1 Keyword map describing the confidentiality engineering ecosystem .......................................................... 147
5.2 A function map for a function $S$ and possible function maps for different variants of its twin function $\tilde{S}$ ................................................................. 171
5.3 Deriving generalized patterns from patterns of confidentiality requirements ............................................. 175
6.1 Tripartite graph of user-to-role and role-to-permission assignments in the Bank information system .......... 195
6.2 Use case diagram for the Bank information system .................................................................................. 195
6.3 User roles and user observations ............................................................................................................ 201
6.4 Specification of Bank information system - code block 1 of 2 ................................................................... 204
6.5 Structure of the Circus specifications used in the mechanised analysis .................................................. 210
6.6 Intersection of user roles $A$, $B$ and $C$ .................................................................................................. 212
6.7 Pairwise disjoint statements for users roles $A$, $B$ and $C$ ..................................................................... 214
6.8 Specification of Secret Highest Bid - code block 1 of 1 ........................................................................... 218
6.9 Specification of Secret Highest Bid - code block 1 of 1 ........................................................................... 219
6.10 Specification of Bank information system - code block 1 of 2 ............................................................... 222
6.11 Use case diagram for the Bank information system ............................................................................. 231
6.12 Use case diagram for the Secure electronic examination system ........................................................... 239
6.13 Use case diagram for the e-Purse system ............................................................................................... 246
6.14 Counter example generated due to an insecure operation ...................................................................... 251
7.1 Some possible stages and paths that can be taken when moving from business goals to Circus specifications ................................................................. 278
A.1 Specification of Online expenditure tracker - code block 1 of 1 .............................................................. 282
A.2 Circus BNF as published in Freitas (2005) doctoral thesis .................................................................... 286
A.3 Use case diagram for the Phone book system ........................................................................................ 296
A.4 Specification of Phone book system - code block 1 of 2 ...................................................................... 300
A.5 Use case diagram for a Secure electronic examination system .............................................................. 303
List of Figures

A.6 Specification of Secure electronic examination system - code block 1 of 6 . 312
A.7 Use case diagram for the ePurse system . 318
A.8 Specification of ePurse system - code block 1 of 2 . 324
Acknowledgements

This work was supported by a scholarship from the Islamic Development Bank (IDB).

First and foremost, I thank my mother Zareena Ali and my father Ali Hassan Fulhu for going the whole nine yards to support me throughout my life.

I am indebted to my supervisor Jeremy Jacob for believing in me and for trusting me through the rugged terrain of my PhD journey. To benefit an open door policy from my supervisor was a testament of his support towards my research. Thank you for not giving up on me the day I walked in and told that I must pursue the impractical path of manually evaluating systems if I am to make progress towards my goal.

I thank my wife Fathimath Humam for her sacrifice and understanding throughout the length of this journey. I can’t thank you enough. It was my best source of inspiration that my lovely daughter Aishath Zara Shiyam understood and mentioned often on her own that her ‘daadoo’ needs to do his never ending work to get his ‘third hat’.

A special thank you to my brother Ahmed Shah Ali who has support me throughout this journey and throughout my life.

Apart from my supervisor, my dear friend Pedro De Oliveira Salazar Ribeiro has been the most significant contributor to my research in terms of being a sound board as well as a wonderful critique. I thank my colleague Gerard Ekembe Ngondi for being my other sound board and critique, especially towards the final months of my PhD.
Acknowledgements

In the PLASMA group, I must specially thank José Manuel Calderón Trilla and Glyn Faulkner for the valuable discussions in functional programming. And to Rudy Braquehais for giving me a crash course on property based testing.

I thank the various people who have explained me various concepts during the various stages of my journey including Michael J. Banks for the lengthy Skype conversation to explain me the lifted semantics, Leonardo de Freitas for agreeing for me to visit his office at the University of Newcastle and explaining and showing me the guts and bones of the CZT tool, Simon Foster for introducing me to the Isabelle/UTP framework, Frank Zeyda\(^1\) and Neeraj Kumar Singh\(^2\) for the valuable advice on shaping my approach towards customising the Isabelle/UTP framework and finally Jan Burse\(^3\) and Abderrahmane Feliachi\(^4\) for Isabelle theorem prover related discussions.

A special thanks to Jim McCarthy of the Department of Defence in Australia for contacting me almost 10 months later to let me know that they had declassified the Z mathematical tool-kit which was classified at the time I had requested for it.

I thank Justice Hassan Saeed and my dear friend Sharif Ahmed of the Family Court in the Republic of Maldives for assisting me in getting the case management procedure of that court. Finally, I thank my friends Dr. Faisal Saeed and Dr. Mohamed Adil for devoting their valuable time for me to test my technical explanations on them, the positive feedback of which was used to adjust the explanations so that people from outside the domain can understand.

\(^1\) University of Newcastle
\(^2\) University of Toulouse
\(^3\) XLOG Technologies GmbH
\(^4\) RATP Group
Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.
1 Introduction

1.1 Confidentiality

Confidentiality is one of the three classic goals of the data security triad where the other two are integrity and availability (Article 29 Working Party, 2013, p. 27). Confidentiality of a system is an assurance that information will not be leaked to an unauthorised audience by that system. Confidential information may include Personally Identifiable Information (PII) (Antón and Earp, 2001) and business secrets amongst other things. Disclosure of confidential information has the potential to compromise privacy, results in huge financial losses and also have social repercussions. The massive data leakage of Personally Identifiable Information of 33 million users at Ashley Madison in 2015 (BBC News, 2015) is a perfect example of such damage. However, this has been dwarfed by the recent announcement of the 2014 data breach at Yahoo Inc. (The New York Times Company, 2016).

“Security experts say the breach could bring about class-action lawsuits, in addition to other costs. An annual report by the Ponemon Institute in July found that the costs to re-mediate a data breach is $221 per stolen record. Added up, that would top Yahoo’s $4.8 billion sale price”


The financial aspects of such data leakages in organisations has been published in the media (see Table 1.2). However, the author was not able to find details on the approaches which the perpetrators used for stealing confidential information from these
1 Introduction

Therefore, there is no way to conclude whether these data leakages were due to poor system design. Studies conducted by security experts such as Cast Software Inc (2014), Verizon Enterprise Solutions (2014) and Depaula (2016) have identified poor system design as one of the reasons for data leakage. One such aspect of a poor system design can be the inconsistency between the functional requirements of a system and its confidentiality requirements. If a functional requirement and a confidentiality requirement of a system are inconsistent, confidential data maybe revealed through an implementation that only satisfies such a functional requirement.

To address the thesis scope and challenges, one must consider system development approaches that ensure that the confidentiality requirements are engineered into the design of an information system. Further, one must ensure that the chosen approach supports a mathematically accurate transformation from a system model to an implementation.

1.2 Confidentiality engineering

Confidentiality engineering can be defined as the integration of tools and techniques within the system development process whereby practitioners can verify the conformance of a system design against a given set of confidentiality requirements. Given that confidentiality is a non-functional property\(^1\), how can engineers verify whether the design of a system respects both the functional as well as the non-functional requirements of a system? Model driven verification (Holzmann and Joshi, 2004) and formal

\[\text{IDG Communications} \, 2013.\]

\[\text{Onabajo, 2009, p. 21.}\]
1.2 Confidentiality engineering

Thesis scope and challenges

The focus of the research presented in this thesis is on analysing models of information systems for data leakage, by detecting inconsistencies between functional requirements and confidentiality requirements. The motivation for this research is discussed in Section 1.4.

The hypothesis presents two main challenges.

1. Find an approach for system engineers to verify whether their system designs respect the data leakage related confidentiality requirements of a system.
2. Find an approach for system engineers to ensure that their system designs are transformed into implementations that are correct by construction.

These challenges demand the need for system development approaches where the engineers can verify that:

- The system designs are confidentiality assuring.
- The step by step transformation from system designs to implementation can be verified as preserving the embedded confidentiality assurances.

verification (Blanchet, 2008; Hadj-alouane et al., 2005) are different approaches whereby system designs can be verified for conformance to a given set of requirements.


Using BCF, the information flow in a system can be analysed. Login and logout functions that are usually implemented to secure a system are part of an access control mechanism that does not protect against information flow related security issues.

“Information flow (IF) analysis is a suitable verification technique that focuses on the information propagation throughout the system (end-to-end) rather than mere data access (point-to-point). IF analysis can identify leaks, so-called interferences, that circumvent access control mechanisms” (Accorsi and Wonnemann, 2010, p. 194). 

25
1 Introduction

The case for formal methods. Formal methods are techniques based on mathematics to
describe system properties.

“A method is formal if it has a sound mathematical basis, typically given
by a formal specification language. This basis provides the means of pre-
cisely defining notions like consistency and completeness and, more relevantly,
specification, implementation, and correctness” (Wing, 1990, p. 8).

Correctness of a system can be described as an assurance that a software does
exactly what it is supposed to do. This assurance is often seen as a reflection
of the success percentage of applying a collection of test scenarios on the sys-
tem. However some of the correctness demanding systems such as mission
critical systems, military systems and life-saving systems needed a much more
correctness-guaranteed approach for their design, development and verification
(Cofer, 2010; Hall and Chapman, 2002). This is due to the criticality of the target
application environment as well as the financing involved in such huge pro-
jects. The approach for testing systems using test cases does not provide such
correctness guarantees as the set of test cases define a finite set of scenarios.

“Program testing can be used to show the presence of bugs, but never to show
their absence!” (Dijkstra, 1972).

Potter, Sinclair and Till (Potter et al., 1996b) believe that formal methods was
born out of this push towards assuring the ‘accuracy and correctness’ of software.
This belief was further supported by many academics in the software engineering
discipline where they commonly agreed that errors introduced early in the
development process are much harder and time-consuming to fix if detected later
and also consequently cost a lot more (Woodcock and Davies, 1996, p. 1; Bowen,
1996, p. 31; Peine et al., 2008, p. 9; Defence Science and Technology Organisa-
1.2 Confidentiality engineering

Confidentiality vs. Privacy. Many researchers including [Jamal et al. (2014), Hayashi (2013), Moore and McSherry (2013), McClelland (2002), Mayer (2002), Anderlik and Rothstein (2001) and Mlinek and Pierce (1997)] argue that people often use the terms confidentiality and privacy interchangeably. For example, [Hansen (1971), Mayer (2002) and Tschantz and Wing (2008)] used the word privacy interchangeably with confidentiality, when discussing techniques that are used to analyse scenarios with a confidentiality requirement.

Both these words are used in relation to information secrecy. The topic of this thesis is focused on proposing an approach to make a certain confidentiality framework practical. Therefore, it is important to clarify the difference between these two terms and explain how one is related to the other.

A knowledge map of the field of information security research is presented in Figure 1.1. The objective of presenting Figure 1.1 is to show the association between confidentiality, privacy and security which are sometimes used interchangeably. Further, the knowledge map also highlights the area of the information security research discussed in this thesis.

Privacy refers to “the right of individuals or cooperative users to maintain confidentiality and control over their information when it’s disclosed to another party” [Porambage et al. 2016, p. 37].

Confidentiality refers to “the assurance on non-disclosure of sensitive resources to unauthorised subjects” [Margheri et al. 2015, p. 34].

In summary, privacy is an individual’s desire for information secrecy whereas confidentiality is an assurance for information secrecy. Privacy demands form a subset of information secrecy requirements, where other requirements might include secrecy of company information or government information, to name but two.
As shown in Figure 1.1, there are many aspects of confidentiality such as legal (Al-Fedaghi, 2012, p. 6), financial (Rubinstein, 2011, p. 1456), ethical (Orb et al., 2001) and data leakage related (Gordon, 2007) to name a few. In this thesis, we consider the data leakage related aspect of confidentiality. An introduction to data leakage is given in Section 1.3.

1.3 Data leakage through communication channels

Shabtai et al. (2012, p. 5) state that data leakage is the intentional or unintentional distribution of private or sensitive data to an unauthorised entity. Private and sensitive data are considered confidential and may include personal, corporate, military or government data. Some intentional and unintentional activities that may result in a data leakage are shown in Figure 1.2.

Different technological approaches are being used to provide data leakage detection and prevention such as designated Data Leak Prevention (DLP) systems, access control and encryption mechanisms, advanced/intelligent security measures and standard security measures such as firewalls, antivirus systems and intrusion detection systems. Apart from direct theft, data leakage may occur as a result of poor information infrastructure design (Cast Software Inc., 2014; Depaula, 2016; Verizon Enterprise Solutions, 2014) as well as poor data management practices amongst others. Among the mechanisms that may help in detecting and mitigating data leakages, is the implementation of information security policies. The information infrastructure of an organisation must be governed by such policies to ensure that the security of the data is always maintained whether in transit or at rest. This includes integrating such policies within the top-level

---

3 “Designated DLP solutions are intended to detect and prevent attempts to copy or send sensitive data, intentionally or unintentionally, without authorization, mainly by personnel who are authorized to access the sensitive information” (Shabtai et al., 2012, p. 9).

4 “Advanced or intelligent security measures include machine learning and temporal reasoning algorithms for detecting abnormal access to data” (Shabtai et al., 2012, p. 9).
1.3 Data leakage through communication channels

Figure 1.1: How the work in this thesis is related to the rest of the information system security research
Introduction

design of information systems. The focus of the research in this thesis is to provide a practically applicable approach for using a formal framework that supports such integrations.

Channels are used to communicate information between agents[^a] in an information system. Within the context of an information system, an individual with a low security clearance may use various channels to gather data that maybe classified at a higher classification. Such channels may include side channels, covert channels, inference channels and overt channels.

**side channel** A side channel is a physical observable side-effect of a computation, that an adversary can measure (Lawson 2009, p. 65).

**covert channel** A covert channel is described as “any communication channel that can be exploited by a process to transfer information in a manner that violates the systems security policy” (Latham 1986, p. 80).

**inference channel** An inference problem exists when a user with a lower security clearance uses information which he is authorized, to draw conclusions about information at a higher security clearance (Garvey et al. 1991, p. 119). Such a link that may allow the flow of information from a higher security class to a lower security class is an inference channel.

**overt channel** An overt channel is described as “a communication path within a computer system or network designed for the authorized transfer of data” (Lucena et al. 2006, p. 147).

BCF codifies the information a user may not gain by observing overt channels in an information system. The aim of this thesis is to extend the value of BCF. This has been achieved by mechanising BCF and evaluating the mechanisation to understand how effective it is for analysing system models for data leakages.

[^a]: “Agents are active components forming the system. They can be humans, devices, legacy software, etc” (De Landtsheer and Van Lamsweerde 2005, p. 41).
1.3 Data leakage through communication channels

Intentional activities
- Physical theft
- Social engineering
- Phishing
- Dumpster driving
- Malware distribution
- SQL injection

Unintentional activities
- Instant messaging
- Email
- Malicious webpage
- Hidden in SSL
- Removable media

Implications
- Legal liability
- Regulatory violations
- Lost productivity
- Bad business reputation

Motivation
- Corporate espionage
- Financial reward
- Privacy violation

Channels of communication
- side channels
  - Timing attacks
  - Power analysis
  - Cache attacks
- covert channels
  - Steganography
  - Encryption
- inference channels
- overt channels

Data leakage
- may occur as a result of
- may be caused through
- may result in
- may be minimized by using

Confidentiality
- assurances of a system
- may compromise
- reason about
- formализ communications over

Information flow theories
- formalized

Detection / Mitigation mechanisms
- Intrusion detection systems
- Thin client rollout
- Virtual desktop rollout
- Ban on removable storage
- Anti-virus
- Anti-spyware
- information security policies

Confidentiality policies
- include

Figure 1.2: Aspects of a data leakage
1 Introduction

1.4 Motivation

The regulatory demand for secure by design practices during system development and the size of the financial risk due to data leakage incidences motivate work on techniques to integrate confidentiality engineering during the system development process. System development practices that guarantee confidentiality, contribute towards assuring privacy as well as trustworthy computing. In this regard the work presented in this thesis is highly encouraging.

The untrusted access to confidential information leads to privacy violations. Such digital-era related privacy concerns have been raised as early as in the 1990s. One of the recommendations of the study conducted by The New York Public Service Commission during 1989 and 1990 on privacy in telecommunication services recommended that privacy promoting technologies should be encouraged in future service offerings (Rotenberg, 1993).

Technology and policy experts in security, privacy and networking who participated in the 2003 conference on the Grand Challenges in Trustworthy Computing organised by the Computing Research Association (2003) declared that:

- ensuring trustworthiness of important societal applications such as electronic voting systems and healthcare record databases is one of the great challenges in trustworthy computing.

- a possible progress on ensuring trustworthiness of important societal applications will be to assure users that systems are designed with strong mathematical guarantees that eventually can achieve confidentiality of records amongst other security requirements.

Even though software designers have started to include non-functional properties such as performance and reliability in the system development process, "security still remain an afterthought" (Giorgini et al., 2004, p. 2). The same issue has been highlighted by a number of author’s (CAUSE, 1997, p. 3; Mouratidis et al., 2003, p. 63; Mouratidis et al.)
In this regard, this thesis explores ways in which the value of a validation framework for systems with a confidentiality requirement can be extended. The eventual goal is to have a well tested and mature confidentiality validation framework so that the framework can be integrated into professional software development tool kits. The work presented in this thesis contributes towards realising this goal.

1.4.1 Mandatory regulation demanding secure-by-design practices

Privacy legislations and regulations in many countries (General Services Administration, 2005; Office for National Statistics, 2002; Office of Parliamentary Counsel, 1988; U.S. National Institute of Standards and Technology, 2014) mandate that personally identifiable information must never be released to unauthorised individuals. However, decisions about the implementation mechanisms to achieve legislative compliance is solely left to the parties who handle personal data of individuals. Dr. Ann Cavoukian (2009) first introduced the phrase Privacy by Design (see Figure 1.1) to refer to such mechanisms, back in the 90s.

“Privacy by Design refers to the philosophy and approach of embedding privacy into the design specifications of various technologies” (Cavoukian, 2009).

Both the preliminary and the final report of the U.S. Federal Trade commission (FTC) on “Protecting Consumer Privacy in an Era of Rapid Change” in 2010 and 2012 consecutively recommends companies to adopt Privacy by Design practices by building privacy at every stage of their product development (Federal Trade Commision (FTC), 2012; Federal Trade Commission, 2010). However, the FTC stopped short of using the
phrase Privacy by Design in its call to the U.S. Congress, but rather requested them to consider enacting baseline privacy legislation and reiterated its call for data security legislation.

The General Data Protection Regulation of The European Parliament and The European Council (2016) states that organisations must implement “data protection by design and by default” whereby organisations must ensure that personal data will be “processed in a manner that ensures appropriate security of the personal data, including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organisational measures (‘integrity and confidentiality’)”. The European Digital Rights (2012) and European Parliament (2014) elaborate these measures by specifying that technical aspects should include the design of software and hardware and that organisational aspects should include internal and external policies and current best practices, a call advocated earlier by Dr. Ann Cavoukian (2009) and later repeated by the U.S. FTC (Federal Trade Commission, 2012; Federal Trade Commission, 2010).

This shows a shift from general compliance recommendations and directives towards legislation and regulation that mandates the adoption of secure-by-design practices during software design and development. It is also important to note here that this European Parliament regulation also applies to organisations outside the European Union (EU) that collect personal data from EU citizens. Hence, the regulation throws a broader net than the Euro-zone and practically covers all multinational companies in the world.

1.4.2 Cost of data leakages

Data leakages cost millions of dollars to companies around the world. For example, as per the Article 83 of the General Data Protection Regulation (The European Parliament and The European Council, 2016, p. 82), companies with severe data breaches will be fined either 2% of the company’s worldwide annual turnover or €10,000,000 which
ever is greater. In addition, a data leakage has many other associated losses including loss of customer confidence and legal fees and settlements related to multiple lawsuits. Table 1.2 presents a list of major data leakages between 2011 to 2015 where each single breach compromised more than 1 million unique customer records. Table 1.1 presents the ratio of the cost of the data leakage to the net asset value of that company, just to give an idea of the size of the risk to an organisation from data leakage. This shows that data security translates into a financial risk for an organisation.

<table>
<thead>
<tr>
<th>Organisation</th>
<th>Information lost</th>
<th>Records compromised (in millions)</th>
<th>Year</th>
<th>Net asset value of the company (in $M)</th>
<th>Cost of the data breach (in $M)</th>
<th>Risk ratio on net asset value of the company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthem Insurance</td>
<td>personal information</td>
<td>80</td>
<td>2015</td>
<td>24,251</td>
<td>8,000</td>
<td>32.99%</td>
</tr>
<tr>
<td>The Home Depot</td>
<td>personal and financial information</td>
<td>56</td>
<td>2014</td>
<td>12,520</td>
<td>8,120</td>
<td>64.85%</td>
</tr>
</tbody>
</table>

Table 1.1: Financial risk ratio against net asset value of two companies where more than 1 million records were compromised

---

The cost of a data leakage cannot be realized for sometime. In the case of The Home Depot data breach, the 2014 annual report of The Home Depot (2014) has revealed
Introduction

that the gross expense for their 2014 data breach stood at $63m. However, the report also stated that “We expect to incur significant legal and other professional services expenses associated with the Data Breach in future periods”. The 2015 annual report of The Home Depot (2015) has revealed that the gross expense related to its 2014 data breach had increased to $232m.
### Motivation

<table>
<thead>
<tr>
<th>Organisation</th>
<th>Information lost</th>
<th>Records compromised (in millions)</th>
<th>Year</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ashley Madison</td>
<td>personal info</td>
<td>33</td>
<td>2015</td>
<td>BBC News, 2015</td>
</tr>
<tr>
<td>TalkTalk</td>
<td>personal and financial info</td>
<td>4</td>
<td>2015</td>
<td>Gemalto, 2015</td>
</tr>
<tr>
<td>Anthem Insurance</td>
<td>personal info</td>
<td>80</td>
<td>2015</td>
<td>Lockton Inc., 2015</td>
</tr>
<tr>
<td>Adult FriendFinder</td>
<td>personal info</td>
<td>3.8</td>
<td>2015</td>
<td>Gemalto, 2015</td>
</tr>
<tr>
<td>Community Health System</td>
<td>personal info</td>
<td>4.5</td>
<td>2014</td>
<td>Verizon, 2015</td>
</tr>
<tr>
<td>eBay</td>
<td>personal info</td>
<td>145</td>
<td>2014</td>
<td>Verizon, 2015</td>
</tr>
<tr>
<td>Adobe</td>
<td>personal info</td>
<td>38</td>
<td>2013</td>
<td>BBC News, 2013</td>
</tr>
<tr>
<td>Target</td>
<td>personal and financial info</td>
<td>70</td>
<td>2013</td>
<td>The Association of Data Protection Officers, 2015</td>
</tr>
<tr>
<td>Evernote</td>
<td>personal info</td>
<td>50</td>
<td>2013</td>
<td>The Association of Data Protection Officers, 2015</td>
</tr>
<tr>
<td>Epsilon</td>
<td>personal info</td>
<td>60</td>
<td>2011</td>
<td>The Association of Data Protection Officers, 2015</td>
</tr>
<tr>
<td>Sony</td>
<td>personal info</td>
<td>78</td>
<td>2011</td>
<td>The Association of Data Protection Officers, 2015</td>
</tr>
</tbody>
</table>

Table 1.2: Data leakages between 2011-2015 where more than 1 million records were compromised
1 Introduction

1.5 Hypothesis

Banks (2012) proposed a formal framework that can be used for the formal analysis of
data leakage related confidentiality requirements in systems. Banks’s Confidentiality
Framework (BCF) can be used to demonstrate that the formal model of a system does
not leak data through legitimate communication channels of the system in violation
of any confidentiality requirements of the system. The manual application of BCF
is practically infeasible (Section 4.2). Further, there are no tools that can be used to
analyze system models using BCF. However, there are different tools and frameworks
that can be used to derive and analyse formal predicates from system models, similar
to BCF (Section 2.2).

The hypothesis

The hypothesis of this thesis is that a practically applicable approach exists that
supports the process of analysing system models using Banks’s Confidentiality
Framework (BCF) (Banks 2012) to verify if those models respect the integrated confi-
dentiality requirements pertaining to data leakage through legitimate channels.
1.5 Hypothesis

1.5.1 The challenges

In order to justify the hypothesis, the following issues were addressed.

• Identify the step-by-step process for applying BCF.

• Identify what is required for applying BCF such as:
  – how formal models of systems should be presented.
  – how confidentiality requirements should be encoded in system models.

• Identify how the process of applying BCF can be automated.

• Identify how the predicates generated through BCF can be translated to a format that can be simplified by a tool.

• Identify a tool that can simplify complex predicates.

• Identify a formal language that can be used to specify systems and confidentiality requirements in a way that the combined specification can be type checked with an existing tool.
1 Introduction

1.5.2 Testing the hypothesis

In order to test the hypothesis, the following tasks were required.

1. Constructing a tool-chain for analysing systems using BCF.

2. Using the tool-chain to analyse purpose written specifications based on typical system scenarios in which data leakage is a known issue.

3. Carrying out the following types of tests.

   a) Multiple tests that show that analysing a system using BCF confirms that the confidentiality requirements are respected in a given specification, if there were no contradictions in the system specification.

   b) A test that shows that BCF correctly flags an issue that is not apparent with a seemingly correct specification.

   c) A test that shows that BCF correctly flags an issue with a specification that has an artificially inserted inconsistency within its functionality and confidentiality requirements.

4. Comparing the time taken to analyse systems using the tool-chain and comparing it with manual application to show the value of the mechanisation.

5. Demonstrating the practical applicability of the mechanisation by showing that confidentiality requirements related to data leakage through legitimate channels can be generalised into patterns that supports tool based analysis.
1.6 Contributions

Major contributions of this thesis are:

1. Proposing a practical approach for using Banks’s Confidentiality Framework (BCF), to reason about the conformance of a system to a given set of data leakage related confidentiality requirements, by developing a mechanisation for BCF (contribution from Chapter 4).

2. Identifying and extracting generalized patterns of confidentiality requirements from literature, where these requirements are related to data leakage in systems through overt channels (contribution from Chapter 5).

3. Demonstrating the value of the proposed mechanisation by using it to analyse systems with data leakage related confidentiality requirements. This includes:

   a) Carrying out a comparison of execution times during the manual vs. mechanised application of BCF (contribution from Chapter 4).

   b) Reviewing the time consumed for the mechanised application of BCF on hand-crafted systems with varying degrees of complexity, with a confidentiality requirement that reflects a generalized pattern (contribution from Chapter 6).
1 Introduction

1.7 Thesis structure

The structure of the thesis is as follows.

Chapter 2 presents a discussion of the preliminary knowledge that is required to follow this thesis.

Chapter 3 presents the mechanisation of BCF. The chapter discusses ways in which a practically applicable approach can be developed for BCF.

Chapter 4 presents a discussion on the approach which has been followed in this research to evaluate the mechanisation of BCF.

Chapter 5 presents a systematic literature search for confidentiality related discussions in order to identify common recurring patterns of confidentiality requirements. Further, an approach for deriving generalized patterns of confidentiality requirements is also presented.

Chapter 6 presents a case study analysis where systems with confidentiality requirements, that represent instances of patterns of confidentiality requirements, has been analysed. A formal model has been developed for each system and the results of the mechanised analysis has been presented and compared.

Chapter 7 presents an evaluation of the research in this thesis. Further, this chapter discusses the future directions in which the work pursued in this thesis may be extended. The chapter further summarises the overall contributions, findings and limitations of the work presented in this thesis.
2 Background

2.1 Introduction

The research presented in this thesis establishes a practically applicable approach for the analysis of data leakage in systems with a confidentiality requirement. The approach is based on the mechanisation of BCF (Banks 2012). BCF is based on Unifying Theories of Programming (UTP) by Hoare and He (1998) and has been instantiated using the Circus notation (Oliveira et al. 2009). The denotational semantics of the Circus notation is based on UTP. This chapter provides an account of the preliminary knowledge which the user must be equipped with in order to understand the technical content in this thesis.

First, a discussion about the rationale behind selecting BCF for analysing data leakage related confidentiality requirements in systems is presented. This includes a comparison of BCF with research carried out by other researchers in the realm of formal analysis of confidentiality. Next, the chapter introduces the reader to UTP, discusses how the concept of BCF is captured in UTP, introduces the Circus notation and explains how UTP forms the basis for the notation, describes how systems are modelled using the Circus notation, describes how BCF is instantiated for Circus and discusses how BCF in Circus can be used to analyse data leakage related confidentiality requirements in systems.
2.2 Analysing systems with a confidentiality requirement

This section reviews some existing formal approaches for analysing systems with a confidentiality requirement. The Figure 2.1 shows the stage of the system development where each approach is utilised. These approaches will be discussed later in this section. The approaches discussed include PROMELA and the SPIN model checker (Holzmann, 1997), SecureUML (Basin et al., 2003), Secure Tropos (Bresciani et al., 2004), CoNaN tool (Cerny and Alur, 2009), InDico (Accorsi and Wonnemann, 2010), CONCHITA using KAOS (De Landtsheer and Van Lamsweerde, 2005), TEES framework (Howitt, 2008) and Confidentiality properties and the B Method (Onunkun, 2012).

In addition, the Figure 2.1 includes the techniques abuse case and UML-RT to Circus transformation. Even though these techniques are not used for analysing confidentiality in systems, abuse case is included in Figure 2.1 to show that there is a possible alternative to the existing KAOS technique used in CONCHITA, whereas UML-RT to Circus, which is discussed later in Section 2.5.2, is included to show the stage of the system development process where UML-RT to Circus is utilised.

This section presents a critical analysis of BCF by discussing the comparative advantages and limitations of BCF with respect to other proposed approaches shown in Figure 2.1.

**Abuse case.** McDermott and Fox (1999) define abuse case as a complete interaction between an actor and the system that is harmful to the system, one of the actors or to one of the stakeholders of the system. McDermott and Fox (1999) introduced the notion of abuse case to enable engineers to model possible harmful scenarios of system interactions in such a way that the artefact could be understood by both the user as well as the customer.

Abuse case and KAOS shown in Figure 2.1 are both goal-oriented requirement engineering approaches that can be used for deriving security requirements from security goals. However, they differ in how they derive and model security requirements.
2.2 Analysing systems with a confidentiality requirement

---

**Figure 2.1:** Approaches for eliciting, analysing, transforming and formally verifying confidentiality properties in systems
2 Background

- Knowledge Acquisition in automated Specification (KAOS) specification language contains formal definitions in temporal first-order logic for components of the system meta-model (Black, 2009, p. 22). These components include system goal, agents, actions, entities and the relationships between them. KAOS addresses security requirements by way of anti-goals (Lamsweerde, 2004).

- Abuse case uses the UML (Rumbaugh et al., 2004) approach for modelling possible harmful scenarios of system interactions.

> “In the requirement phase, the abuse case helps in gaining better understanding of system security between stakeholders, especially users and customers. Any security design trade-offs can be made easily” (Srivatanakul, 2005, p. 76).

PROMELA and the SPIN model checker. The Protocol/Process Meta Language (PROMELA) (Holzmann, 1997) is a language for modelling finite state machines. It allows a user to model concurrent processes. PROMELA supports embedded C code. Communication between the processes are established through channels defined for message passing. The Simple Promela INterpreter (SPIN) (Holzmann, 1997) analyses the logical consistency of concurrent systems. SPIN supports the PROMELA modelling language. A PROMELA model consists of a number of processes templates with at least one instantiation (Holzmann, 1997). The SPIN model checker translates each template into a finite automaton. The global system behaviour or state space of the system is represented by computing asynchronous interleaving of all the finite automaton. Security properties are modelled as claims in Linear Temporal Logic (LTL). SPIN converts the encoded security properties to a Büchi automaton (Büchi, 1960) and computes the synchronous product of this.

---

1 The Büchi automata extends the theory of finite automata on finite words to languages over infinite words. “While finite runs of finite automata are accepting if an accepting state is visited at the end of the run, an infinite run of a Büchi automaton is accepting if a final state is visited (or a final transition is taken) infinitely many times during the course of the run. The Büchi acceptance condition thus specifies a set of states (or transitions) that have to be visited (respectively, taken) infinitely often” (Varghese, 2014, p. 1).
2.2 Analysing systems with a confidentiality requirement

automaton and the automaton representing the global state space. The model checker visits every reachable state and also remembers all the states it has visited (Wang et al., 2000, p. 50). The memory requirement to store this information and resources required to traverse the state space depends on the size of the state and the number of reachable states and can be resource intensive depending on the given system specification. Edelkamp et al. (2001) combined SPIN with HSF (Edelkamp, 1999) heuristic search workbench to develop the LISP SPIN tool that can traverse large state spaces more efficiently than SPIN.

The model checker either validates the given security property or else a counter example is produced that shows how the violation can take place. Some other researchers have used SPIN to model and analyse confidentiality properties of systems as described below.

Dabaghchian and Abdollahi Azgomi (2015) used embedded C code in PROMELA to model a variant of the observational determinism confidentiality property. Observation determinism ensures confidentiality in concurrent programs by assuring that public variables and private variables are kept independent during the execution of a program.

Ahmed and Tripathi (2003) used SPIN to model and verify confidentiality properties of a Computer Supported Cooperative Work (CSCW) system. They re-modelled the course activity case study by Foley and Jacob (1995).

In this thesis, the same system has been modelled and analysed using BCF (Banks, 2012).

Having to remember every visited state is the biggest limitation of SPIN. Further, the specifications that can be written using PROMELA are restricted to the available PROMELA templates. However, BCF is defined using the UTP framework and can be instantiated for a range of UTP theories as shown by Banks (2012). Therefore a broad set of features can be supported by instantiating BCF on relevant UTP theories than in SPIN+PROMELA.
2 Background

SecureUML - Integrating UML based process modelling and RBAC. Basin et al. (2003) proposed an approach to integrate RBAC based security policies into formal system models by using OCL constraints in UML. The resulting UML based language is called SecureUML. The RBAC policies imposed by OCL constructs are used to derive the precondition for every action in a UML class. Further, they integrate SecureUML with a process modelling language by defining a dialect that identifies process elements as protected resources. The semantics of this dialect uses a form of weakest pre-condition computation to decide whether a user is allowed to execute a particular process element. The process models are translated to implementations of controller objects for multi-tier applications. They evaluate their approach by generating prototypes using an extended version of a tool called ArcStyler (Basin et al., 2003, p. 8).

SecureUML integrates security into formal models of systems. However they do not provide an approach for verifying those models. The fundamental objective of BCF is to provide an approach for verifying confidentiality integrated formal models of systems.

Secure Tropos - Secure requirements engineering with reasoning. Tropos (Bresciani et al., 2004) is an agent based software engineering methodology. Tropos lacks the ability to capture the functionality and security features of an organisation at the same time. Giorgini et al. (2004) extended Tropos by defining predicates that represent properties of actors and trust properties between actors. These predicates are combined to form security properties in the form of axioms. These axioms are formalized as a datalog logic program and analysed using the DLV system (Leone et al., 2006). Consistency checks guarantee that the security specification is not self contradictory (Giorgini et al., 2004).

---

2 ArcStyler “provides a transformation function for converting UML classes and state machines into controller classes for web applications, executed in a Java Servlet environment” (Basin et al., 2003, p. 8).

3 Abiteboul et al. (1995, p. 273) give an introduction to the datalog language.
2.2 Analysing systems with a confidentiality requirement

Secure Tropos utilizes a graphic model of a system whereas BCF utilizes a formal specification based model of a system. It is not clear how a formal specification of a Secure Tropos based model can be derived so that the specification can be refined to an eventual implementation. However, with certain instantiations of BCF, this is possible.

**ConAn tool - Automated confidentiality analysis.**  Automated program checking for confidentiality by [Cerny and Alur (2009)](http://example.com) presents a notion of confidentiality based on possibilistic reasoning. The tool supports a subset of Java that includes boolean variables, integer variables, data variables and variables that range over an infinite domain. However, only equality tests on data variables are supported.

A program execution produces an observation with respect to a condition \( \text{cond} \). If two executions exist where, in one execution a confidentiality predicate \( \text{secret} \) holds and in another execution the same confidentiality predicate \( \text{secret} \) does not hold, then the two executions will be indistinguishable to the observer, hence achieving confidentiality.

The proposed ConAn tool takes in a program in Java bytecode, a condition \( \text{cond} \), a confidentiality predicate \( \text{secret} \) and some other required parameters. The tool automatically inserts annotations to the Java program that record user observations based on program execution. The tool generates formal scripts compatible with the Yices SMT solver ([Dutertre 2014](http://example.com)). The Yices SMT solver decides satisfiability of the resulting formulae.

Automated program checking for confidentiality related to data leakage is specific to a subset of Java constructs whereas BCF is defined at a higher level since BCF is based on a formal notation and not specific to any programming language.

The approach for verification does not start from a formal specification but rather reverse engineers a program to derive a set of formal scripts which then get evaluated. Data leakage in Circus models can be verified using BCF and subsequently those models can be refined to be implemented in any supported
programming language. The ConAn tool based approach is limited to restricted Java based implementations of systems, whereas since BCF supports verification of system models at the formal specification level, this limitation does not exist.

InDico - Automated analysis of business processes for confidentiality. Accorsi and Wonnemann (2010) proposed an approach for information flow analysis in business processes. First, a formal model of a business process is developed. For example, the Business Process Model and Notation (BPMN) (White, 2004) can be used for this. The model is mapped to a coloured Petri Net (Jensen, 1987). Next, the activities of the resulting IFnet's are labelled with security labels. Analysis of the IFnet relies on a manual verification process. The first step is to inspect the IFnet for structured patterns that might leak information. The second step is to check whether the identified patterns are reachable within a run of the system. This is a state space explosion problem when processes have non-trivial number of states. Therefore, the approach is not practical for non-trivial processes and also has scalability issues due to a manual analysis approach. Further, this approach supports only non-interference properties.

In comparison to InDico, BCF is not limited by state explosion and scalability because BCF validates a system based on predicates which can be machine evaluated.

CONCHITA framework. The CONfidentiality CHecker for Incremental Threat Analysis (CONCHITA) by De Landtsheer and Van Lamsweerde (2005) is a reasoning framework based on bounded model checking and constraint solving techniques. Formal system models are defined using the goal oriented software requirements engineering methodology called KAOS (Van Lamsweerde et al., 1991). The knowledge of agents at specific states in the system are captured using epistemic constructs. These epistemic constructs are combined with LTL (Linear Temporal Logic) (Pnueli, 1977) to define axioms that capture the interaction between

---

4 An IFnet is a specialization of coloured petri nets for work-flow modelling and information flow analysis (Accorsi and Wonnemann, 2010, p. 195).
knowledge and time. A set of templates that represent specification patterns for confidentiality requirements are given. Specifiers must utilize one of these templates to define their confidentiality claims. CONCHITA propositionalizes the input specification and confidentiality claims. The resulting propositions are converted into non-temporal logic versions by defining the length of the trace as a given bound. The confidentiality verification problem is translated to a constraint satisfaction problem by quantifying confidentiality patterns over the user agent’s finite bound of knowledge of the system. The resulting constraint is analysed using the Oz constraint solver (Henz et al., 1993) which presents a counter example if the knowledge of a user agent contradicts with the quantified confidentiality claim.

CONCHITA depends on a limited set of templates that can be used to define confidentiality requirements whereas BCF does not have this limitation. Further, BCF uses the Circus notation that represents system entities and state variables more closely than epistemic logic used in CONCHITA.

TEES confidentiality model. The TEES confidentiality model (Howitt, 2008) is an authorization model that utilizes both Role-Based Access Control (RBAC) (Ferraiolo and Kuhn, 1992) and Identity-Based Access Control (IBAC) that includes override. The model is based on a state machine approach and is formalized using the B notation and verified using the Rodin B-toolkit (Butler and Hallerstede, 2007). TEES concentrates on the antecedent of a conditional confidentiality requirement where TEES defines the condition under which the confidentiality of a state variable must be maintained. However, BCF is more flexible in that BCF can additionally define the degree of confidentiality of a state variable. For example, consider a set S that represents the set of bids in an auction. BCF can be used to define a confidentiality property where either all values of S can be kept secret or a defined subset of values of S can be kept secret.

---

5 A conditional confidentiality requirement is equivalent to a conditional information flow, a flow of information that only occurs when some condition is met at runtime (Tschantz and Wing, 2008, p. 108).
2 Background

Confidentiality Properties and the B Method. Onunkun (2012) proposes a framework to reason about information flow security in B machines. The framework is restricted to Multi Level Security properties. In this framework, state variables are mapped to security classes in a security flow lattice. The framework extends the flow logic analysis approach proposed by Clark et al. (2002). Reasoning about the information flow is based on predicates computed based on this extension. Even though the B machines were checked with Atelier-B (2017) for consistency, it is not clear from Onunkun (2012) as to how the described information flow violations were detected using the custom tool which Onunkun (2012) had developed.

BCF detects information flow violations through inconsistencies in a computed predicate similar to that in Onunkun’s method. However, BCF is not restricted to Multi Level Security properties only.

As can be seen from Figure 2.1, the approaches based on Secure Tropos and SecureUML are utilized in the requirement analysis phase. However, there is little research on how the requirements phase is reconciled with the formal specification phase (Nakagawa et al., 2007, p. 531). The work by Nakagawa et al. (2007) is an attempt towards bridging this gap by proposing an approach to generate formal specifications based on VDM++ (Durr and van Katwijk, 1992) from KAOS models.

None of the above approaches support the analysis of a concrete formal specification of a system where confidentiality requirements can be modelled using constraints on the channels through which a system communicates with its environment. However, the instantiation of BCF (Section 2.6) discussed in thesis supports supports such constraints. The features in Table 2.1 are considered as critical criteria based on which BCF has been selected for analysing systems with a confidentiality requirement.

Table 2.1 identifies at least one limitation of each approach in comparison to the BCF approach for analysing data leakage related confidentiality requirements in systems.
2.3 Banks’s confidentiality framework (BCF)

Banks (2012) introduced BCF for the formal analysis of confidentiality requirements in systems related to data leakage. BCF uses the general Unifying Theory of Programming (UTP), and BCF has been instantiated (but not automated) for the Circus notation (Oliveira et al., 2009). Hereafter, the phrase “BCF in Circus” will be used when referring to this instantiation. BCF combines functionality and confidentiality requirements within a formal framework to allow the specification of systems that are secure-by-design (Banks and Jacob, 2011).

<table>
<thead>
<tr>
<th>Feature</th>
<th>BCF</th>
<th>PROMELA</th>
<th>SecureUML</th>
<th>Secure Tropos</th>
<th>CoNaN</th>
<th>InDico</th>
<th>CONCHITA</th>
<th>TIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not restricted by templates</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supports formal analysis of confidentiality properties in system models</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not limited by state explosion problem</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not restricted by Multi-Level Security properties</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Features supported by the various approaches for analysing confidentiality requirements

A tick (✓) denotes that the particular feature is available in the related formal approach. A cross (×) denotes that the particular feature is not available in the related formal approach. A blank denotes that the author has not verified the availability of the particular feature in the related formal approach. It is not necessary to identify all the features supported by each approach as the intention of this table is to show that none of the approaches except BCF support all the features listed in the table.
2 Background

As per Kerckhoffs’ Principle, ‘security through obscurity’ is considered bad practice in the design of a system (Kerckhoffs, 1883). In line with Kerckhoffs’ Principle, BCF assumes that the adversary who wants to learn confidential information from the system may have knowledge about the complete source code of the system.

Section 2.4 introduces UTP, describes the concept of BCF in UTP and discusses some of the challenges that arise in automating a framework based on UTP. Section 2.5 gives a brief introduction to the Circus notation, discusses how BCF has been instantiated for the Circus notation and outlines some of the advantages of using Circus models in detecting data leakage in systems.

2.3.1 The conceptual basis for BCF

The development of information flow security theories has been a gradual process. Denning (1976) proposed a lattice model of secure information flow and provided a formal semantics for the formal verification of information flow in program specifications. Other notable information flow properties discussed in the literature include Goguen and Meseguer’s Non-Interference (Goguen and Meseguer, 1982), Sutherland’s Non-Deducibility (Sutherland, 1986), Jacob’s Inference function (Jacob, 1988), McCullough’s Generalized Non-Interference (McCullough, 1988), O’Halloran’s Non-inference (O’Halloran, 1990), McLean’s Generalized Non-inference (McLean, 1994) and Zakinthinos’s Perfect Security Property (Zakinthinos and Lee, 1997).

BCF is loosely based on the user inference function proposed by Jacob (1988). Further, BCF adapts the idea of Morgan’s shadow semantics (Morgan, 2009). Both Jacob’s inference function and Morgan’s shadow semantics are described below.

**Jacob’s User Inference.** Sometimes a system might only require hiding the occurrences of certain properties of high$^6$ events, rather than all high events and the knowledge about the occurrence of other high events are considered acceptable. To cater for

---

$^6$ Events executed by a user in a higher security level than an ordinary user of the system.
such scenarios, there was a need for a formal approach to calculate how much a
group of users can infer about the events executed by another group of users in a
system.

Jacob (1988) proposed the inference function to carry out such calculations. The
inference function calculates how much a user \(U\) can infer about the interactions
of other users in a system \(S\), by carrying out an interaction allowed for the user \(U\).
This is an observation based approach where the set of all possible observations
which the user \(U\) can make by interacting with \(S\) is syntactically written as \(S@U\)
and is called the projection of \(U\) onto \(S\) (Jacob, 1988, p. 15). The set of all traces in
the system \(S\) is denoted by \(\tau S\). For any trace \(t\), where \(t \in \tau S\), \(U\) can only observe
parts of \(t\) that is visible to \(U\) through its window which is defined as \(t \upharpoonright U\). The
projection of \(U\) onto \(S\) is defined as follows.

\[
S@U \triangleq \{ t \upharpoonright U \mid t \in \tau S \}
\]

After observing the output \(X\) of an interaction \(l\) with the system \(S\), the user \(U\) can
calculate all possible traces which could have generated the observation \(X\). This is
a general approach suitable for calculating the amount of knowledge a user can
obtain by interacting with the system.

Morgan’s shadow semantics. Classical refinement does not preserve ignorance related
properties. This phenomenon commonly known as the refinement paradox was first identified by Jacob (1988). Morgan’s approach (Morgan, 2009) consists both in resolving the refinement paradox as well as maintaining the uncertainty during the refinement of sequential programs.

A system state might have a visible as well as a hidden state space based on
the security policy requirements of the system. A variable \(h\) whose observations
are to be kept secret is part of the hidden state space. In addition, Morgan

---

7 An observer’s ignorance of data is his/her uncertainty about the parts of the program state which he/she cannot see (Morgan, 2009, p. 2).
introduces a third (set valued) variable to the hidden state space called the *shadow*, which consists of all potential values of the variable $h$ in the hidden state space, including the current value of $h$. Consider the scenario where an adversary has the complete knowledge of the source code, can view the visible part of system state (based on his authorisation) as well as have the knowledge of the program flow. However because of the uncertainty about the hidden state space due to atomic non-determinism (introduced by the *shadow*), the adversary cannot deduce for sure, the exact value of the variable $h$ in the hidden state space.

Banks (2012) borrowed the concept of *shadow* variables from Morgan (2009). In BCF, the concept of *shadow* variables is established by introducing the concept of a *shadow* system, where the variables in the *shadow* system reflects the same from the original system, under certain user observations. BCF in *Circus* also borrows the concept of the inference function from Jacob (1988). BCF in *Circus* provides an inference function that can be used to calculate the possible observations a user can make about the state space of a system by his/her interactions. BCF in *Circus* assumes that the calculated observation of the state space of a system is always a reflection of the state space of the associated *shadow* system.

### 2.3.2 Advantages and limitations of BCF

The following are some of the advantages of BCF.

**BCF being based on UTP.** Since BCF is based on UTP, the underlying concept can be specialised for other UTP theories. Banks described how BCF can be specialised for the *theory of designs* (Banks 2012, p. 35) and the *theory of reactive processes* (Banks 2012, p. 40). Banks has also instantiated BCF using the *Circus* notation as stated before in page 53. In this thesis, this instantiation has been used in analysing data leakage in systems with a confidentiality requirement.

**BCF based analysis involves a comprehensive search of the state space.** BCF uses logical expressions to capture the knowledge about the entire state space of a
2.4 Unifying Theories of Programming (UTP)

BCF is a generic formal approach defined in Unifying Theories of Programming (UTP). UTP is a unified framework for formal program semantics, which can be used to formally specify systems (Hoare and He, 1998). UTP is a mathematical treatment of computer programming using the simple non-deterministic programming language introduced by Dijkstra (1975).

In the UTP semantics, all the possible observations of executing a program are captured in a predicate (Hoare and He, 1998). Each variable in the predicate models an attribute of the system. The following example describes a universal observation of a system where an attribute $x$ of the system is defined to be greater than 0 and another attribute

program at any given state.

The following are some of the limitations of BCF.

**Weak notion of confidentiality.** BCF reasons about the existence of at least one alternative explanation when hiding the value of a state variable. This is a weak notion of confidentiality when compared to Caroll Morgan’s shadow semantics (Morgan, 2009). In the shadow semantics, a set of other possible alternative explanations exist for each hidden value.

**Tedious, time consuming and error-prone calculations required in BCF.** The calculations required to analyse a system using BCF is tedious, time consuming and error-prone since the probability of human errors exists in any activity related to manually manipulating large blocks of formulae text (Verma, 2011, p. 11).

**Requirements must be written specific to a given system.** When using BCF, confidentiality requirements must be written specific to a given system and cannot be expressed in a more abstract or generalized manner.
2 Background

\(y\) is defined to be greater than \(x\).

\[ x > 0 \land y > x \tag{2.1} \]

Valid observations of the system may include \((x = 5 \ \& \ \ y = 6)\) or \((x = 12 \ \& \ \ y = 13)\). However, as per the predicate in Equation (2.1), observations such as \((x = 0 \ \& \ \ y = 1)\) or \((x = 3 \ \& \ \ y = 2)\) are not valid. Such observations cannot exist in practice.

2.4.1 UTP theories

A unified theory in UTP is comprised of alphabets, signatures and healthiness conditions. In addition to the observational variables in a system, a UTP theory may contain other variables which are also called auxiliary variables (Foster and Payne, 2013; Oliveira et al., 2009; Verma, 2011) that record certain properties about the execution of a program. The alphabet of a program conforming to such a UTP theory includes these auxiliary variables. A signature of a UTP theory for a programming language consists of a set of operators and atomic components that represent the constructs of a programming language (Hoare and He, 1998, p. 12). A set of basic signatures common to a range of programming language theories are listed in Table 2.2. Healthiness conditions are idempotent functions that define the subset of predicates that satisfy a given UTP theory. The theory of relations (Hoare and He, 1998), the theory of designs (Hoare and He, 1998), the theory of reactive processes (Wei, 2013, p. 3) and the theory of reactive designs (Cavalcanti and Woodcock, 2006) are UTP theories that form the foundation of the Circus theory (Oliveira et al., 2009), the semantic basis for the Circus notation. The build up to the theory of Circus from the theory of relations is shown in Figure 2.2. The following is a brief introduction to each of those theories.

---

8 Healthiness conditions of a UTP theory are higher order predicates, that select only some predicates as meaningful in that particular UTP theory. Healthiness conditions are often written as fixed point equations such as \(P = H(P)\), where \(P\) represents a predicate and \(H\) is a predicate transformer.

9 "An idempotent function is a function \(f\), where \(f(f(x)) = f(x)\) for all \(x\)" (Gormish et al., 1997, p. 8).
### 2.4 Unifying Theories of Programming (UTP)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assignment</strong></td>
<td>$a := e$</td>
<td>$a' = e \land x' = x$</td>
<td>state variable $a$ is assigned the value of $e$ and all other variables (denoted by $x$) remain unchanged</td>
</tr>
<tr>
<td><strong>Sequential composition</strong></td>
<td>$P ; Q$</td>
<td>$\exists x_0 \cdot P[x_0/x'] \land Q[x_0/x]$</td>
<td>sequential composition between the programs $P$ and $Q$ is the relational composition of their intermediate state</td>
</tr>
<tr>
<td><strong>Conditional</strong></td>
<td>$P \triangleright b \triangleright Q$</td>
<td>$(b \land P) \lor (\neg b \land Q)$</td>
<td>a program that behaves like $P$ if $b$ is true, or like $Q$ if $b$ is false. $b$ is a truth function without dashed variables.</td>
</tr>
<tr>
<td><strong>Skip</strong></td>
<td>$\Pi$</td>
<td>$x' = x$</td>
<td>does not change the program state in any way and all state variables (denoted by $x$) remain unchanged</td>
</tr>
<tr>
<td><strong>Non-determinism</strong></td>
<td>$P \sqcap Q$</td>
<td>$P \lor Q$</td>
<td>the greatest lower bound between the programs $P$ and $Q$</td>
</tr>
</tbody>
</table>

| Table 2.2: A set of basic signatures common to a range of programming language theories |
2 Background

Theory of relations. In the UTP theory of relations (Hoare and He, 1998), every programming construct is formalised as a relation between an initial observation and an intermediate or a final observation of a system. Any variable in a program predicate can also be decorated with a prime (‘) to denote the value of the same variable immediately after the execution of the program. The undecorated counterpart represents the value of the observation before the execution of the program. The observations made before and after the execution of the program can be combined to form a UTP relation which describes the relationship between the initial and intermediate or final observation of the system.

For example, assume that the initial observation of the system in Equation (2.1) is such that \( x = 3 \) and \( y = 16 \). Now, run the program statement:

\[
x := x \times 5
\]

(2.2)

The following UTP relation represents the initial and final observation resulting from executing the program statement in Equation (2.2):

\[
x = 3 \land x' = 15 \land y = 16 \land y' = 16
\]

The theory of relations does not distinguish between terminating and non-terminating programs (Cavalcanti et al., 2006, p. 210).

Theory of designs. A program statement written with preconditions and postconditions and which respects certain healthiness conditions is called a design. The theory of designs introduces two observation variables where:

- \( \text{ok} \) indicates that the program has started.
- \( \text{ok}' \) indicates that the program has terminated.

The subset of relations that conform to the theory of designs (Hoare and He, 1998) must respect the healthiness conditions \( \text{H1} \) and \( \text{H2} \) (Hoare and He, 1998, p. 281).
2.4 Unifying Theories of Programming (UTP)

\[ H_1(P) \equiv ok \Rightarrow P \]
\[ H_2(P) \equiv P ; ((ok \Rightarrow ok') \land v' = v) \]

where \( P \) is a relation with the alphabet \{ok, ok', v, v'\} and \( v \) represents all state variables of the system except \( ok \) and \( ok' \). If \( P \) is \( H_1 \) healthy then any observation on \( P \) can only be made after the program has started. If \( P \) is \( H_2 \) healthy then the program must always terminate.

**Theory of reactive processes.** A reactive process is a program which may engage with its environment and whose behaviour might depend on these interactions (Wei, 2013, p. 3). In addition to \( ok \) and \( ok' \), the theory of reactive processes introduces the variables \( wait, tr \) and \( ref \) and their primed counterparts where:

- if \( wait' \) is **True** then \( P \) is in an intermediate state.
- if \( wait' \) is **False** then \( P \) has successfully terminated.
- if \( ok' \) is **True** then the state of \( P \) depends on \( wait' \).
- if \( ok' \) is **False** then \( P \) has diverged.
- \( tr \) - sequence of events that have occurred up to the start of \( P \).
- \( tr' \) - sequence of events that have occurred after the start of \( P \), to the point when the subsequent observation is made.
- \( ref \) - events that \( P \) may have refused to participate in, up to the start of \( P \).
- \( ref' \) - events that \( P \) may have refused to participate in after the start of \( P \), to the point when the subsequent observation is made.

where \( P \) is a process. The undashed variables \( ok \) and \( wait \) represents the similar states of the predecessor of \( P \).

A reactive process \( P \) must satisfy the healthiness conditions \( R_1, R_2 \) and \( R_3 \) where
2 Background

UTP theories

Healthiness conditions of UTP theories

<table>
<thead>
<tr>
<th>syntax</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z notation</td>
<td>as UTP relations</td>
</tr>
<tr>
<td>CSP notation</td>
<td></td>
</tr>
<tr>
<td>Dijkstra’s guarded</td>
<td></td>
</tr>
<tr>
<td>commands</td>
<td></td>
</tr>
<tr>
<td>Morgan’s specification statements</td>
<td></td>
</tr>
</tbody>
</table>

The theory of Circus

Figure 2.2: UTP theories, healthiness conditions and the theory of Circus
R1 states that $P$ must never change history, R2 states that $tr$ has no influence on the behaviour of $P$ and R3 states that $P$ has no effect on the observation before it starts execution (Wei, 2013, p. 4). A reactive process is a fixed point on the function composition $R$ where:

$$R(P) \triangleq R1 \circ R2 \circ R3(P)$$

**Theory of reactive designs.** The theory of reactive designs (Oliveira et al., 2009) combines the valuable qualities of both the theory of designs (Hoare and He, 1998) and the theory of reactive processes (Cavalcanti and Woodcock, 2006, p. 240) where:

the ability to model programs in terms of preconditions and postconditions in the theory of designs

is combined with,

the ability to capture the intermediate behaviour of programs in the theory of reactive processes.

“The space of reactive designs is a sub-space of the reactive processes, which is derived by applying $R$ to the space of designs” (Banks, 2012, p. 14).

**Theory of CSP.** Hoare and He (1998) and Cavalcanti and Woodcock (2006) extend the theory of reactive designs to give a UTP semantics to CSP processes (Banks, 2012, p. 15). The space of CSP processes is a sub-space of the space of reactive designs that also satisfies the healthiness conditions CSP1 and CSP2. CSP1 states that if a given process $P$ diverges, then the only guarantee is the extension of the trace. CSP2 states that a given process $P$ cannot require non-termination.

Even though each of the above theories has a different syntax, the semantics of all those theories are based on UTP. An important advantage of a formalised program semantics such as UTP is being able to formally reason about a program. How can one ensure
that the developed program is consistent against its specification? This can be answered using program correctness.

### 2.4.2 Program correctness

A program is correct if every observation made of every possible run of the program results in values that satisfy the original specification (Hoare, 1997, p. 7). For example, consider the specification $S$ and a possible observation $Q$ made from running a program $P_1$ that correctly implements $S$.

\[
S \triangleq (l' < m' \land l > 0 \land m' < 100)
\]

\[
Q \triangleq (l = 5 \land l' = l \times 5 \land m' = 90 \land m = 90)
\]

To formalise the notation that the observation $Q$ satisfies the specification $S$, we state that the specification is implied by the observation.

\[
(l = 5 \land l' = l \times 5 \land m' = 90 \land m = 90) \Rightarrow (l' < m' \land l > 0 \land m' < 100)
\]

Further, program correctness states that this implication is **true** for all possible values of the observable variables, if $P_1$ correctly implements $S$.

\[
\forall l, m, l', m' \bullet ((l = 5 \land l' = l \times 5 \land m' = 90 \land m = 90) \Rightarrow (l' < m' \land l > 0 \land m' < 100))
\]

This universal quantification can also be written as $[Q \Rightarrow S]$, using the conventional square brackets by Dijkstra and Scholten (1990). The square brackets is an abbreviation for the universal closure of the implication over all the variables in the alphabet.

\[
[(l = 5 \land l' = l \times 5 \land m' = 90 \land m = 90) \Rightarrow (l' < m' \land l > 0 \land m' < 100)]
\]

The universal quantification of $Q \Rightarrow S$ must be **true** if $P_1$ is a correct implementation of
2.4 Unifying Theories of Programming (UTP)

S. In this case:

\[ \forall l, m, l', m' \quad ((l = 5 \land l' = l \times 5 \land m' = 90 \land m = 90) \quad \Rightarrow (l' < m' \land l > 0 \land m' < 100)) \]

\[ \equiv \]

(Leibniz)

\[ \forall l, m, l', m' \quad ((l = 5 \land l' = l \times 5 \land m' = 90 \land m = 90) \quad \Rightarrow (25 < 90 \land 5 > 0 \land 90 < 100)) \]

\[ \equiv \]

(simplify)

true

The above calculation shows that \( P_1 \) is a correct implementation of \( S \). Now, consider an observation \( R \) that is made from running another program \( P_2 \).

\[ R \equiv (l = 5 \land l' = l \times 20 \land m' = 90 \land m = 90) \]

In this case:

\[ \forall l, m, l', m' \quad ((l = 5 \land l' = l \times 20 \land m' = 90 \land m = 90) \quad \Rightarrow (l' < m' \land l > 0 \land m' < 100)) \]

\[ \equiv \]

(Leibniz)

\[ \forall l, m, l', m' \quad ((l = 5 \land l' = l \times 20 \land m' = 90 \land m = 90) \quad \Rightarrow (125 < 90 \land 5 > 0 \land 90 < 100)) \]

\[ \equiv \]

(simplify)

false

The above calculation shows that \( P_2 \) is not a correct implementation of \( S \).

Different theories have been proposed to facilitate reasoning about the correctness of programs. Next, a discussion of such theories are presented.
2.4.2.a Theories of program correctness

Formal program verification is proving properties of a program using logic and mathematics. Hoare logic (Hoare, 1969) and Dijkstra’s weakest precondition (Dijkstra, 1975) are two well-known calculi for the formal verification of programs. BCF utilises a form of weakest precondition computation to reason about confidentiality in systems.

**Hoare logic.** Hoare logic (Hoare, 1969) is a calculus that gives a set of inference rules and axioms to reason about program correctness. A precondition is a boolean expression that must be satisfied before the execution of the program. A postcondition is a boolean expression that must be satisfied immediately after a program has completed its execution. A precondition and postcondition assert a subset of acceptable states on the initial and final states of a program. Hoare uses such assertions to reason about certain programming constructs in UTP. If a program Q starts in a state that satisfies P and completes its execution then the program will reach a state that satisfies R.

\[ P \{Q\} R \]  

(2.3)

This structure is called the Hoare triple (Hoare, 1969, p. 577).

**Weakest pre-condition.** Dijkstra (1975) used Hoare style assertions (Hoare, 1969) to define a different construct for program development called the weakest precondition.

The weakest precondition function \( \text{wp} \) is a predicate transformer that maps a post condition Q of a program statement S to a precondition P. Dijkstra (1975) states that if the program S is executed with its initial state satisfying P, the program is guaranteed to reach a state satisfying the post condition Q.

\[ P = \text{wp}(S, Q) \]
Figure 2.3 shows the post condition and the resulting weakest precondition of an ePurse payment operation where $e\text{PurseBalance}$ is the existing balance in the ePurse and $\text{cost}$ is the amount to be deducted from the ePurse for the sale of items. In this case $wp(S, Q) \equiv (e\text{PurseBalance} - \text{cost}) > 0$, or $e\text{PurseBalance} > \text{cost}$.

![Figure 2.3: Weakest pre-condition and post condition of an ePurse payment operation](image)

2.4.3 Refinement

Refinement is the verifiable transformation of an abstract specification of a system to a more concrete one. If an abstract specification $R$ is refined by a more concrete specification $S$, the statement is formally written as $R \subseteq S$. Both of these specifications as well as the relation between the two must be formally defined in order to prove that the concrete specification is a “correct realisation” of the abstract specification (Potter et al., 1996a). Program refinement is the process of applying such a set of correctness-preserving transformations on an abstract specification eventually to produce executable code (Back and Wright, 1998, p. 20).

**Refinement Calculus.** A Refinement Calculus is a framework for reasoning about the correct derivation of programs using refinement. One approach for deriving such correct programs is through stepwise refinement (Wirth, 1971). Back (1980) used Dijkstra’s weakest precondition calculus (see Section 2.4.2.a) as a basis when formalizing stepwise refinement in a refinement calculus. The refinement calculus proposed by Morgan (1998) is an alternative approach where specifications and executable code are regarded equally as programs. A third approach has been proposed by Morris (1987). The approach by Morris (1987) is directly based on that of Back (Cavalcanti et al., 1998, p. 1).
2 Background

2.4.4 BCF in UTP

BCF allows a user to validate the consistency of the requirements in a system by comparing the state space of the original system \( P \) and an isomorphic state space of a possible copy of the system \( \tilde{P} \). From now on, this possible copy of the system, having the possible copy of the state space, will be called the twin system and its state space the twin state space. Banks called the twin state space the fog space (Banks and Jacob, 2014, p. 4). However, the word twin brings clarity to the concept of the fog space by emphasizing that BCF assumes that there is exactly one alternative copy of the state space and that the composition of the fog space is similar to that of the original system state space. This composition implies that:

\[
\text{if} \quad \text{a user is uncertain about the value of a state variable in the original system state space based on his/her knowledge about the program counter}
\]

\[
\text{then} \quad \text{the user will also be uncertain about the value of its twin counterpart in the twin state space}
\]

The user’s knowledge about the program counter depends on what functions the user is allowed to perform using the system. This in turn is defined by the functional requirement of the system. Retrospectively, a confidentiality requirement defines what a user must not learn from his/her interactions with a system. To achieve this, it must be made sure that the user is uncertain about the value of the particular state variable which the confidentiality requirement demands to be hidden or concealed. This uncertainty can be achieved by restricting the user’s knowledge about the program counter at the state where the value of that particular variable needs to be secured. To achieve this restriction, a separation between the two state spaces must be enforced and this can be specified by defining a predicate that uses variables from both the original as well as the twin system.
2.4 Unifying Theories of Programming (UTP)

2.4.5 Possible twin state space

Identifying the state space of a system and subsequently introducing a possible twin copy of the state space of the system, reflecting a possible copy of the system are pre-requisites for using BCF. Banks uniquely identified the variables of the twin system using the tilde decoration. In his thesis (Banks, 2012, p. 56), the twin variable for every variable \( x \) is written as \( \tilde{x} \).

The twin system always exists in tandem with the original system. The predicate that represents the combined state space of the original system and the twin system is derived using the predicate transformer \( U \) (Banks, 2012, p. 56). The predicate transformer \( U \) is defined as:

\[
U(S) \equiv S \land \tilde{S}
\]

where \( S \) represents a relational predicate that defines a system, \( \tilde{S} \) denotes \( S[\tilde{x},\tilde{x}' / x,x'] \), that is, \( S \) with each variable \( x \) and \( x' \) systematically renamed to \( \tilde{x} \) and \( \tilde{x}' \). Every state variable in the original system \( S \) has a twin variable in the twin system \( \tilde{S} \). The combined state space represented by the relation \( S \land \tilde{S} \) is called a lifted relation (Banks, 2012, p. 56).

For example, consider the relational predicate \( S \) that represents a system.

\[
S \equiv x > 3 \land y \leq 100
\]

The combined predicate that represents the system \( S \) in the lifted state space is:

\[
U(S) \\
\equiv S \land \tilde{S} \\
\equiv x > 3 \land y \leq 100 \land \tilde{x} > 3 \land \tilde{y} \leq 100
\]
2 Background

where \( \tilde{x} \) and \( \tilde{y} \) are *twin* state space variables introduced by renaming the alphabetised relational state variables \( x \) and \( y \) in the relational predicate \( S \). The variable \( \tilde{x} \) is a copy of \( x \) with the same value domain\(^{10}\). Likewise, \( \tilde{y} \) is a copy of \( y \) having the same value domains.

\(^{10}\) "A value domain is defined as the permissible values for a data element" (American National Standard Institute [ANSI], 1999, P. 27). Here, the term ‘value domain’ is used to refer to the set of values a variable might assume throughout all the stable states of a system.
2.5 Circus: a formal specification language

Banks (2012, p. 85) illustrates BCF by instantiating BCF using the Circus notation (Freitas 2005; Oliveira et al. 2009; Woodcock and Cavalcanti 2002). Circus is a formalism that combines a state based formalism called the Z notation (Spivey 1989) and a process-oriented formalism called CSP (Roscoe 1995; Schneider 1999) using the underlying UTP semantics of the languages. In addition, Circus uses Dijkstra’s guarded command notation (Dijkstra 1997) and Morgan’s refinement calculus (Morgan 1998). Since Circus combines both Z and CSP (Woodcock and Cavalcanti 2002), Circus utilizes the syntactic structures from both the Z and the CSP notations.

2.5.1 Advantages of Circus

The state of a Circus process is hidden except for communications through channels through which values of specified state variables may be observed (Cavalcanti and Gaudel 2014, p. 416). Data security policies may be captured in a Circus specification by introducing constraints on the data that can be communicated through defined channels. Further, Circus supports complex data structures (Mahony and Dong 1998). The Circus notation has a strong formal semantics that is based on UTP, a framework that unifies programming theories across many different computational paradigms (Oliveira et al. 2005, p. 1). The refinement strategy for Circus by Sampaio et al. (2003) allow the stepwise refinement of Circus specifications to code in a calculational way. The underlying relational model of the Circus notation has proved convenient for reasoning (Ramos et al. 2005, p. 100).

“We can benefit from the use of the Circus refinement calculus to model a system at different abstraction levels, and, by using its refinement laws, verify the consistency of the different refinement levels with the help of formal proofs.”

(Gomes 2012, p. 5)
2 Background

2.5.2 Uses of Circus

The Circus notation by [Woodcock and Cavalcanti (2001a)] has benefited from active development from the academic community since its introduction in 2001. Work has been done to both illustrate the suitability of the Circus notation as a modelling language as well as to extend the formalism to support a richer set of system characteristics.

The work by [Sherif and Jifeng (2002), Wei et al. (2010) and Wei et al. (2011)] proposed different timed models for Circus for studying the properties of timed programs in the untimed model, OhCircus by [Cavalcanti et al. (2003)] extended Circus with object-oriented features (classes, inheritance and dynamic binding), SCJ-Circus by [Miyazawa and Cavalcanti (2015)] supported the specification and verification of Safety-Critical Java (SCJ) models [Henties et al., 2009].

Oliveira et al. (2004) presented a refinement strategy for industrial scale systems in Circus. They illustrated this strategy by refining a Circus specification for an industrial fire control system. They further stated that the illustration is an empirical evidence that the strategy is applicable to large systems.

Freitas and Cavalcanti (2006) described a tool that uses a translation strategy for converting a Circus specification to a Java program. Later, Cavalcanti et al. (2011) extended her work (Cavalcanti et al., 2005) by proposing a semantics for automatically deriving Circus specifications from a subset of Ada programs[12] and proving that such an Ada implementation of a control law diagram is correct.


---

11 "The Safety-Critical Java (SCJ) specification is designed to enable the creation of safety-critical applications using a safety-critical Java infrastructure and using safety-critical libraries that are amenable to certification under DO-178B, Level A and other safety-critical standards" [Henties et al. 2009, p. 3].

12 Wegner (1980) gives a brief history including characteristics of the Ada programming language.

13 "In a control law diagram, systems are modelled by directed graphs of blocks connected by wires. Roughly speaking, wires carry signals, and blocks represent functions that determine how outputs are calculated from the inputs. In a continuous-time model, signals vary continuously; in a discrete model, signals are sampled at fixed time intervals, so that input and output take place in cycles." [Cavalcanti et al., 2011, p. 467]
They use extended versions of existing tools to translate a control law diagram to both Z and CSP specifications respectively, capturing the state and reactive models of a system as required. Finally, they proposed a translation strategy to derive a Circus specification from the generated Z and CSP specifications.

Ramos et al. (2005) proposed a semantics for UML Realtime (UML-RT) via mapping the realtime objects of the UML-RT into Circus. They proposed and proved a decomposition law for those realtime objects to illustrate that the proposed model transformation from UML-RT to Circus is sound (Ramos et al., 2005, p. 109).

Gomes (2012) presented a Circus specification for the Integrated Modular Avionics (IMA) architecture for aircraft systems. In compliance with the ARINC 653 standard (Prisaznuk, 2006), the formalisation focuses on modelling the temporal partitioning of the application layer of the IMA architecture, that prevents the direct communication between applications running on that layer. The Circus specification is validated by deriving a CSP specification from the Circus specification and using the Failures-Divergence Refinement (FDR) tool (Goldsmith et al., 2005) on the resulting CSP specification.

2.5.3 Challenges of using the Circus notation

The absence of a single common BNF for the Circus notation is one of the challenges of using the notation for the formal specification of systems. Further, the absence of a dedicated tool for specifying and type checking the resulting Circus specification is another barrier that hinders researchers from using the Circus notation for developmental

---

14 “The UML Real-Time Profile (UML-RT) addresses modeling concepts that have proven suitable for modelling the run-time architectures of complex real-time systems in application domains such as telecommunications, aerospace, and industrial control.” (Cheng and Garlan, 2001, p. 104)

15 “The IMA architecture consists of a distributed system, where many aircraft applications can be executed in the same hardware module, sharing computing resources, communications and input and output devices.” (Gomes, 2012, p. 3)

16 “The operating system of the IMA architecture is designed in such a way to prevent, through the concept of partitioning, direct communication among applications. It ensures that none of the partitions can share the same memory area or processing time slice.” (Gomes, 2012, p. 4)
2 Background

research. Even though CZT (Malik and Utting, 2005) can be used for type checking a Circus specification, it does not show helpful error messages making it difficult to type check a Circus specification.

2.6 BCF using Circus

This section illustrates how BCF is instantiated using the Circus notation. Recall from Section 2.3 that this instantiation is referred as ‘BCF in Circus’ throughout the rest of this thesis.

2.6.1 User inference through observation

The Circus specification notation provides constructs called channels that allow external actors to interact with a system through inputs and outputs. Through the aforementioned communications and combined with the knowledge about the source code of the system, users may learn information about the values assumed by certain state variables in a particular state or states. This information obtained by a user is termed a user observation of that particular state or states of the system. If a user does not know the exact value of a variable in a particular state, the user will be uncertain about the exact value of that variable in that particular state.

Consider the trivial system in Figure 2.4 that maintains a secret number. Figure 2.5 shows how the inference of two users Alice and Bob differ based on the channels they can access from the system in Figure 2.4.
2.6 BCF using Circus

Naming convention used in Circus specifications

All Circus specifications presented in this thesis follow the naming convention presented in Table 2.3. This convention has been inspired by the naming convention used by Barden et al. (1995) for presenting specifications using the Z notation. The names used in the example column of the Table 2.3 are borrowed from the Circus specification in Figure A.1, except in the case of Free data type.

<table>
<thead>
<tr>
<th>Circus construct</th>
<th>Naming convention used</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic data type</td>
<td>Written with all caps.</td>
<td>CUSTOMER</td>
</tr>
<tr>
<td>Free data type</td>
<td>The data type name is written with all caps. The elements of the free type are written using all lowercase letters. For example, \textit{STAFF} is a free type with elements \textit{no} and \textit{yes}.</td>
<td>\textit{STAFF} ::= no \mid yes</td>
</tr>
<tr>
<td>State variable</td>
<td>Written using all lowercase letters. If the state variable name is a combined word then starting from the second word, capitalize the first letter of each word.</td>
<td>\textit{spent} \textit{currentCustomer} \textit{buyItem}</td>
</tr>
<tr>
<td>Channel</td>
<td>Written using all lowercase letters. If the channel name is a combined word then starting from the second word, capitalize the first letter of each word. Add the postfix ‘\textit{In}’ at the end of the channel name if the channel is used for inputting data into the system. Add the postfix ‘\textit{Out}’ at the end of the channel name if the channel is used for outputting data from the system.</td>
<td>\textit{buyItemIn}</td>
</tr>
<tr>
<td>Channel set, Action,</td>
<td>Capitalize the first letter and use lowercase letters afterwards. If the name is a combined word then capitalize the first letter of each word and use lowercase letters afterwards.</td>
<td>\textit{Customer} \textit{RecordMyReceipt} \textit{State}</td>
</tr>
</tbody>
</table>

Table 2.3: Naming convention used for Circus specifications
2 Background

\[
\text{State} \\
\hspace{2em} n : 1..9
\]

channel \( xIn, xOut \) : 1..9
channelset Alice == \(| xIn, xOut |\)
channelset Bob == \(| xOut |\)

process SecretNumber \( \triangleq \) begin
state State
RecordSecret \( \triangleq \) var \( x : 1..5 \) • \( xIn?x \rightarrow n := x? \)
ShowSecret \( \triangleq \) \( xOut!(n \mod 2) \rightarrow \text{Skip} \)
• RecordSecret ; ShowSecret
end

Figure 2.4: Circus specification of the secret number system

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial observation ( n \in {1..9} )</td>
<td>( n \in {1..9} )</td>
</tr>
<tr>
<td>when executing</td>
<td></td>
</tr>
<tr>
<td>RecordSecret</td>
<td></td>
</tr>
<tr>
<td>channel ( xIn ) visible</td>
<td>Yes</td>
</tr>
<tr>
<td>can observe</td>
<td>( x ) (through ( xIn ))</td>
</tr>
<tr>
<td>can infer about ( n )</td>
<td>( n = x ) (from spec)</td>
</tr>
<tr>
<td>ShowSecret</td>
<td></td>
</tr>
<tr>
<td>channel ( xOut ) visible</td>
<td>Yes</td>
</tr>
<tr>
<td>can observe</td>
<td>( n \mod 2 ) (through ( xOut ))</td>
</tr>
<tr>
<td>can infer about ( n )</td>
<td>already know value of ( x ) and that ( n = x )</td>
</tr>
<tr>
<td>certainty about the value of ( n )</td>
<td>certain</td>
</tr>
</tbody>
</table>

Figure 2.5: Inferences of Alice and Bob from system observations
Alice In the post state of ShowSecret, Alice knows the exact value of \( n \) because she knows the exact value of \( x \) and \( n = x \). In this case, the value of the twin variable \( \tilde{n} \) is the same as the value of the variable \( n \) as follows.

\[
U(n = x) = n = x \land \tilde{n} = x = n = \tilde{n}
\]

Bob In the post state of ShowSecret, Bob only knows that the value of \( n \) is such that either \( n \in \{1, 3, 5\} \) or \( n \in \{2, 4\} \). Assume that Bob figures out that \( n \in \{2, 4\} \). In this case the value of \( \tilde{n} \) in the twin system will be such that \( \tilde{n} \in \{2, 4\} \) as shown below.

\[
U(n \in \{2, 4\}) = n \in \{2, 4\} \land \tilde{n} \in \{2, 4\}
\]

If a user observes the exact value of a state variable \( n \) in a particular state then the value of the twin variable in that particular state is such that \( \tilde{n} = n \) as we have seen from Alice’s inference. We call this the coercion of observation. In the case of Bob’s inference, the uncertainty around the value of the variable \( n \) makes it possible for \( n \) and \( \tilde{n} \) to have different values in the state immediately after the operation ShowSecret. BCF captures the notion of information secrecy based on this uncertainty as discussed in Section 2.6.2.

The indistinguishability relation. The indistinguishability relation \( I(\mathcal{L}) \) (Banks, 2012, p. 87) codifies the observable behaviour of the Circus process \( P \) in the original system and the Circus process \( \tilde{P} \) in the twin system as indistinguishable to a user, having window \( \mathcal{L} \) to a system, when:

\(^{17}\) “Each Circus process has a state and accompanying actions that define both the internal state transitions and the changes in control flow that occur during execution.” (Sampaio et al., 2002, p. 451)
2 Background

- the two processes have the same values for the auxiliary variables (ok, wait, etc).
- the projection\(^{18}\) of the traces [\cite{Banks2012} p. 24] through \(L\) are the same.
- the refusal sets are the same, as long as the behaviour has not terminated.

Here, the user’s window \(L\) is defined as “the set of events communicated by a reactive process which are visible to the user” [\cite{Banks2012} p. 40]. The indistinguishability relation \(I(L)\) is defined as follows:

\[
I(L) \equiv \left( \begin{array}{l}
ok = \widetilde{ok} \land ok' = \widetilde{ok}' \\
\land \text{wait} = \widetilde{\text{wait}} \land \text{wait}' = \widetilde{\text{wait}}' \\
\land (tr' - tr) \uparrow L = (\widetilde{tr'} - \widetilde{tr}) \uparrow L \\
\land \text{wait}' \Rightarrow \text{ref}' \cap L = \widetilde{\text{ref}}' \cap L
\end{array} \right)
\]

By lifting the semantics of the Circus action \(A\) using the construct \(U(A)\) and enforcing \(I(L)\) on the resulting relation, we confine the observational capabilities of the user to the user’s window. This process is captured using the predicate transformer \(UC\).

\[
UC(L, A) \equiv U(A) \land I(L)
\]

Banks models a user’s window as the subset of the channels in a Circus process [\cite{Banks2012} p. 114]. Such a channel set [\cite{Oliveira2009} p. 5] that contains only the set of events visible to a particular group of users reflects the user window of those users to the system.

\(^{18}\) A projection is an observable trace of a process in relation to a particular user window \(L\), where \(L\) is a channelset through which the user access a system.
Blocks. A block is a syntactic structure of BCF in Circus that can be used to specify how Circus actions should be translated to lifted state space. A block is defined as:

\[
\langle L : A \rangle \triangleq (UC(L,a) \triangleleft \ell = L \triangleright A)
\]

where \( L \) is a channelset, \( A \) is a Circus action and \( \ell \) is a window label.

Blocks delineate the boundaries between lifted actions explicitly. BCF in Circus assumes that a system user can only learn information about the program counter at these boundaries. Further, the information that the user can learn at these boundaries depends on the channels which the user can access from \( L \). BCF in Circus formalizes the information the user can learn at these delineated boundaries by proposing back propagation laws. Each back propagation law is proposed for a particular type of Circus action \( A \) and is used to calculate a predicate that represents the information that the user can learn immediately after the execution of the action \( A \). The back propagation laws discussed later in Table 2.4 show how these formal predicates can be calculated from blocks.

Blocks provide a systematic structure to extend the state space of a given Circus action \( A \) while the indistinguishability relation allows us to identify the distinguishable knowledge that can be learnt by observing \( A \) through a given channelset \( L \). The objective of a confidentiality requirement in the context of a system environment is to limit this knowledge someone can obtain by observing the system through the same channelset \( L \). To address this objective, an approach for formalizing the confidentiality requirements of a system is required, so that those formalized requirements can be integrated into a formal specification in such a way that the resulting system specification can be analysed for consistency.
2 Background

2.6.2 Formalising a confidentiality requirement

Confidentiality requirements demand constraints on the information that can be revealed to a user through the user’s interactions with a system. BCF captures confidentiality requirements by maintaining a user’s uncertainty about the value assumed by a state variable, based on his/her observation of the state space of the system. For example, a confidentiality requirement is defined such that $x \neq \tilde{x}$, where $x$ is a variable in a system. In this case all values of $\tilde{x}$ that are different from the current value of $x$ serve as cover stories (Banks, 2012, p. 61) or alternative possible values for $x$. By combining variables from the original system and its twin counterpart as in $x \neq \tilde{x}$, BCF encapsulates a relation between the two when defining a confidentiality requirement. The resulting predicate defines a coercion between a variable in a particular state and its twin counterpart.

The confidentiality predicate must be associated with a channelset to define the scope of the system communications on which the confidentiality predicate can be enforced. For example, the construct $\langle L | x \neq \tilde{x} \rangle$ encodes that the confidentiality predicate $x \neq \tilde{x}$ must be enforced on all the communications through the channels that are included in the channelset $L$.

A confidentiality annotation (CA) (Banks, 2012, p. 105) is a structure similar to a block (see Section 2.6.1) where a logical predicate confidentiality_predicate that encapsulates a confidentiality requirement is associated with a set of channels channelset as in Equation (2.4).

$$\langle channelset \mid confidentiality\_predicate \rangle$$  \hspace{1cm} (2.4)

The CA defined in Equation (2.4) mandates that if the user observing a system at a particular state has access to a channel in the set of channels channelset then the confidentiality constraint defined by the predicate confidentiality_predicate must be enforced on that state. Confidentiality is a ‘relative’ phenomena whereby confidentiality may be conditional on several attributes such as who to conceal the information
2.7 Analysing confidentiality requirements using BCF in Circus

from and under which other conditions the information must be concealed. These constraints may be applied as part of the antecedent of a confidentiality predicate.

Since confidentiality requirements in a system can now be formalized and integrated into a formal system specification, the back propagation approach (Section 2.7) provided in BCF in Circus can now be utilized.

### 2.7 Analysing confidentiality requirements using BCF in Circus

BCF contains a predicate transformer $bw$ that derives a predicate from a Circus specification $S$ that may contain both Circus actions $A$ as well as a confidentiality predicate $Conf$. The predicate transformer $bw$ calculates the weakest precondition of $A$. The derived predicate is used to reason about the consistency of the requirements in $S$.

$$S \triangleq \langle A \rangle; \langle Conf \rangle$$

The specification $S$ embeds the requirement whereby the confidentiality predicate $Conf$ must be satisfied immediately after the execution of $A$. To validate whether this requirement is satisfied, we calculate the weakest precondition of $A$ such that the post state of $A$ can satisfy $Conf$. This calculation can be done using the function $bw(A, \theta)$ whereby it calculates the weakest precondition of $A$ such that the post state of $A$ can satisfy $\theta$, where $\theta$ is a predicate.

If the action was a sequential composition of two or more actions, then the weakest precondition calculation is carried out iteratively starting from the right most action. Consider the specification $S1$.

$$S1 \triangleq \langle B1; B2 \rangle; \langle Conf \rangle$$

---

19 Tschantz and Wing (2008) describes a requirement as a conditional confidentiality requirement if it contains a conditional information flow where information flow occurs only when some condition is met at runtime (Tschantz and Wing, 2008, p. 108). The same has been highlighted later in Section 6.2.4.

20 It must be noted that many confidentiality predicates can be integrated into a single specification.
To observe whether $Conf$ is respected immediately after the action $\langle B_1; B_2 \rangle$ we first calculate the weakest precondition of the action $\langle B_2 \rangle$ using the formula $bw(\langle B_2 \rangle, \langle Conf \rangle)$ and use the resulting predicate when calculating the weakest precondition of $\langle B_1 \rangle$. Hence the overall weakest precondition of the system is defined as:

$$bw(\langle B_1; B_2 \rangle, \langle Conf \rangle) \equiv bw(\langle B_1 \rangle, bw(\langle B_2 \rangle, \langle Conf \rangle))$$ (2.5)

In BCF in Circus terminology, calculating the weakest precondition of a Circus action in this manner is called back propagation. [Banks (2012) p. 138] has provided a set of back propagation laws to compliment BCF in Circus. These laws facilitate the calculation of the weakest precondition for certain atomic actions and composite constructs defined using the Circus notation as shown in Table 2.4.

### 2.7.1 Back propagation laws

The process of back propagation proposed in BCF in Circus uses a collection of back propagation laws [Banks 2012 p. 138]. Later in this thesis, some of the back propagation laws are used to analyse specifications of systems modelled using the Circus notation. The definitions of these back propagation laws are presented in Table 2.4. The back propagation approach for validating the consistency of the requirements in a system specification involves:

- back propagating the specification of the system to generate a resulting predicate that can be used to reason about the consistency of the requirements in the system specification.
- simplifying the generated predicate to reveal whether the predicate is satisfiable.

According to BCF in Circus, a satisfiable predicate indicates that the specification respects all the confidentiality properties coded in the specification whereas if the predicate has a contradiction, this indicates that there is an inconsistency in the specification.

82
### 2.7 Analysing confidentiality requirements using BCF in Circus

<table>
<thead>
<tr>
<th>Action</th>
<th>Syntax</th>
<th>BCF in Circus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign</td>
<td>(a := E)</td>
<td>(\text{bw}((L : a := E), \theta) = \theta[E, \tilde{E} / a, \tilde{a}])</td>
</tr>
<tr>
<td>Output</td>
<td>(c!E \rightarrow \text{Skip})</td>
<td>(\text{bw}((L : c!E \rightarrow \text{Skip}), \theta) = \begin{cases} \theta \land E = \tilde{E} &amp; \text{if } c \in L \ \theta &amp; \text{if } c \notin L \end{cases})</td>
</tr>
<tr>
<td>Input</td>
<td>(c?e : P)</td>
<td>(\text{bw}((L : c?e : P \rightarrow e := e?), \theta) = \begin{cases} \forall e : \delta(c) \cdot P(e) \Rightarrow \theta[e/\tilde{e}] &amp; \text{if } c \in L \ \forall e : \delta(c) \cdot P(e) \Rightarrow \exists \tilde{e} : \delta(c) \cdot P(\tilde{e}) \land \theta &amp; \text{if } c \notin L \end{cases})</td>
</tr>
<tr>
<td>Guard</td>
<td>(g \land B)</td>
<td>(\text{bw}(g \land B, \theta) = (\text{bw}(B, \theta) \land U(g)) \lor U(\neg g))</td>
</tr>
<tr>
<td>External choice</td>
<td>(B_1 \sqcup B_2)</td>
<td>(\text{bw}(B_1 \sqcup B_2, \theta) = \text{bw}(B_1, \theta) \land \text{bw}(B_2, \theta))</td>
</tr>
<tr>
<td>Scope</td>
<td>(\text{var } a : T \cdot B)</td>
<td>(\text{bw}(\text{var } a : T \cdot B, \theta) = \forall a : T \cdot \exists \tilde{a} : T \cdot \text{bw}(B, \forall a : T \cdot \exists \tilde{a} : T \cdot \theta))</td>
</tr>
</tbody>
</table>

**Table 2.4:** A subset of back propagation laws of BCF in Circus by Banks (2012)
2 Background

2.8 Limitations of BCF in Circus

In addition to the limitations of BCF, highlighted in Section 2.3.2, below are some of the limitations specific to BCF in Circus.

**BCF in Circus does not have a mechanizable law for parallel processes.** [Banks (2012)](2012, p. 145) provides a discussion on how to back propagate a CA through a parallel construct. However, this approach involves manual intervention to ‘reform’ ([Banks] 2012, p. 147) the parallel process and so cannot be mechanised. Therefore, further research is required to define a systematic approach and possibly a restricted parallel construct for Circus that can be mechanised.

**BCF in Circus has not been applied to a real system.** [Banks (2012)] demonstrated the practical application of BCF by using BCF in Circus to analyse the confidentiality requirements of a small fictitious auction system. However, he used a heuristic approach when doing the necessary BCF calculations rather than doing them from first principles ([Banks (2012)] p. 135). Further, he did not apply BCF in Circus to a real problem.

**Back propagation laws has not been machine verified.** While all back propagation laws of BCF in Circus has been derived and justified by hand proofs, some of these derivations include non-trivial and lengthy manual calculations.

“All theorems, lemmas and laws presented in this thesis have been justified by hand proof. Nevertheless, it would be expedient to verify their correctness by encoding their proofs in a theorem prover” ([Banks (2012)] p. 187).

Therefore, it would be strongly advisable to machine verify the correctness of these laws before using them. For example, the “bw specification statement” has been derived through such lengthy manual calculations. During the current research, an error with "bw input prefix" law was identified and the correct definition of the law was proposed (see Section 3.3).
2.9 Summary

The main objective of this chapter is to present a preliminary description of the background material that is required to follow the discussions in the remaining chapters of this thesis. It must be recalled that the intention of this research is to develop a practically applicable approach for analysing data leakage related confidentiality requirements in systems using BCF.

Similar to Banks (2012), the intention is to use the instantiation of BCF in Circus for modelling systems. The definition of a practically applicable approach in the current context is a mechanisation where system models can be written and type checked using a tool, extracting the predicate from BCF analysis on the system model can be automated and the simplification of the derived predicate can be automated.

In this chapter:

- a brief description of UTP including the theory of relations, the theory of design, weakest precondition, program correctness and refinement was presented.

- a brief description of the Circus notation was presented.

- a detailed description of BCF was presented.

- other formal and semi-formal approaches that have been proposed by other researchers for analysing confidentiality properties related to data leakage in systems with a confidentiality requirement were reviewed.
3 Mechanisation of BCF

3.1 Introduction

The core objective of this thesis is to produce a practically applicable approach for reasoning about confidentiality in systems using BCF in Circus. While working on achieving this objective, we were also determined to extend the value of BCF in Circus for the research community. Therefore, while designing a practically applicable and suitable mechanisation, the efficiency of the mechanisation was also considered. Figure 3.1 shows the benefits derived from the mechanisation that has extended the value of BCF in Circus. Further, they can be considered as high-level criteria for evaluating the mechanisation produced in this thesis. The benefits derived from the mechanisation are described below.

Practicality explore the development of a practically applicable approach that uses BCF in Circus for analysing systems with a confidentiality requirement. During this exercise, we have fixed an erroneous law in BCF.

Suitability evaluate the suitability of the mechanisation for analysing the different types of confidentiality requirements, as identified in page 149 under Section 5.2.5.

Efficiency explain one possible approach for determining the efficiency of the mechanisation. Compare the efficiency achieved with mechanisation when compared to the manual process.
The above criteria demand the adoption of a combination of design and development research (Ellis and Levy, 2010) and case study research (Easterbrook et al., 2008). In this context, the design and development research involves developing a solution that is practically applicable for analysing systems with a confidentiality requirement, using BCF in Circus. The practicality of the mechanisation has been evaluated through a number of case studies. Case studies for this analysis have been selected through an explicit framework as advised by Easterbrook et al. (2008).

The following subsection discusses how the mechanisation helps in achieving a practically applicable approach for analysing a system using BCF in Circus. After that, the subsequent subsections discuss why the mechanisation is suitable for analysing different types of confidentiality properties supported by BCF in Circus and how the relative efficiency of the mechanisation in comparison to the manual approach can be determined.
3.2 Practicality

The practical use of BCF has been hindered by the following issues:

1. The application of BCF is lengthy and hence, tedious and error prone.

2. Because of the state explosion problem, the back propagation of a system specification results in a huge logical predicate spanning multiple pages. Manually simplifying such huge predicates is error prone.

   “Manual proofs are time-consuming, error-prone and often not economically viable” (Cao and Yu, 2012, p. 48).

Therefore, the application of BCF can only be viable if it is done through a software tool. Further, to mitigate manual errors and to be economically viable, the evaluation of lengthy predicates should be automated or machine assisted so that the users can enjoy the luxury of both “time and precision” when using BCF.

3.2.1 Rationale for a custom tool for mechanising BCF in Circus

Currently, there are no tools that support the mechanised application of BCF. However, mechanising the application of BCF and subsequently automating the simplification of the generated predicate will remove the complex, time-consuming and error prone exercise demanded during the use of BCF. Since BCF is based on the Circus notation, one would be tempted first to try and extend an existing tool that supports the Circus notation, to reduce the development life cycle for the required tool. A custom tool development should only be considered if an extendible candidate cannot be found. Tool support for the Circus notation is limited. To the author’s knowledge, the only tools that provide any form of support for specifying systems using the Circus notation are
3 Mechanisation of BCF

Symphony IDE \cite{Coleman2014} supporting the COMPASS Modelling Language (CML) \cite{Woodcock2012}, CZT \cite{Malik2005} and CRefine \cite{Oliveira2008}. A detailed look into the architecture of these three tools (see Appendix A.2) revealed that the work required to modify any of these tools to support an extension to the Circus notation is not viable within the realm of this doctoral research.

Since there was no viable platform that could be adopted and extended to support BCF in Circus, a decision was made to develop a simple tool for the same purpose. The tool has been named the ‘Confidentiality Framework Application Tool’ (CFAT). And subsequently, the notation supported in this tool is referred as the CFAT notation. Even though BCF has been instantiated for the Circus notation in BCF in Circus, CFAT notation is a non-\LaTeX\ notation. Later, in Section 3.2.3 under the title “CFAT notation”, a discussion is presented that details the reasons for adopting a non-\LaTeX\ notation for modelling systems for the purposes of this thesis.

3.2.2 The proposed mechanisation of BCF in Circus

The proposed mechanisation for analysing systems using BCF in Circus is centred on a tool-chain that contains the CFAT tool developed by the author, Isabelle theorem prover for theorem proving and CZT for type checking Circus specifications. The CFAT tool generates the necessary input that is required by each of the other two tools. Figure 3.2 shows the overall architecture and flow in the proposed mechanisation.

\footnotesize{\begin{itemize}
\item COMPASS Modelling Language (CML) is a language designed for modelling and analysing systems of systems \cite{Woodcock2012}. CML is based on VDM \cite{Gulati2012}, CSP \cite{Hoare1980} and Circus \cite{Oliveira2006}. Symphony IDE \cite{Coleman2014} is a tool that utilizes CML models to generate theorem files to reason about certain properties of these models. The generated theorem files are based on the Isabelle/UTP framework \cite{Woodcock2015}. Isabelle/UTP is a deep embedding of UTP notation in the Isabelle theorem prover \cite{Nipkow2015}.
\end{itemize}}
3.2 Practicality

3.2.2.a The process of analysing a system for data leakage using BCF in *Circus*

The process of analysing a system for data leakage using BCF in *Circus* includes four stages as shown in Figure 3.3. They are the Specification stage, the Back propagation stage, the Predicate simplification stage and the Conclusion stage as described below.

**Specification stage**

A formal model of the system is developed at this stage. Further, the confidentiality requirements of the system are integrated into the formal model.

- In the case of the manual analysis, the system and the confidentiality requirements are modelled using BCF in *Circus*.
- In the case of the proposed mechanisation, a model of the system and its confidentiality requirements are developed using the CFAT notation.

**Back propagation stage**

The back propagation calculation is carried out at this stage.

- In the case of manual analysis, the calculation results in a logical predicate.
- In the case of the proposed mechanisation, a HOL based theory file compatible with the Isabelle theorem prover is generated.

**Predicate simplification stage**

The simplification of the generated predicate is carried out at this stage.

- In the case of the manual analysis, the simplification of the generated predicate is carried out manually.
- In the case of the proposed mechanisation, the simplification of the generated predicate is carried out using the Isabelle theorem prover.

**Conclusion stage**

At this stage, a conclusion is drawn from the result of the simplification carried out during the previous stage.
3 Mechanisation of BCF

3.2.2.b Why each component of the mechanisation is required

The following is a brief description of each component in the tool chain including the reason why the component is required.

Specification stage

Parser. The system model is supplied to the tool in the CFAT notation. The parser is required to parse the input and build an object model that depicts the components of a Circus specification.

\[\text{LATEX interpreter.}\] The LATEX interpreter is required to generate a Circus specification from the CFAT object model of the system, derived by the parser.

Community Z tools (CZT). CZT is used to type-check the Circus specification of the system, generated by the LATEX interpreter.

Figure 3.2: The architecture and the flow of the mechanisation of BCF in Circus for analysing systems with a confidentiality requirement
3.2 Practicality

**Back propagation stage**

**Back propagation laws.** Back propagation laws of BCF are a catalogue of transformation laws for Circus actions. It contains algebraic logic that converts Circus specification statements of certain patterns to formal predicates in higher order logic. Therefore, back propagation laws are required to generate this predicate that can be used to reason about confidentiality in the related system. Table 2.4 presents a subset of the back propagation laws of BCF.

**HiVE mathematical tool-kit.** The Z data structures in the generated predicate needs to be mapped to their equivalent implementations in the target platform, that will be used for predicate simplification. Section 3.2.3 discusses possible frameworks that can be used for this mapping and why HiVE was selected for the particular mechanization approach discussed in this thesis. The HiVE tool-kit provides an implementation of Z data structures in the Isabelle/HOL platform, which is used for predicate simplification. The function names defined in HiVE are used for semantic mapping between Z data structures and their HOL equivalent functions, when the HOL compliant back-propagated predicate is generated by the Isabelle/HOL interpreter.

**Isabelle/HOL interpreter.** The Isabelle/HOL interpreter is responsible for creating HOL definitions for the data types defined in the CFAT model. Further, the interpreter packages these definitions and generates a HOL compliant Isabelle theorem prover theory file which contains the back-propagated predicate and the data type definitions in HOL. The interpreter also creates other supporting theorem files as required. The
Isabelle/HOL interpreter is required because it generates all the necessary theorem files that are required for analysing confidentiality in a given system.

**Predicate simplification stage**

**Isabelle theorem prover.** The Isabelle theorem prover is required to machine assist the simplification of the back-propagated predicate, generated by the Isabelle/HOL interpreter.

### 3.2.3 Design decisions

Decisions were made in selecting specific tools for building the mechanised tool chain. The following is an explanation of what features about the specific tools influenced the design decisions.

**Parser.** The CFAT tool uses a grammar based on ANTLR 4 (Parr 2013) for parsing specifications of systems submitted to the CFAT tool. Some other potential parsers include yacc (Johnson 1975) and JavaCUP (Hudson et al. 1998). The grammar file for yacc is difficult to read because both the grammar rules as well as the instantiations are in the same file. In comparison, ANTLR keeps the grammar rules and the visitor classes in separate files. Further, to the author’s knowledge there is no GUI tool that supports yacc. However, there is GUI support for ANTLR through the standalone tool ANTLRWorks (Bovet and Parr 2008) as well as through plugins for NetBeans (Salter and Dantas 2014), Eclipse (Burnette 2005) and IntelliJ (JetBrains 2017). The GUI support for ANTLR is very useful in debugging a grammar. In addition, the visitor classes are generated in Java, the same language used for CFAT tool development.

The parsed system model can subsequently be used for generating the proof files containing the HOL predicates that are then submitted to a theorem prover for simplification. The parsed system model can also be used for generating Circus specifications.

---

2 A visitor class is an interface that computes and returns values by walking the parse tree (Parr 2013, p. 40).
3.2 Practicality

Figure 3.3: The process for evaluating a system with a confidentiality requirement using the manual approach as well as using the Confidentiality Framework Application Tool
Mechanisation of BCF in \( \LaTeX \) that can be type checked using the CZT tool to assure that the CFAT tool correctly transforms the given CFAT specification to the Circus \( \LaTeX \) syntax.

**CFAT notation.** One concrete syntax of the Circus notation is compatible with \( \LaTeX \). However, inspired by the non-\( \LaTeX \) approach adopted by CRefine \cite{Oliveira2008}, Perfect Developer \cite{Crocker2003} and Symphony IDE \cite{Coleman2014} (that is discussed in Appendix A.3), a decision was made to support a simple and concise notation in our tool, that closely resemble the structure of Circus specifications, but also one that can easily be transformed using a preprocessor into data structures that collectively represent a system model.

---

Formal definition of the Circus notation

To the author’s knowledge, there is no official BNF for the Circus notation. However, there is a need to decide on a clear definition of the Circus notation, before BCF in Circus can be mechanized. Moreover, since our Circus specifications will be type checked using the CZT \cite{Malik2005}, it was logical to follow the Circus BNF presented in Leonardo Freitas’s doctoral thesis “Model Checking Circus” \cite{Freitas2005}; the preliminary work that integrated Circus type checking as part of the CZT tool.

---

Isabelle theorem prover. The earlier design decision to build a custom tool to carry out the back propagation process (see Section 3.2.1) further required a decision on a platform that will be used to evaluate the predicate resulting from the back propagation. The critical criteria when selecting a platform for evaluating this predicate is its support for the Z data structures. In this regard, the theorem prover extensions Isabelle/HiVe \cite{Mahony2009}, Isabelle/ZF, ProofPower-Z \cite{Lemma1Ltd2006, p. 1}, Z/EVES \cite{Freitas2004} and the translation tool Z2SAL \cite{Derrick2011} are suitable since they all contain an encoding of the Z data structures.

Isabelle theorem prover is a LCF style interactive theorem prover \cite{Nipkow2014}. It has a fixed set of core axioms to which all proofs must conform. ProofPower-Z
3.2 Practicality

(\text{Lemma 1 Ltd., 2006} p. 40) extends ProofPower (\text{Lemma 1 Ltd., 2006} p. 1) and supports specification and proofs in Z. ProofPower is also an LCF style interactive theorem prover (\text{Lemma 1 Ltd., 2006} p. 6). Z/EVES is an extension to the EVES proof engine and supports specification and proofs in Z (\text{Freitas, 2004} p. 1). Z2SAL (\text{Derrick et al., 2011}) is a translation tool for model checking Z specifications by translating Z specifications to SAL input language (\text{Moura et al., 2003}) to be used by tools in the SAL tool suit.

Hands-on experience of some academics with Z2SAL has revealed that they have run into difficulty in running the SAL simulator because of the state explosion problem which is related to model checking (\text{Siregar et al., 2014} p. 230). This is not a limitation when using theorem proving. This persuaded the adoption of an approach based on theorem proving, to simplify the predicate.

Z/EVES has a handful of tactics available for dispatching proofs whereas ProofPower has over thousand tactics (\text{Freitas, 2004}). However, support for automatic theorem provers (ATPs) and satisfiabilitymodulo-theories (SMT) solvers in the Isabelle theorem prover lifts the Isabelle theorem prover platform to a whole new level in automating the proof dispatch process. As stated by \text{Blanchette and Paulson, 2016}, the sledgehammer tool in the Isabelle theorem prover provides support for ATPs such as AgsyHOL (\text{Lindblad, 2014}), Alt-Ergo (\text{Bobot et al., 2008}), E (\text{Schulz, 2002}), E-SInE (\text{Hoder and Voronkov, 2011}), iProver (\text{Korovin, 2008}), iProver-Eq (\text{Korovin and Sticksel, 2010}), LEO-II (\text{Benzmüller et al., 2008}), Satallax (\text{Brown, 2012}), SNARK (\text{Stickel et al., 1994}), SPASS (\text{Weidenbach et al., 2000}), Vampire (\text{Riazanov and Voronkov, 2002}), Waldmeister (\text{Hillenbrand et al., 1997}) and Zipperposition (\text{Cruanes, 2015}) and SMT solvers such as CVC3 (\text{Barrett and Tinelli, 2007}), CVC4 (\text{Barrett et al., 2011}), veriT (\text{Bouton et al., 2009}), and Z3 (\text{Böhme and Weber, 2010}). In addition, the Isabelle theorem prover has other automatic proof dispatch tools such as \text{auto} and \text{blast} (\text{Blanchette and Paulson, 2016} p. 24). The decision to select an encoding based on the Isabelle theorem prover has been supported further by the fact that Isabelle theorem prover is a stable and mature

---

3 “A tactic is an ML function which, when applied to a goal, reduces the goal to a list of subgoals and provides a ‘proof function’ which justifies why solving the subgoals will solve the goal” (\text{Cant, 1992} p. 30).
Mechanisation of BCF

theorem prover (Feliachi et al., 2013) and has been utilised in many projects. Further, its has a huge on-line community base and is constantly under development.

The author was not able to find a comparison between Isabelle/ZF and Isabelle/HiVe in the publicly accessible literature. And so, the preference to use HiVe tool-kit was purely based on the “possible” quality of the tool-kit. Isabelle/HiVe has been sanctioned by a government body [4] whereas the the code-base of Isabelle/ZF has been maintained at the University of Cambridge by interested researchers. It may be assumed that Isabelle/HiVe would have gone through a rigorous audit process for a clean and accurate code but this maybe less true for Isabelle/ZF.

**HOL mathematical toolkit.** A mathematical tool-kit is required for the semantic representation of data structures that are translated from Circus to Isabelle/HOL. Data structures in the Circus notation are defined as per the Z notation (Z Standards Panel, 2000) where the semantics of the Z notation is based on the Zermelo-Fraenkel set theory (Spivey, 1988). The HiVe mathematical toolkit by Mahony et al. (2009) and the HOL-Z 2.0 tool-kit by Brucker et al. (2003), both encode the mathematical data structures of the Z notation in the Isabelle/HOL. HiVe was selected rather than the HOL-Z 2.0 because the publicly available version of the HiVe toolkit supports Isabelle 2013-2 whereas the HOL-Z 2.0 is much older and was written in 2003 raising compatibility issues with Isabelle 2013-2 and further is not available publicly. A subset of the definitions from the HiVe tool-kit has been used to encode the mathematical structures of Z using Isabelle theorem prover data structures. From now on, the HiVe mathematical toolkit maybe referred as the ‘mathematical toolkit’. Some of the mathematical notations being used in this thesis is include in Appendix A.4.

**Community Z Tools (CZT).** Community Z Tools by Malik and Utting (2005) supports the parsing of Circus specifications. In spite of the the many limitations (see Section 7.6), CZT is the only available tool for type checking Circus specifications. Therefore, CZT was utilized for this function.

---

4 Australian Department of Defence (Mahony et al., 2009).
3.2 Practicality

3.2.4 Requirements of the major components in the architecture

**Input/output format and the content type.** A number of integration challenges resulted from the architectural decisions that were considered while designing the tool chain, as shown in Figure 3.2. The main three components of the tool chain are the CFAT tool, Isabelle theorem prover and the CZT tool. Table 3.1 shows the input and output file format and content and the additional components required by each major component in the tool chain to perform its function.

<table>
<thead>
<tr>
<th>Component</th>
<th>Input file format and content type</th>
<th>Output file format and content type</th>
<th>Additional components required for the component to function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFAT tool</td>
<td>System specification in CFAT format</td>
<td>- theorem file generated with .thy extension</td>
<td>- Back propagation laws of BCF in Circus</td>
</tr>
<tr>
<td></td>
<td>- Circus specification file generated in \LaTeX{} format with .tex extension</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Isabelle theorem prover</td>
<td>Isabelle theorem prover theory file (extension .thy)</td>
<td>Result of the simplification is shown in the results pane of the theorem prover. (No output file is produced.)</td>
<td>- HiVe mathematical tool kit - Typing files generated by CFAT tool</td>
</tr>
<tr>
<td>CZT tool</td>
<td>Circus specification of a system in the \LaTeX{} format (extension .tex)</td>
<td>Result of type checking is shown on the editor pane of CZT. (No output file is produced.)</td>
<td>- \LaTeX{} type setting package ‘circus’</td>
</tr>
</tbody>
</table>

Table 3.1: The input and output format of each major component of the architecture of the mechanisation of BCF in Circus
Mechanisation of BCF

**Syntactic renaming.** The input and output variables in the Circus notation have the postfix decorations ‘?’ and ‘!’ respectively. Likewise the same postfixes are attached to similar variables in the CFAT notation. The Isabelle theorem prover does not accept these decorations. Hence, a syntactic renaming of the input and output variables are carried out before generating the Isabelle theorem files, one of which contains the back propagated predicate. When generating the predicate, the input and output variables are renamed whereby the “?” and “!” decorations are replaced with “.i” and “.o” respectively.

Each variable of the twin state in BCF in Circus (see Section 2.4.5) is decorated with a tilde such as ˜x where x is a state variable. This decoration is not supported by either CZT or by the Isabelle theorem prover. Therefore, when writing the confidentiality annotation using the CFAT notation, each twin variable in the confidentiality annotation is written with a ‘Z’ prefix. The ‘Z’ prefix represents the twiddle decoration on the twin variable.

When generating the theorem file for analysis using the Isabelle theorem prover, every twin variable reference ˜x has been renamed by prefixing the variable with a ‘Z’ such as Zx in place of ˜x.

When generating the predicate for the Isabelle theorem prover, the twin variables continue to have the ‘Z’ prefix in their variable name. However, when generating the Circus specification file for type checking with CZT, the ‘Z’ prefix is removed from each each twin variable Zx in the CFAT specification followed by decorating the same variable with a subscript such as x₉. The subscript integer 9 or the prefix ‘Z’ are arbitrary choices that do not clash with anything else provided that the restrictions on naming variables are followed as specified next.

**Restrictions on variable names.** In the current development of the CFAT tool, variables of the twin state are identified using the prefix ‘Z’, input variables are identified using the suffix ‘.i’ and output variables are identified using the suffix ‘.o’. Therefore, the following restriction have been applied to any specification submitted via the CFAT
3.2 Practicality

tool editor interface.

- A variable name cannot have the letter ‘Z’ as the first letter.
- A variable name cannot end with the suffix ‘_i’ or ‘_o’.

The above restrictions are specific to the current version of the tool and does not hinder the functionality of the tool. In a future iteration of the tool, these literal identifiers can be replaced with other identifiers outside the letters of the alphabet.

**Transforming CFAT notation to constructs of the Circus notation and Isabelle/HOL.** The CFAT notation provides support for a limited number of constructs which can be used to specify variables, relations, actions and schemas of a Circus specification. The structures must be converted from the original CFAT specification to the target notation using constructs from the target notation that represent the structures of the CFAT notation. Two different types of files are generated by the CFAT tool.

- For type checking purposes, the \LaTeX{} based Circus specification generated has the same system model as the submitted CFAT model.

- For theorem proving purpose, the file generated contains a logical predicate and has no system model. However, the predicate uses data structures from the original CFAT specification. Therefore, the data objects specified using the CFAT notation must be represented using the host notation in the Isabelle theorem prover. The data objects of the CFAT notation themselves represent Z data structures. Appendix A.4 describes how this can be achieved. It is important to note here that Appendix A.4 does not represent a translation from Z to HOL. But rather, Appendix A.4 discusses how type definitions, variable definitions and relation definitions specified using the CFAT notation, that represents their equivalent Z data structures, can be represented using the host language in the Isabelle theorem prover.
Figure 3.4: The CFAT editor window

Figure 3.4 contains part of the CFAT specification of the Circus specification in Figure 6.8. A parser built using ANTLR is used to parse the CFAT specification. The labels in blue color resemble structures of a Circus specification. Predicates can be presented with the same syntactic notation as written using Circus. The keywords var, guarded, assign, output are special keywords in CFAT notation to identify the type of action and chan is used to specify a channel or channelset depending on whether you associate a datatype to it. Liftedvar defines the variables on the twin system.
Figure 3.5: How the predicate \( \text{loginUser} \notin (\text{cashiers} \cup \text{managers}) \) is parsed using the CFAT tool.

Figure 3.5 shows how the predicate \( \text{loginUser} \notin (\text{cashiers} \cup \text{managers}) \) is parsed using the in-built parser in the CFAT tool. How to read the parse tree in Figure 3.5 can be found in the ANTLR Reference Manual (Parr, 2013).
Mechanisation of BCF

3.2.5 The mechanised analysis process

Once a confidentiality integrated CFAT specification is developed from a system description, the consistency of the confidentiality and functionality requirements captured in that specification can be analysed using the mechanised tool chain proposed in this chapter. Figure 3.6 shows the steps which must be followed during this analysis. The following is a description for each step.

1. Develop a CFAT specification of the system based on the description and use cases of the system

Based on the description and models of a system, a CFAT specification for the system is generated. There is no formal translation between the system model or system description to the CFAT specification. The requirement for this translation has been highlighted in Section 7.8 as a potential area of further research that can add value to the mechanised evaluation process.

2. Feed the specification into the mechanised tool

Next, the CFAT specification of the system is fed into the CFAT tool. This is done by typing the specification into the editor interface of the CFAT tool.

3. Generate the back propagated predicate and typing files

The CFAT tool has an interface button called ‘Generate’. When the user presses this button, the tool executes the back propagation of the submitted specification as shown in Figure 3.3. This execution generates theorem files for both the back propagated predicate and the data types defined in the specification. Both these files utilize Z data constructs from the HiVe Mathematical Toolkit by Mahony et al. (2009). The files are generated in a format compatible with the Isabelle theorem prover.

4. Feed the theorem files into the Isabelle theorem prover

The theorem file generated in step 3 for the back propagated predicate is opened in the Isabelle theorem prover. During preprocessing of the file, the
3.2 Practicality

System description and (or) use cases

1. Develop a CFAT specification of the system based on the description and use cases of a system

CFAT Specification

2. Feed the specification into the mechanised tool

Mechanised tool for BCF in Circus

3. Generate the back propagated predicate and typing files

Back propagated predicate (theorem file)

Uses data constructs from

Typing info (theorem file)

HiVe Mathematical toolkit

4. Feed the theorem files into the Isabelle theorem prover

Isabelle theorem prover

5. Apply interactive commands in the Isabelle theorem prover such as:
   * sledgehammer
   * smt
   * auto
   * etc...

Results of analysing the system for consistency of requirements

Simplified: There are no contradictions
Counter example found: There is a contradiction
Theorem prover timeout: Isabelle cannot conclude whether there are no contradictions

Figure 3.6: The mechanised analysis process
Isabelle theorem prover loads all the necessary dependencies. In this case, it includes loading the HiVe Mathematical Toolkit theory package (Mahony et al., 2009) and the theorem files generated for the types defined in the system.

Apply interactive commands in the Isabelle theorem prover

Relevant commands in the Isabelle theorem prover are then used at the provided interactive interface to simplify the predicate.

3.2.6 Interpreting the result of a mechanised analysis

Once the Isabelle theorem prover theory files, generated as per 3 in Figure 3.6, is submitted to the Isabelle theorem prover, as per step 4 in Figure 3.6, the user can use interactive theorem proving commands on the theorem prover, as per 5 in Figure 3.6, to simplify the predicate in the theorem file.

The final result of the simplification may identify one or more contradictions in the predicate. This may be as a result of contradictions between the functional requirements in the system or between the functional requirements and the confidentiality requirements of the system. At the end of the analysis, three possible outcomes can be expected as shown in Figure 3.3. They are:

Simplified If the predicate could be simplified to true, then according to BCF in Circus, there are no contradictions in the system model being analysed.

Counter example found If the theorem prover identifies a counter example, then according to BCF in Circus, there might be a possible contradiction in the system model being analysed. Page 250 shows the automatic generation of a counter example while analysing a specification using the mechanisation developed in this chapter.
Theorem prover time-out If the theorem prover cannot reach a conclusion on simplifying the predicate but rather times-out, then nothing can be concluded about the presence or absence of contradictions in the specification. Therefore the predicate could be true or false. This may be a limitation of the mechanisation proposed in this thesis. In some cases, even though the tool has timed out, it may be possible to manually demonstrate whether the predicate is ‘true’ or ‘false’.

3.3 Fixing the input prefix law

During the mechanisation process it was identified that the bw input prefix law (Banks, 2012, p. 140) was erroneous. Banks presented the following Circus input action

\begin{equation}
\text{bw}(\langle L : c?e : P \longrightarrow e := e? \rangle, \theta)
\end{equation}

and described his encoding of the input prefix law as follows.

“A prefixing which accepts an input value e? from the environment on channel c reveals the exact value of e? to Low, provided Low can observe c. Conversely, if Low cannot observe c, Low can still infer that e? has the type δ(c) and that P(e?) holds” (Banks, 2012, p. 140).

The expression P defines a set of values that represent the type of the variable e?. However, P may also be defined in terms of one or more state variables of the system. For example, consider a system with a state variable called bal and a Circus action called CheckBalance, where PERSON is a set of identifiers for customers.

\begin{align*}
\text{bal} & : \text{PERSON} \rightarrow \text{N} \\
\text{CheckBalance} & \triangleq c?e : (\text{dom bal}) \longrightarrow e := e?
\end{align*}
In CheckBalance, the type $P$ of the variable $e?$ is represented by the expression $\text{dom } bal$. In the lifted state space, the equivalent $\tilde{e}$ will have the type $\tilde{P}$ with the expression $\text{dom } \tilde{bal}$. In the system state space the typing constraint $P(e)$ must hold and in the lifted state space the typing constraint $\tilde{P}(\tilde{e})$ must hold. Which means, if Low cannot observe $c$, Low can still infer that $\tilde{e}$ has the type $\delta(c)$ and that $\tilde{P}(\tilde{e})$ holds. Based on this discussion, the corrected input prefix law is defined in Definition 3.1.

**Definition 3.1. bw Input prefix law (corrected).** A prefixing which accepts an input value $e?$ from the environment on channel $c$ reveals the exact value of $e?$ to Low, provided Low can observe $c$. Conversely, if Low cannot observe $c$, Low can still infer that $\tilde{e}$ has the type $\delta(c)$ and that $\tilde{P}(\tilde{e})$ holds.

$$\text{bw}((\langle L : c?e : P \rightarrow e := e? \rangle), \theta) = \begin{cases} \forall e : \delta(c) \bullet P(e) \Rightarrow \tilde{P}(\tilde{e}) \land \theta[e/\tilde{e}] & \text{if } c \in L \\ \forall e : \delta(c) \bullet P(e) \Rightarrow \exists \tilde{\tilde{e}} : \delta(c) \bullet \tilde{P}(\tilde{\tilde{e}}) \land \theta & \text{if } c \notin L \end{cases}$$

In comparison, the existing bw input prefix law (Banks, 2012, p. 140) proposed in BCF is:

$$\text{bw}((\langle L : c?e : P \rightarrow e := e? \rangle), \theta) = \begin{cases} \forall e : \delta(c) \bullet P(e) \Rightarrow \theta[e/\tilde{e}] & \text{if } c \in L \\ \forall e : \delta(c) \bullet P(e) \Rightarrow \exists \tilde{\tilde{e}} : \delta(c) \bullet P(\tilde{\tilde{e}}) \land \theta & \text{if } c \notin L \end{cases}$$

**Examining the issue with the existing bw input prefix law**

Consider the Circus action ShowBalance and the channelset cashier.

```plaintext
channelset cashier == \{ c, nout \}
```
3.3 Fixing the input prefix law

\[ \text{ShowBalance} \; \equiv \; c?e : (\text{dom bal}) \rightarrow \text{nout}!\text{bal e}? \rightarrow \text{Skip} \]

Assume that a confidentiality annotation CA is back propagated through ShowBalance where ShowBalance is lifted through the channelset cashier.

\[ \langle \text{cashier} \mid \text{ShowBalance} \rangle \; ; \; \langle \text{CA} \rangle \]

The application of BCF is as follows.

\[
\begin{align*}
\langle \text{ShowBalance} \rangle \; ; \; \langle \text{CA} \rangle & \quad (\text{definition of } \text{ShowBalance}) \\
\langle c?e : (\text{dom bal}) \rightarrow \text{nout}!\text{bal e}? \rightarrow \text{Skip} \rangle \; ; \; \langle \text{CA} \rangle & \quad (\text{decomposing the prefix}) \\
\langle c?e : (\text{dom bal}) \rightarrow \text{Skip} \rangle \; ; \; \langle \text{nout}!\text{bal e}? \rightarrow \text{Skip} \rangle \; ; \; \langle \text{CA} \rangle & \quad (\text{definition of } \text{bw} \text{ sequence}) \\
\langle c?e : (\text{dom bal}) \rightarrow \text{Skip} \rangle \; ; \; \text{bw}(\langle \text{nout}!\text{bal e}? \rightarrow \text{Skip} \rangle, \langle \text{CA} \rangle) & \quad (\text{definition of } \text{bw} \text{ output, } \text{nout} \in \text{cashier}) \\
\langle c?e : (\text{dom bal}) \rightarrow \text{Skip} \rangle \; ; \; \langle \text{bal e}? = \text{bal} \text{e}? \wedge \text{CA} \rangle & \quad (\text{definition of } \text{bw} \text{ sequence}) \\
\text{bw}(\langle c?e : (\text{dom bal}) \rightarrow \text{Skip} \rangle, \langle \text{bal e}? = \text{bal} \text{e}? \wedge \text{CA} \rangle) & \quad (\text{definition of the existing } \text{bw} \text{ input prefix law, } c \in \text{cashier}) \\
\forall e? \cdot e? \in \text{dom bal} \Rightarrow \text{bal e}? = \text{bal} \text{e}? \wedge \text{CA} & 
\end{align*}
\]

Looking at the resulting predicate, we see that there is not enough information in the predicate to determine if \( e? \in \text{dom bal} \). If this is the case, \( \text{bal} \text{e}? \) might be undefined for certain values of \( e? \).

Now, the same calculation is carried out again, but with the new input prefix law proposed in Definition 3.1.
3 Mechanisation of BCF

\[ \langle \text{ShowBalance} \rangle ; \langle \text{CA} \rangle \]

\[ = \]  
(definition of ShowBalance)

\[ \langle c? : (\text{dom bal}) \rightarrow \text{nout!bal e?} \rightarrow \text{Skip} \rangle ; \langle \text{CA} \rangle \]

\[ = \]  
(decomposing prefix)

\[ \langle c? : (\text{dom bal}) \rightarrow \text{Skip} \rangle ; \langle \text{nout!bal e?} \rightarrow \text{Skip} \rangle ; \langle \text{CA} \rangle \]

\[ = \]  
(definition of \text{bw} sequence)

\[ \langle c? : (\text{dom bal}) \rightarrow \text{Skip} \rangle ; \text{bw}(\langle \text{nout!bal e?} \rightarrow \text{Skip} \rangle, \langle \text{CA} \rangle) \]

\[ = \]  
(definition of \text{bw} output, \text{nout} ∈ \text{cashier})

\[ \langle c? : (\text{dom bal}) \rightarrow \text{Skip} \rangle ; \langle \text{bal e?} = \text{̂bal} e? \land \text{CA} \rangle \]

\[ = \]  
(definition of \text{bw} sequence)

\[ \text{bw}(\langle c? : (\text{dom bal}) \rightarrow \text{Skip} \rangle, \langle \text{bal e?} = \text{̂bal} e? \land \text{CA} \rangle) \]

\[ = \]  
(definition of the new \text{bw} input prefix law, \text{c} ∈ \text{cashier})

\[ \forall e? • e? \in \text{dom bal} \Rightarrow e? \in \text{dom} \text{̂bal} \land \text{bal e?} = \text{̂bal e?} \land \text{CA} \]

The result of the above calculation shows that \text{e?} \in \text{dom} \text{̂bal} and so \text{̂bal e?} has been defined for all possible values of \text{e?}.

3.4 Suitability

The usefulness of the mechanisation of BCF in Circus can be evaluated in terms of its suitability for the intended purpose. For this, there is a need to check if the types of confidentiality analysis supported by BCF in Circus can be carried out using the using the mechanisation.
3.4 Suitability

3.4.1 Types of data leakage supported by BCF in Circus

Recall from Section 1.3 that Shabtai et al. (2012, p. 5) describe data leakage as the intentional or unintentional distribution of private or sensitive data to an unauthorised entity. Following are some types of data leaks that BCF in Circus can be used to reason about.

Data leakage through direct communication. One way data might leak is through direct communication to the environment. This may happen if a function communicates the value of a state variable $x$ to the environment, while $x$ has already been declared as confidential by a confidentiality requirement in the same system. BCF in Circus can identify such contradictions. For example, the Circus action $\text{ShowX}$ in Equation (3.2) outputs the value of $x$ through the channel $out$. The confidentiality requirement $\text{ConfX}$ in Equation (3.3) states that $x$ must never be revealed through the channelset $L$. Assume that $out \in L$.

$$\text{ShowX} \equiv \langle L \mid out!x \rightarrow \text{Skip} \rangle \quad (3.2)$$

$$\text{ConfX} \equiv \langle L \mid x \neq \tilde{x} \rangle \quad (3.3)$$

The contradiction between the functionality $\text{ShowX}$ and the confidentiality $\text{Conf}$ is brought to light with the following calculation.

$$\langle \text{ShowX} \rangle ; \langle \text{ConfX} \rangle$$

$$= \text{bw}((\langle \text{ShowX} \rangle ; \langle \text{ConfX} \rangle), \text{True}) \quad \text{[Back propagation]}$$

$$= \text{bw}((\langle \text{ShowX} \rangle, \text{bw}(\langle \text{ConfX} \rangle, \text{True}))) \quad \text{[Law 6.32 - bw sequence]}$$

$$= \text{bw}((\langle \text{ShowX} \rangle, \langle x \neq \tilde{x} \rangle)) \quad \text{[Law 6.18 - bw CA]}$$

$$= x = \tilde{x} \land x \neq \tilde{x} \quad \text{[Law 6.34 - bw output]}$$

$$= \text{False} \quad \text{[Simplify]}$$

Given a particular user $u$ with a user role having access to the channels in the channelset $L$, if $u$ cannot observe the channel $out$ then the value of $x$ will not
be revealed to \( u \). This can be possible if \( \text{out} \notin L \). Such role based access control restrictions can be implemented using BCF in Circus.

**Data leakage through inference.** One of the ways in which a data leakage may occur is through inference. A data leakage through inference is where a legal functionality allows an authorised user to deduce something about a part of the system state to which they do not have access rights. The challenge in addressing such a data leakage is called an inference problem (Farkas and Jajodia, 2002).

“The inference problem is denoted as the compromise or increased probability of compromise by deduction of unauthorized information due to combinations of the possession, known existence, known absence, chronology and location of authorized information” (Hubbard et al., 1986, p. 23).

In the context of software systems, access control mechanisms regulate the access which subjects (users or other processes) have on the objects (state variables) of a system (Denning, 1999). However, access control cannot capture the flow of information.

“Access control checks place restrictions on the release of information but not its propagation” (Sabelfeld and Myers, 2003, p. 5).

The philosophy of BCF “for protecting the confidentiality of information is to regulate the information flow from systems to their users” (Banks, 2012, p. 54). Therefore, BCF in Circus maintains knowledge about the origin of the information that is communicated to the environment through a particular state variable \( x \) rather than just knowing the value of \( x \) that is communicated to the environment. By bundling this capability with an RBAC implementation in BCF in Circus, engineers can address some inference problems in Circus specifications.

---

5 Role Based Access Control (RBAC) can be modelled in Circus specifications by regulating user access to channels used in a specification.
### 3.4 Suitability

How BCF in *Circus* can track the flow of information is demonstrated using the following example. For example, the *Circus* action $\text{AssignX}$ in Equation (3.4) assigns the value of the state variable $y$ to the state variable $x$. The confidentiality requirement $\text{ConfY}$ in Equation (3.5) states that $y$ must never be revealed through any channel in the channelset $L$. Assume that $\text{out} \in L$.

\begin{align*}
\text{AssignX} & \triangleq \langle L \mid x := y \rangle \quad (3.4) \\
\text{ConfY} & \triangleq \langle L \mid y \neq \tilde{y} \rangle \quad (3.5)
\end{align*}

Consider the program fragment $\langle \text{AssignX} \rangle; \langle \text{ShowX} \rangle; \langle \text{ConfY} \rangle$. From an access control point of view, the fragment does not have a contradiction because the only output action $\text{ShowX}$ in the fragment does not reveal the state variable $y$, that is required to be kept confidential by $\text{ConfY}$. However, the indirect flow of confidential information in $y$ through the channel $\text{out}$ is revealed if the fragment is analysed using BCF in *Circus*, as shown in the following calculation.

\[ \text{bw}(\langle \text{AssignX} \rangle; \langle \text{ShowX} \rangle; \langle \text{ConfY} \rangle, \text{True}) = \text{bw}(\langle \text{AssignX} \rangle; \langle \text{ShowX} \rangle, \text{bw}(\langle \text{ConfY} \rangle, \text{True})) \quad [\text{Law 6.32 - bw sequence}] \]

\[ = \text{bw}(\langle \text{AssignX} \rangle; \langle \text{ShowX} \rangle, (y \neq \tilde{y})) \quad [\text{Law 6.18 - bw CA}] \]

\[ = \text{bw}(\langle \text{AssignX} \rangle, \text{bw}(\langle \text{ShowX} \rangle, (y \neq \tilde{y}))) \quad [\text{Law 6.32 - bw sequence}] \]

\[ = \text{bw}(\langle \text{AssignX} \rangle, (x = \tilde{x} \land y \neq \tilde{y})) \quad [\text{Law 6.34 - bw output}] \]

\[ = \langle x = \tilde{x} \land y \neq \tilde{y} \rangle[y, \tilde{y} / x, \tilde{x}] \]

\[ = \langle y = \tilde{y} \land y \neq \tilde{y} \rangle \quad [\text{Renaming}] \]

\[ = \text{False} \quad [\text{Simplify}] \]

In summary, BCF in *Circus* can be used to analyse systems to reason about the:

- direct communication of values of state variables to the environment, that are required to be kept confidential as per a confidentiality requirement.
- indirect communication of a confidential data to the environment through a state variable authorised for that particular user.
3 Mechanisation of BCF

3.4.2 Analysing data leakage through indirect communication using BCF in Circus

How the mechanised analysis proposed in this chapter detects a possible data leakage through direct communication is demonstrated later in Section 6.2.1. In this section, data leakage through indirect communication is demonstrated using the results of a mechanised analysis carried out on a fictitious hand crafted specification.

The system has two bidders defined by the free type BIDDER where the bidder can either Alice or Bob. The system maintains the highest bidder of a round of bidding using the state variable highestBidder and the last bidder who has proposed a bid for the same round using the state variable lastBidder. The channelsets Clerk, CustomerX and CustomerY represents user roles where a person having that user role can observe communications through any channels included in that channelset. A detailed discussion of user roles and their importance in the data leakage analysis carried out in this thesis is presented in Section 6.2.3.d. The channel recordBidderIn is used to submit the name of a bidder to the system while showOnlyLastBidderOut and showLastBidderOut are used to output the name of a bidder to the environment. The Circus action RecordHighestBidder is used to record the name of the highest bidder of a bidding round, ShowLastBidder and ShowLastBidderOnly are used to output the name of the last bidder who has submitted a bid for the current round and finally, the Circus action HideHighestBidder (through the schema HideHighestBidder) defines a confidentiality property where the value of the highest bidder must never be revealed. To keep the example small, it is assumed that a value for lastBidder has already been recorded.

The possible interpretations of the results of a mechanised analysis are considered in Section 3.2.6. Table 3.2 discusses the possible reasons for the results of the mechanised analysis of HideHighestBidder. The results show that the mechanisation is suitable for identifying data leakage through indirect flow of confidential data.
\textbf{BIDDER ::= Alice | Bob}

\textbf{State}
\begin{itemize}
  \item \texttt{lastBidder, highestBidder : BIDDER}
\end{itemize}

\textbf{HideHighestBidder}
\begin{itemize}
  \item \texttt{State 9 • highestBidder ≠ highestBidder_9}
\end{itemize}

\textbf{channel} recordBidderIn, showOnlyLastBidderOut, showLastBidderOut : BIDDER

\textbf{channelset} Clerk == \{ recordBidderIn \}
\textbf{channelset} CustomerX == \{ showOnlyLastBidderOut \}
\textbf{channelset} CustomerY == \{ showLastBidderOut \}

\textbf{process} Secret_Highest_Bidder \triangleq \textbf{begin}
\begin{itemize}
  \item \textbf{state} State
  \begin{itemize}
    \item \texttt{RecordHighestBidder} \triangleq \texttt{var recordBidder : BIDDER • recordBidderIn?recordBidder \rightarrow highestBidder := recordBidder?}
    \item \texttt{ShowLastBidder} \triangleq \texttt{lastBidder := highestBidder; showLastBidderOut !(lastBidder) \rightarrow Skip}
    \item \texttt{ShowOnlyLastBidder} \triangleq \texttt{showOnlyLastBidderOut !(lastBidder) \rightarrow Skip}
    \item \texttt{HideHighestBidder} \triangleq \texttt{HideHighestBidder}
  \end{itemize}
  \item \( \mu X \left( \langle \texttt{RecordHighestBidder} \rangle ; \langle \texttt{ShowLastBidder} \rangle ; \langle \texttt{ShowOnlyLastBidder} \rangle ; X \right) \)
\end{itemize}
\textbf{end}

\textbf{Figure 3.7: Specification of Secret Highest Bidder - code block 1 of 1}
### Mechanisation of BCF

<table>
<thead>
<tr>
<th>User role</th>
<th>Outcome of the analysis</th>
<th>Reason for the outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clerk</td>
<td>Counter example found</td>
<td>Any user with the user role Clerk has access to the channel <code>highestBidderIn</code> and therefore the user already knows the highest bidder <code>highestBidder</code>. The confidentiality requirement <code>HideHighestBidder</code> to maintain the secrecy of the highest bidder from this user role will rightly result in a contradiction as shown from the outcome of the analysis.</td>
</tr>
<tr>
<td>CustomerY</td>
<td>Counter example found</td>
<td>The value that represents the highest bidder <code>highestBidder</code> is passed to the state variable <code>lastBidder</code> through the assignment action in <code>ShowLastBidder</code> and therefore the value being revealed in <code>ShowLastBidder</code> through the state variable <code>lastBidder</code> is not the original value of the last bidder but the value of the highest bidder. Since a user with the user role CustomerY has access to the channel <code>showLastBidderOut</code> that user will learn the value of the highest bidder. This knowledge contradicts with the confidentiality requirement <code>HideHighestBidder</code> where it demands to maintain the secrecy of the highest bidder from all users. The analysis identifies this by producing a counter example.</td>
</tr>
<tr>
<td>CustomerX</td>
<td>Simplified</td>
<td>The value that represents the highest bidder is not passed to the state variable <code>lastBidder</code> and therefore the value being revealed through the state variable <code>lastBidder</code> is the current value that represents the last bidder.</td>
</tr>
</tbody>
</table>

Table 3.2: Analysing data leakage through indirect communication
### 3.4.3 Confidentiality violation through recursion

The analysis carried out in Section 3.4.2 has been restricted to a single run of the system because BCF in Circus and therefore its mechanisation does not support the analysis of recursive constructs in Circus. This has been highlighted as an important future area of research (see Section 7.2).

The analysis carried out in Section 3.4.2 is a scenario where the support for recursion in BCF in Circus would have revealed the highest bidder to a user with the user role CustomerX, in contradiction to the outcome shown in Table 3.2. The execution of ShowLastBidder would pass the value of the highest bidder highestBidder to the state variable lastBidder and a subsequent execution of the ShowOnlyLastBidder would have reveal the value of the highest bidder that has been recorded in the variable lastBidder.

### 3.5 Efficiency

Efficiency is a measure of the estimated cost which includes the total time taken for executing user procedures (Seffah et al., 2006, p. 164). Efficiency of a process can be measured in relation to the level of effectiveness achieved to the expenditure of resources where a resource can be the time taken for the process, which can be used to give a measure of the temporal efficiency of the process (Bevan and Azuma, 1997, p. 176). The efficiency measure can be used to compare two or more tasks when carried out in an environment where all parameters of the environment are the same except the variable property of the environment that is being studied. The objective is to check the relative efficiency of the mechanised process in comparison to the manual process.

The manual impracticality of both the back propagation as well as the simplification of the generated predicate when analysing a non-trivial system has been the inspiration for this thesis. Therefore, it goes without question that time taken for the manual analysis of non-trivial systems cannot be obtained. However, since the mechanization
Mechanisation of BCF

developed in this chapter is an important contribution of this thesis, it is important to compare the relative efficiency between the manual and the mechanized analysis of a small specification of a system, for which the time taken for the manual analysis can be obtain.

3.5.1 Comparison of efficiency between the manual and the mechanized analysis

Section 4.2 presents a manual analysis of the system specification in Figure 4.1. The time taken for this analysis as well as the analysis of the same system using the mechanization developed in this chapter is shown in Table 3.3. It must be noted here that there are no matrices to accurately measure the manual analysis of a system using BCF in Circus. Therefore, the stated time for the manual analysis is an estimate. As stated earlier in Section 3.2.6, the outcome “Simplified” indicates that there are no contradictions between the functionality and the confidentiality requirements in the system. Bevan and Azuma (1997, p. 176) propose the following formulae to measure temporal efficiency of a process.

\[
\text{Temporal Efficiency} = \frac{\text{Effectiveness}}{\text{Task Time}}
\]

It is assumed that the Effectiveness is the same for both the manual as well as the mechanized analysis in this example since the analysis outcome from both are the same as shown in Table 3.3. There is no particular unit of measurement given for the Effectiveness measurement. For example, if we state that in this particular scenario both approaches were 100% effective, then the Temporal Efficiency can be calculated as shown in Table 3.3.
### 3.5 Efficiency

<table>
<thead>
<tr>
<th>Type of analysis</th>
<th>Analysis outcome</th>
<th>Time taken for the analysis</th>
<th>Temporal Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>Simplified</td>
<td>3600000 ms</td>
<td>100/3600000 = 0.000027</td>
</tr>
<tr>
<td>Mechanized</td>
<td>Simplified</td>
<td>120 ms</td>
<td>100/120 = 0.833333</td>
</tr>
</tbody>
</table>

Table 3.3: Time taken for analysing Figure 4.1 using BCF in Circus

For this particular scenario, the ratio of the Temporal Efficiency of the mechanized approach in comparison to the manual approach is 1 : 0.003 as shown below.

\[
\frac{\text{Manual efficiency}}{\text{Mechanized efficiency}} = \frac{0.000027}{0.833333} = 0.003
\]
3 Mechanisation of BCF

3.6 Summary

The main contribution of this chapter is the proposition of a mechanisation to make the process of analysing systems using BCF in Circus practically applicable. In addition, this chapter discusses the suitability of the proposed mechanisation, for analysing the different types of confidentiality requirements, supported by BCF in Circus. Finally, the chapter presents a comparison of efficiency between the manual and the mechanized analysis of a given system.

The proposed mechanisation is a tool chain that consists of a custom tool, Isabelle theorem prover and CZT. The custom tool is used for the mechanized back propagation and generation of a back propagated predicate for a given system. The Isabelle theorem prover is used to simplify the predicate, the result of which is used to reason about confidentiality in the given system. The tool further generates a Circus specification of the given system which is type checked using the CZT tool. The mechanisation will be evaluated in Chapter 6.

In this chapter:

- the mechanisation approach that has been adopted in this research, to extend the value of BCF in Circus, has been described.
- the types of confidentiality requirements that can be analysed using BCF in Circus has been identified.
- the correct definition in place of an erroneous definition of the existing bw input prefix law has been proposed.
4 An approach for evaluating the mechanisation of BCF

4.1 Introduction

This chapter presents the approach which has been followed in evaluating the usefulness of the mechanisation of BCF in Circus. There is a need to analyse case studies using the mechanisation so that the advantage of the mechanisation can be demonstrated in terms of saving time and detecting the possibility of a data leakage in systems with a confidentiality requirement. Since the function of the mechanisation is to detect data leakage in a given system, every potential case study system chosen must have one or more distinct confidentiality requirements relating to data leakage so that the case study can be useful in this analysis.

An ideal evaluation would require testing the mechanisation on a range of realistic case studies with different specifications and confidentiality requirements. Such an evaluation can illustrate that the mechanised analysis approach can cope with varied Circus specifications having data leakage related confidentiality requirements. However, such realistic case studies can only be carried out using real world requirement specifications of systems. Such requirement specifications maybe obtained from officially published documents of real life systems. However, if such documents are not available, as was the case in this research, we would have to compile them by being in the field and studying the original system and engaging with the real stakeholders. Such an approach was used by Srivatanakul (2005) in conducting a case study on the Baggage Handling System of the Bangkok International Airport in Thailand. However, based
4 An approach for evaluating the mechanisation of BCF

on the structure of a given Ph.D. research, such an engagement might not be always possible, given the time limitation as was the case in this doctoral research.

4.2 The advantage of mechanisation over a manual approach

Mechanising the back propagation process can save a lot of resources related to time which otherwise might be required when following manual calculations which needs to be carried out with meticulous detail, to avoid human error. The advantage that this mechanisation brings is self evident in a manual run that involves back propagating a non-trivial Circus specification of a system. In this section, a step-by-step walk through for back propagating a small component of a fictitious system is presented.

Figure 4.1 presents the Circus specification of the patient details component of a Health Information System where a doctor uses a system to access details of patients who are being treated by that doctor. The descriptions of the types, state variables, state invariants, channels and actions of this specification are included in Table 4.1.
### 4.2 The advantage of mechanisation over a manual approach

**Table 4.1:** Descriptions of the constructs of the Patient details component of the Health Information System

<table>
<thead>
<tr>
<th>Constructs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATIENT</td>
<td>The set of all possible patient identifiers.</td>
</tr>
<tr>
<td>PATIENTDETAILS</td>
<td>The set of all possible patient detail identifiers.</td>
</tr>
<tr>
<td><strong>treats</strong></td>
<td>The set of people currently being treated at the clinic where the doctor practices.</td>
</tr>
<tr>
<td><strong>reqPatient</strong></td>
<td>The identifier of the patient whose patient details is being requested from the system.</td>
</tr>
<tr>
<td><strong>patientInfo</strong></td>
<td>A function that identifies the details of a patient, if any.</td>
</tr>
<tr>
<td><strong>treats ⊆ dom patientInfo</strong></td>
<td>The set of people treated must be from the set of patients whose patient details are recorded in the system.</td>
</tr>
<tr>
<td><strong>patientIn</strong></td>
<td>The use of this channel by the system is to input the identifier of the patient whose details are being requested from the system.</td>
</tr>
<tr>
<td><strong>detailsOut</strong></td>
<td>The use of this channel by the system is to output the details of the particular patient being requested from the system.</td>
</tr>
<tr>
<td><strong>PatientDetails</strong></td>
<td>Allows a person to request the details of a patient, if the identifier of that particular patient is in the set <em>treats</em>.</td>
</tr>
<tr>
<td><strong>ConfType</strong></td>
<td>Enforces a confidentiality requirement that if the requested patient is not in <em>treats</em> then do not reveal the details of the patient.</td>
</tr>
</tbody>
</table>
[PATIENT, PATIENTDETAILS]

\[
\begin{align*}
\text{State} & \quad \text{treats} : \mathcal{P} \text{PATIENT} \\
& \quad \text{patientInfo} : \text{PATIENT} \rightarrow \text{PATIENTDETAILS} \\
& \quad \text{reqPatient} : \text{PATIENT} \\
\end{align*}
\]

\[
\begin{align*}
treats & \subseteq \text{dom patientInfo} \\
\end{align*}
\]

\[\Xi \text{State} \]

\[
\begin{align*}
\exists \sim \text{State} & \quad \text{reqPatient} \notin \text{treats} \Rightarrow \\
& \quad (\text{patientInfo reqPatient} \neq \sim \text{patientInfo} \sim \text{reqPatient}) \\
\end{align*}
\]

\text{channel} \text{patientIn} : \text{PATIENT}
\text{channel} \text{detailsOut} : \text{PATIENTDETAILS}
\text{channelset} \text{All} \quad \text{==} \quad \{ \text{detailsOut, patientIn} \}

\text{process} \text{PatientSystem} \triangleq \begin{align*}
\text{Init} & \triangleq [\text{State}'] \\
\text{PatientDetails} & \triangleq \textbf{var} \ \text{patient} : \text{PATIENT} \bullet \\
& \quad \text{patientIn?patient} \rightarrow \text{reqPatient} = \text{patient}?; \\
& \quad (\text{patient}? \in \text{treats} \land \text{patient}? \in \text{dom patientInfo}) \land \\
& \quad \text{detailsOut!(patientInfo patient?)} \rightarrow \textbf{Skip}
\end{align*}
\]

\text{ConfType} \triangleq \text{HidePatientDetails}

\bullet \langle \text{Init} \rangle ; \langle \text{PatientDetails} \rangle ; \langle \text{ConfType} \rangle

\text{end}

Figure 4.1: Specification of the patient details component of a Patient information system
4.2 The advantage of mechanisation over a manual approach

Assume that the state variables patientInfo and treats have already been populated. The following is a step-by-step back propagation of the specification in Figure 4.1.

\[ \text{bw}((\langle \text{Init} \rangle; \langle \text{PatientDetails} \rangle; \langle \text{ConfType} \rangle, \text{True}) \]
\[ = \quad [\text{Law 6.32 - bw sequence}] \]
\[ \text{bw}((\langle \text{Init} \rangle; \langle \text{PatientDetails} \rangle), \text{bw}((\langle \text{ConfType} \rangle, \text{True})) \]
\[ = \quad [\text{Law 6.18 - bw CA}] \]
\[ \text{bw}((\langle \text{Init} \rangle; \langle \text{PatientDetails} \rangle), \langle \text{ConfType} \rangle) \]
\[ = \quad [\text{Law 6.32 - bw sequence}] \]
\[ \text{bw}((\text{Init}), \text{bw}((\langle \text{PatientDetails} \rangle, \langle \text{ConfType} \rangle))) \]

The following is the back propagation calculation for \( \text{bw}((\langle \text{PatientDetails} \rangle, \langle \text{ConfType} \rangle)) \).

\[ \text{bw}((\langle \text{PatientDetails} \rangle, \langle \text{ConfType} \rangle)) \]
\[ = \quad [\text{definition of PatientDetails}] \]
\[ \text{bw}(\langle \text{var} \ \text{patient} : \text{PATIENT} \cdot \]
\[ patientIn?\text{patient} \rightarrow \text{reqPatient := patient?;} \]
\[ (\text{patient?} \in \text{treats} \land \text{patient?} \in \text{dom} \ \text{patientInfo} \land \]
\[ \text{detailsOut!(patientInfo patient?) \rightarrow Skip}), \]
\[ \langle \text{ConfType} \rangle) \]
An approach for evaluating the mechanisation of BCF

\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet\]
\[\text{bw}((\langle \text{patientIn} ? \text{patient} \rightarrow \text{reqPatient} := \text{patient}\rangle;\]
\[\langle (\text{patient} ? \in \text{treats} \land \text{patient} ? \in \text{dom patientInfo} \land \]
\[\text{detailsOut}!(\text{patientInfo patient}?) \rightarrow \text{Skip}\rangle),\]
\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet \text{ConfType})\]

= [Law 6.27 - bw scope]

\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet\]
\[\text{bw}((\langle \text{patientIn} ? \text{patient} \rightarrow \text{Skip}\rangle;\]
\[\langle \text{reqPatient} := \text{patient}\rangle;\]
\[\langle (\text{patient} ? \in \text{treats} \land \text{patient} ? \in \text{dom patientInfo} \land \]
\[\text{detailsOut}!(\text{patientInfo patient}?) \rightarrow \text{Skip}\rangle),\]
\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet \text{ConfType})\]

= [Lemma 5.23]

\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet\]
\[\text{bw}((\langle \text{patientIn} ? \text{patient} \rightarrow \text{Skip}\rangle;\]
\[\langle \text{reqPatient} := \text{patient}\rangle),\]
\[\text{bw}((\langle \text{patient} ? \in \text{treats} \land \text{patient} ? \in \text{dom patientInfo} \land \]
\[\text{detailsOut}!(\text{patientInfo patient}?) \rightarrow \text{Skip}\rangle,\]
\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet \text{ConfType})\]

= [Law 6.32 - bw sequence]

\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet\]
\[\text{bw}((\langle \text{patientIn} ? \text{patient} \rightarrow \text{Skip}\rangle;\]
\[\langle \text{reqPatient} := \text{patient}\rangle),\]
\[\text{bw}((\langle \text{patient} ? \in \text{treats} \land \text{patient} ? \in \text{dom patientInfo} \land \]
\[\text{detailsOut}!(\text{patientInfo patient}?) \rightarrow \text{Skip}\rangle,\]
\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet \text{ConfType})\)
4.2 The advantage of mechanisation over a manual approach

= [Law 6.33 - bw guard]

\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet\]

\(\text{bw}(\langle \text{patientIn}?\text{patient} \rightarrow \text{Skip} \rangle;\)

\(\langle \text{reqPatient} := \text{patient}? \rangle,\)

\(\langle \text{bw}(\langle \text{detailsOut}!(\text{patientInfo} \text{patient}) \rightarrow \text{Skip} \rangle,\)

\(\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet \text{ConfType}\)

\(\wedge U(\text{patient}? \in \text{treats} \land \text{patient}? \in \text{dom patientInfo}))\)

\(\lor U(\neg (\text{patient}? \in \text{treats} \land \text{patient}? \in \text{dom patientInfo})))\)

= [Law 6.34 - bw output]

\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet\]

\(\text{bw}(\langle \text{patientIn}?\text{patient} \rightarrow \text{Skip} \rangle;\)

\(\langle \text{reqPatient} := \text{patient}? \rangle,\)

\((\langle \text{bw}(\langle \text{reqPatient} : = \text{patient}? \rangle,\)

\((\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet \text{ConfType} \land\)

\(\text{patientInfo patient}? = \overline{\text{patientInfo}} \overline{\text{patient}}?))\)

\(\wedge U(\text{patient}? \in \text{treats} \land \text{patient}? \in \text{dom patientInfo}))\)

\(\lor U(\neg (\text{patient}? \in \text{treats} \land \text{patient}? \in \text{dom patientInfo})))\)

= [Law 6.32 - bw sequence]

\[\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet\]

\(\text{bw}(\langle \text{patientIn}?\text{patient} \rightarrow \text{Skip} \rangle,\)

\(\langle \text{bw}(\langle \text{reqPatient} := \text{patient}? \rangle,\)

\((\langle \forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet \text{ConfType} \land\)

\(\text{patientInfo patient}? = \overline{\text{patientInfo}} \overline{\text{patient}}?))\)

\(\wedge U(\text{patient}? \in \text{treats} \land \text{patient}? \in \text{dom patientInfo}))\)

\(\lor U(\neg (\text{patient}? \in \text{treats} \land \text{patient}? \in \text{dom patientInfo}))))\)
4 An approach for evaluating the mechanisation of BCF

\[\forall \text{patient} : \text{PATIENT} \bullet \exists \tilde{\text{patient}} : \text{PATIENT} \bullet\]

\[
\text{bw}(\langle \text{patientIn}?\text{patient} \rightarrow \text{Skip} \rangle),
\]

\[
(((\forall \text{patient} : \text{PATIENT} \bullet \exists \tilde{\text{patient}} : \text{PATIENT} \bullet \text{ConfType} \land

\text{(patientInfo patient? = patientInfo patient?)})
\land \U(\text{patient?} \in \text{treats} \land \text{patient?} \in \text{dom patientInfo}))
\lor \U(\neg (\text{patient?} \in \text{treats} \land \text{patient?} \in \text{dom patientInfo})))
\]

\[
[\text{reqPatient, reqPatient/patient?, patient?}]
\]

\[=\]

\[\forall \text{patient} : \text{PATIENT} \bullet \exists \tilde{\text{patient}} : \text{PATIENT} \bullet\]

\[
\text{bw}(\langle \text{patientIn}?\text{patient} \rightarrow \text{Skip} \rangle),
\]

\[
(((\forall \text{patient} : \text{PATIENT} \bullet \exists \tilde{\text{patient}} : \text{PATIENT} \bullet

\text{(reqPatient} \notin \text{treats} \Rightarrow

\text{patientInfo reqPatient} \neq \tilde{\text{patientInfo reqPatient}})
\land \text{(patientInfo patient? = patientInfo patient?)})
\land \U(\text{patient?} \in \text{treats} \land \text{patient?} \in \text{dom patientInfo}))
\lor \U(\neg (\text{patient?} \in \text{treats} \land \text{patient?} \in \text{dom patientInfo})))
\]

\[
[\text{reqPatient, reqPatient/patient?, patient?}]
\]
4.2 The advantage of mechanisation over a manual approach

= [Renaming \( \text{reqPatient}, \text{req\overline{Patient}} \)]

\[
\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet \\
\text{bw}(\langle \text{patientIn}\overline{patient} \rightarrow \text{Skip} \rangle, \\
( ( (\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet \\
\text{(patient} \not\in \text{treats} \Rightarrow \\
\text{patientInfo patient} \neq \overline{\text{patientInfo patient}} \rangle) \\
\wedge (\text{patientInfo patient} = \overline{\text{patientInfo patient}}))) \\
\wedge \text{U}(\text{patient} \in \text{treats} \wedge \text{patient} \in \text{dom patientInfo}) \\
\lor \text{U}(\neg (\text{patient} \in \text{treats} \wedge \text{patient} \in \text{dom patientInfo})))
\]

\[
\forall \text{patient} : \text{PATIENT} \bullet \exists \overline{\text{patient}} : \text{PATIENT} \bullet \\
\text{bw}(\langle \text{patientIn}\overline{patient} \rightarrow \text{Skip} \rangle, \\
( ( (\forall \text{patient} \not\in \text{treats} \Rightarrow \\
\text{patientInfo patient} \neq \overline{\text{patientInfo patient}} \rangle) \\
\wedge (\text{patientInfo patient} = \overline{\text{patientInfo patient}}))) \\
\wedge \text{U}(\text{patient} \in \text{treats} \wedge \text{patient} \in \text{dom patientInfo}) \\
\lor \text{U}(\neg (\text{patient} \in \text{treats} \wedge \text{patient} \in \text{dom patientInfo})))
\]
An approach for evaluating the mechanisation of BCF

∀ patient? : PATIENT • ∃ patient? : PATIENT •

((∀ patient? : PATIENT •

(((patient? ∉ treats ⇒

patientInfo patient? = patientInfo patient?)

∧ (patientInfo patient? = patientInfo patient?)

∧ U(patient? ∈ treats ∧ patient? ∈ dom patientInfo))

∨ U(¬ (patient? ∈ treats

∧ patient? ∈ dom patientInfo)))[patient?/patient?]})

= [Law 6.36 - bw input prefix]

∀ patient? : PATIENT • ∃ patient? : PATIENT •

((∀ patient? : PATIENT •

(((patient? ∉ treats ⇒

patientInfo patient? = patientInfo patient?)

∧ (patientInfo patient? = patientInfo patient?)

∧ (patient? ∈ treats ∧ patient? ∈ dom patientInfo)

∧ (patient? ∈ treats ∧ patient? ∈ dom patientInfo))

∨ (¬ (patient? ∈ treats ∧ patient? ∈ dom patientInfo)

∧ ¬ (patient? ∈ treats ∧ patient? ∈ dom patientInfo)))

[patient?/patient?]}}
4.2 The advantage of mechanisation over a manual approach

\[ \forall \text{patient} \in \text{PATIENT} \bullet \exists \widehat{\text{patient}} \in \text{PATIENT} \bullet (\forall \text{patient} \in \text{PATIENT} \bullet ((\text{patient} \notin \text{treats} \Rightarrow \\
\text{patientInfo patient} \neq \widehat{\text{patientInfo patient}}) \\
\land (\text{patientInfo patient} = \widehat{\text{patientInfo patient}}) \\
\land (\text{patient} \in \text{treats} \land \text{patient} \in \text{dom patientInfo}) \\
\land (\text{patient} \in \widehat{\text{treats}} \land \text{patient} \in \text{dom \widehat{patientInfo}})) \\
\lor (\neg (\text{patient} \in \text{treats} \land \text{patient} \in \text{dom patientInfo}) \\
\land \neg (\text{patient} \in \widehat{\text{treats}} \land \text{patient} \in \text{dom \widehat{patientInfo}}))) \]

= [Renaming]

\[ \forall \text{patient} \in \text{PATIENT} \bullet (\forall \text{patient} \in \text{PATIENT} \bullet ((\text{patient} \notin \text{treats} \Rightarrow \\
\text{patientInfo patient} \neq \widehat{\text{patientInfo patient}}) \\
\land (\text{patientInfo patient} = \widehat{\text{patientInfo patient}}) \\
\land (\text{patient} \in \text{treats} \land \text{patient} \in \text{dom patientInfo}) \\
\land (\text{patient} \in \widehat{\text{treats}} \land \text{patient} \in \text{dom \widehat{patientInfo}})) \\
\lor (\neg (\text{patient} \in \text{treats} \land \text{patient} \in \text{dom patientInfo}) \\
\land \neg (\text{patient} \in \widehat{\text{treats}} \land \text{patient} \in \text{dom \widehat{patientInfo}}))) \]

= [Eliminate outer existential quantifier]
An approach for evaluating the mechanisation of BCF

∀ patient? : PATIENT •

(((patient? ∉ treats ⇒
    patientInfo patient? ≠ patientInfo patient?)
∧ (patientInfo patient? = patientInfo patient?)
∧ (patient? ∈ treats ∧ patient? ∈ dom patientInfo)
∧ (patient? ∈ treats ∧ patient? ∈ dom patientInfo))
∨ (¬ (patient? ∈ treats ∧ patient? ∈ dom patientInfo)
∧ ¬ (patient? ∈ treats ∧ patient? ∈ dom patientInfo))))

Continuing the back propagation from page 125

bw(⟨Init⟩, bw(⟨PatientDetails⟩, ⟨ConfType⟩))

= [simplification of bw(⟨PatientDetails⟩, ⟨ConfType⟩)]

bw(⟨Init⟩,

(∀ patient? : PATIENT •

(((patient? ∉ treats ⇒
    patientInfo patient? ≠ patientInfo patient?)
∧ (patientInfo patient? = patientInfo patient?)
∧ (patient? ∈ treats ∧ patient? ∈ dom patientInfo)
∧ (patient? ∈ treats ∧ patient? ∈ dom patientInfo))
∨ (¬ (patient? ∈ treats ∧ patient? ∈ dom patientInfo)
∧ ¬ (patient? ∈ treats ∧ patient? ∈ dom patientInfo))))

132
The advantage of mechanisation over a manual approach

\[ \forall \text{patient? : PATIENT} \bullet \]

\[
( ( \text{patient?} \notin \text{treats} \Rightarrow
\text{patientInfo patient?} \neq \text{patientInfo patient?})
\land (\text{patientInfo patient?} = \text{patientInfo patient?})
\land (\text{patient?} \in \text{treats} \land \text{patient?} \in \text{dom patientInfo})
\land (\text{patient?} \in \text{treats} \land \text{patient?} \in \text{dom patientInfo}))
\lor (\neg (\text{patient?} \in \text{treats} \land \text{patient?} \in \text{dom patientInfo})
\land \neg (\text{patient?} \in \text{treats} \land \text{patient?} \in \text{dom patientInfo}))
\]

= [Predicate calculus]

true

The above manual back propagation of the specification in Figure 4.1 took approximately one hour to complete whereas the mechanised back propagation of the same specification using the tool (see Chapter 3) developed in this research took 120 milliseconds to complete. This shows a time-saving advantage of the mechanised back propagation approach over the manual approach. In addition, the amount of human errors that could be avoided through this mechanised calculation approach is self evident.
4 An approach for evaluating the mechanisation of BCF

4.3 Value of the mechanisation

During the case study analysis carried out as part of this research, we were able to verify the consistency of the requirements in a specification where a real issue with the formalisation of the confidentiality requirement was not apparent at first glance. However, later it was noticed that specification was seemingly incorrect. Upon further review, it was identified that we had circumvented the weakness in the specification of the system by strengthening the formalisation of the confidentiality requirement, that allowed us to verify the consistency of the requirements in the system. In this thesis, such a specification is called a ‘weak specification’.

**Weak specification.** For the purposes of this research, a specification is referred as a ‘weak specification’ if the specification is seemingly incorrect but the mechanised analysis of the specification using BCF in *Circus* results in a predicate that can be simplified to true, thereby verifying the consistency of the requirements.

This experience with a ‘weak specification’ is discussed and demonstrated in Section 6.2.7 using a fictitious hand-crafted system. The discussion further demonstrates the value of the mechanisation. This is because the analysis did not verify the consistency of the requirements until the confidentiality requirements were strengthened.

4.4 Benchmark for evaluation

To the author’s knowledge, there is no existing literature that defines a “benchmark set” of case studies that can be used to evaluate a potential tool for analysing systems with a confidentiality requirement. However, many papers that cover a broad range of systems especially in the domain of information flow theories, security and privacy discusses various confidentiality properties. And so, deriving an initial catalogue of case studies that can be used for evaluating the mechanisation of BCF in *Circus* or other similar mechanisations of security related tools as well as that can be extended by other
researchers would be a useful contribution. Such a catalogue is proposed in the next chapter. Later in this thesis, this catalogue is used for evaluating the mechanisation of BCF in Circus.

4.5 Limitations of the catalogue approach for evaluation

The papers identified in Chapter 5 discuss confidentiality requirements that are required in certain system contexts. However, none of the identified papers contain any description or formal specification of these systems. Therefore, in order to analyse each identified confidentiality pattern CP, the following activities were carried out.

- Select a confidentiality requirement from Table 5.3 that reflects the pattern CP.
- Hand-craft a set of system requirements for a typical system for which CP has been identified as a confidentiality requirement.
- Develop a formal specification by interpreting the hand-crafted system requirements for the system.

It is a limitation of this research that specifications for case studies had to be developed based on hand-crafted system requirements. However, if there was at least a full description of a system in the identified papers, the system description could have been re-written using a Controlled Natural Language (CNL). An unambiguous and structured system description that follows a CNL can systematically be translated to formal specifications that can be used for analysing those systems for data leakage.

Cabral and Sampaio (2008) have produced a tool that can translate a specification written in a particular CNL to CSP. CSP being the reactive notation used in the Circus notation, such an approach can directly map reactive characteristics from a requirement document to a formal specification in the Circus notation. Since a CNL can be used

---

1 “CNLs are engineered subsets of natural languages whose grammars and vocabularies have been restricted in a systematic way in order to reduce both the ambiguity and complexity of full NLs (e.g. English, French, etc.)” Feuto Njonko et al. (2014). p. 68.)
4 An approach for evaluating the mechanisation of BCF

in such translations, further research can be carried out as to how the data structures can be mapped from a similar CNL to the Z notation, where the Z notation is used to encode data structures using the Circus notation. An effort in this direction is the work by Becker (2007) who proposed the use of Cassandra to formalise policy specifications. Translating specifications written in a particular CNL to the Circus notation might be appropriate as a further extension to the mechanisation presented in this thesis.

---

2 Cassandra is a policy specification language that is based on Datalog (Abiteboul et al., 1995) with constraints.
4.6 Evaluation plan

The evaluation of the tool will be carried out using the following steps.

1. Identify a set of patterns recurring in the catalogue of confidentiality requirements that has been derived from literature, where these requirements are related to data leakage.

2. Present a case study example for each identified confidentiality requirement pattern. The case study example will involve:
   - hand-crafting an unambiguous description of the system.
   - developing an appropriate Circus specification of the system.
   - encoding the confidentiality requirements using BCF in Circus.
   - using the mechanisation of BCF in Circus to analyse the formalized system for data leakage by executing the back propagation process and simplifying the resulting predicate using the Isabelle theorem prover.

3. Demonstrate how the weakness in a specification can be circumvented to make it verifiable when analysed using BCF in Circus.

4. Demonstrate how a contradiction in a specification can be detected when a system specification is analysed using BCF in Circus by introducing a contradiction purposely in the specification.

5. Demonstrating positive tests where specifications in the above case studies without a contradiction are verified when analysed using BCF in Circus.

6. Use the time taken for executing the back propagation process using the tool and the time taken for simplifying the generated predicate as parameters, to compare the relative resource utilization when different specifications are analysed using the proposed mechanisation on a common platform.
An approach for evaluating the mechanisation of BCF

The objective of the evaluation is to show that the mechanisation correctly identifies the specifications that contain an inconsistency within the specification and specifications that do not. As you may recall from Section 2.7.1 on page 82, BCF concludes that a system may leak data if the predicate generated by back propagating the specification of a system contains a contradiction. Similarly, if the predicate generated does not contain a contradiction, then BCF concludes that the system will not leak data. Here, a contradiction reflects a logical inconsistency within the formal definitions of the functional and confidentiality requirements in a system specification.
Types of specifications analysed

During the evaluation, the following different types of specifications have be analysed.

Positive specification  For the purposes of this research, a positive specification is defined as one which is written without any contradictions between the functionality and confidentiality requirements of the system. It is expected that applying the mechanised tool on such a specification will produce a predicate that can be Simplified to true by the Isabelle theorem prover.

Weak specification  Recall from Section 4.3 in page 134 that during the analysis we were able to verify the consistency of the requirements that seemed to be incorrect. However, this verification was possible by circumventing the weakness with the specification using a modified formalisation of the confidentiality requirement. This issue has been demonstrated in Section 6.2.7.

Negative specification  For the purposes of this research, a negative specification is defined as one which is written with at least one definite contradiction between the functionality and confidentiality requirements of the system. It is expected that applying the mechanised tool on such a specification will produce a predicate that cannot be Simplified by the Isabelle theorem prover. Rather, attempting to simplify the predicate using the Isabelle theorem prover will result in either Theorem prover time-out or Counter example found as outcomes.
Possible categories of specifications based on analysis outcome

In general, there can be four different categories of specifications as listed below.

- **positive and provable**: A positive specification that can be proved automatically using proof tactics.

- **positive and not provable**: A positive specification that cannot be proved automatically using proof tactics. In this case, expert assistance is required for theorem proving. The theorem proving exercise may result in a time-out when some automated theorem proving commands are executed.

- **negative and provable**: A negative specification for which a counter example can be generated automatically using proof tactics.

- **negative and not provable**: A negative specification for which a counter example can be generated using automatically using proof tactics. In this case, expert assistance is required for identifying a counter example. The theorem proving exercise may result in a time-out when some automated theorem proving commands are executed.
4.7 Summary

The main objective of this chapter is to layout a plan for evaluating the proposed mechanisation of BCF in Circus. This plan has been derived based on both the capabilities of the mechanisation as well as the factors that confine the space where the mechanisation can be utilized.

In this chapter:

- The comparative advantages of the proposed mechanisation over the manual approach has been demonstrated.
- How the proposed mechanisation can be beneficial in identifying weak specifications which otherwise might be overlooked when following the manual approach was demonstrated.
- The limitations that guide and confine the approach that has been taken in mechanising BCF in Circus was discussed.
- The plan for evaluating the mechanisation of BCF in Circus has been discussed.
5 A systematic literature search for case study material

5.1 Introduction

This chapter presents an approach for compiling a catalogue of confidentiality requirement patterns from the literature. A pattern is a generalized description of a commonly occurring requirement (Dwyer et al. [1999], p. 412). A pattern catalogue is a collection of related patterns subdivided into different categories (Hakeem [2010], p. 23).

In this chapter, a literature search for confidentiality requirements are carried out and the results are analysed to identify recurring patterns in those confidentiality requirements. In the next chapter, system models that contain confidentiality properties based on these patterns will be used in evaluating the practical applicability of the mechanisation of BCF. To the author’s knowledge, system models that contain confidentiality properties based on such a pattern catalogue has not been published in the publicly accessible literature to this date. However, such a catalogue is required to evaluate the BCF in Circus mechanisation developed through this research. This chapter discusses how such a catalogue has been compiled to address this need. A systematic literature search process is adopted to identify publications that can be included in such a catalogue.

In addition, this chapter includes an approach for extracting generalized formalisations from the formalisations of confidentiality requirement patterns. The result is a set of
minimum unique patterns of formalisation required to capture all the identified patterns of confidentiality requirements. Any technique or approach intending to provide support for the catalogue of confidentiality requirements can do so by providing support for this minimum set of generalized formalisations. In this research, this set of generalized formalisations is referred to as the “generalized patterns of confidentiality requirements”. Similarly, when designing a software to support the catalogue of confidentiality requirements, we must ensure that the software supports the specification templates that reflect the generalized patterns of confidentiality requirements.

The ‘pattern’ approach for software engineering became common place in the software development community after Gamma et al. (1995) published the book *Design Patterns: Elements of Reusable Object-oriented Software*. In their book, they used “patterns to capture good solutions to common problems programmers experienced when designing software” (Adolph et al., 2003, p. 7).

### 5.2 Systematic literature search for case study material

A systematic process is followed in this literature search to enhance the quality and relevance of the resulting dataset.

#### 5.2.1 Research question

The research question addressed in this chapter is:

| What confidentiality properties or requirements have been discussed in software engineering literature? |

BCF in *Circus* integrates aspects of secure information flow theories with a formal system development approach. Formal system development is a software engineering approach that aims at delivering reliable software.
5.2 Systematic literature search for case study material

“Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines” (Andreson, 2014, p. 496).

An unambiguous description of a system is required for formal system development. Further, scenarios with a confidentiality requirement or confidentiality property are needed to test secure information flow theories related to confidentiality.

Therefore, the ideal set of testing data for analysing BCF in Circus must have unambiguous descriptions of systems with a confidentiality requirement. The above research question confines the literature search to software engineering to limit the search base while maximising the possibility of extracting unambiguous descriptions of systems.

5.2.2 Identification of indexing services

The following phrases were used as a composite search string in Google Scholar to identify the literature that discusses systematic literature review of software engineering.

"systematic literature review" "software engineering"

The first 20 results were considered. The motivation was to categorize the collection of indexing services which each author thought was relevant for software engineering research. Table 5.1 lists the most recurring indexing services which the author used.

Based on the number of papers where the indexing server was mentioned, as shown in Table 5.1, the indexing services IEEE Explorer, ACM Digital Library, ScienceDirect, SpringerLink and Google scholar were selected as indexing services to based the systematic literature search on. CiteSeer was excluded because it has a smaller digital library than Google Scholar (Khabisa and Giles, 2014, p. 1) even though both had the same number of mentions.
5.2.3 Inclusion criteria

The inclusion criteria for the documents retrieved were as follows.

1. The document must be in English.
2. Do not include repeated copies of the same document.
3. PDF and postscript documents were considered for this collection.
4. Only full documents rather than parts of documents were considered.
5. Only published journal papers, conference papers, thesis and reports were considered.

The collection of documents that conformed to the inclusion criteria is called the literature base.

During the preparation for the literature search, the author identified that privacy and confidentiality were often used in the context of lawyer-client confidentiality, patient privacy and patient-doctor confidentiality. However, sometimes these discussions were not centred on system engineering and design. During the literature search, such papers were therefore not considered. Section 5.2.1 explains how confining the literature search to software engineering may help in getting search results that discusses confidentiality analysis centred on system engineering and design.

5.2.4 Search keywords

The ecosystem of research addressed in this thesis is confidentiality engineering. A keyword map that represents the confidentiality engineering ecosystem is presented in Figure 5.1
5.2 Systematic literature search for case study material

<table>
<thead>
<tr>
<th>Indexer</th>
<th># of times mentioned in the 20 papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE Explorer</td>
<td>16</td>
</tr>
<tr>
<td>ACM Digital Library</td>
<td>15</td>
</tr>
<tr>
<td>ScienceDirect</td>
<td>11</td>
</tr>
<tr>
<td>SpringerLink</td>
<td>9</td>
</tr>
<tr>
<td>Google scholar</td>
<td>5</td>
</tr>
<tr>
<td>CiteSeer</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5.1: Relevant catalogues for software engineering research

![](image)

Figure 5.1: Keyword map describing the confidentiality engineering ecosystem

Based on the keyword map in Table 5.1, the following words and phrases were used to search for literature that may discuss confidentiality engineering during system design.

- “confidentiality” and “formal methods”
- “confidentiality” and “system design”
- “confidentiality properties”
A systematic literature search for case study material

- “confidentiality requirements”
- “security engineering”
- “secure by design”
- “privacy by design”

5.2.5 Literature selection

The literature search carried out in this research showed that very few published papers described a complete model of a system. Rather, papers discussed scenarios within the system execution where a confidentiality assurance is required and how to fix it. Therefore, any literature that mentioned the need for confidentiality of information in the context of an information system was included.

Search results revealed 490 articles. All articles were put on a Mendeley catalogue (Mendeley Ltd., 2016). The catalogue was searched using the keyword ‘confidential’. The inbuilt search system in Mendeley was used to search for the keyword. If a keyword existed in a paper, Mendeley showed the number of occurrences of that keyword in that paper. Mendeley takes the user directly to each occurrence in the paper through its inbuilt interface. The paragraph, table or diagram where each occurrence of the keyword was reviewed to see if a confidentiality requirement was described. Table 5.3 presents the collection of confidentiality requirements that have been identified following this search process.

Sequential access implementations. The system contexts where some of these confidentiality requirements were described might have been based on systems with concurrent access. However, the systems modelled in this research have been restricted to sequential access implementations because the catalogue of back

---

1 As highlighted in the introduction section, the words Privacy and Confidentiality have been used interchangeably by many author’s to mean the same thing. Our objective is to cover a broad ground regarding the existing literature that discusses confidentiality assurances required in system design.
propagation laws for BCF in Circus does not provide support for parallel processes. This limitation has been discussed in Section 2.8.

**Types of data leakage risks considered.** As discussed in "On data leakage and the inference problem" on pages 111-113, BCF in Circus can be used to reason about the possibility of data leakage through:

- direct communication of values of state variables to the environment, that are required to be kept confidential as per a confidentiality requirement.
- indirect communication of confidential data to the environment through a state variable authorised for that particular user.

Therefore, the confidentiality requirements selected for inclusion in the confidentiality catalogue have been restricted to the above types of requirements.

**The importance of analysing patterns of confidentiality requirements.** When evaluating a technique that has been proposed for analysing confidentiality requirements in systems, it will be more justifiable to consider candidate requirements from groups of requirements having the same pattern, rather than individual requirements. By using this approach, a broader range of requirements can be addressed during the evaluation of the mechanized analysis technique. Confidentiality requirements take similar forms in many contexts. Such similarities can be the basis for grouping confidentiality requirements. Each resulting group will have confidentiality requirements with the same pattern. The next section presents the derivation of groups of confidentiality requirements, each having a different pattern.
5.3 Patterns of confidentiality requirements

The main objective of this section is to identify and review common patterns that may exist in a collection of confidentiality requirements, which has been extracted from literature. Each confidentiality requirement in this collection has been identified by a unique identifier of the form \texttt{CRX} where \texttt{X} represents a unique integer assigned to that particular requirement. A total of 33 CRs (\texttt{CR1–CR33}) were extracted from the literature and these confidentiality requirements are presented in Table \ref{tab:patterns}. Section \ref{sec:patterns}

Table \ref{tab:patterns} presents the patterns that exist in the descriptions of these confidentiality requirements. The table further groups the collection of confidentiality requirements based on which pattern each confidentiality requirement falls under. A pattern reflects data leakage risks of a similar form across many contexts.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Pattern description</th>
<th>Confidentiality requirement where the pattern exists</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{CP1}</td>
<td>do not reveal the relation between \textit{x} and \textit{y} in \textit{S}</td>
<td>\texttt{CR3, CR4, CR5, CR6, CR7, CR8, CR9, CR11, CR12, CR13, CR18, CR23, CR27, CR29, CR33}</td>
</tr>
<tr>
<td>\texttt{CP2}</td>
<td>do not reveal whether \textit{x} is a member of \textit{S}</td>
<td>\texttt{CR10, CR14, CR20, CR21, CR22, CR24, CR25, CR26}</td>
</tr>
<tr>
<td>\texttt{CP3}</td>
<td>do not reveal the set \textit{S}</td>
<td>\texttt{CR28, CR30, CR32}</td>
</tr>
<tr>
<td>\texttt{CP4}</td>
<td>do not reveal the exact value of \textit{x}</td>
<td>\texttt{CR1, CR2, CR15, CR16, CR19, CR31}</td>
</tr>
<tr>
<td>\texttt{CP5}</td>
<td>do not reveal whether the value of \textit{x} is lower/higher than a given threshold \textit{n}</td>
<td>\texttt{CR17}</td>
</tr>
</tbody>
</table>

Table 5.2: Patterns of confidentiality requirements
5.3.1 Deriving patterns of confidentiality requirements

Table 5.2 includes the collection of confidentiality requirements, identified from the set of literature, short-listed in Section 5.2.5. It is important to discuss briefly, the process which has been followed in identifying and deriving patterns of confidentiality requirements from this collection. The steps in this process include:

- Extracting a direct quote or paraphrasing the confidentiality requirement discussed in a given paper.
- Rephrasing the extracted description of each confidentiality requirement using a structured statement.
- Categorizing the structured statements based on commonalities in their descriptions.
- Presenting each category as a pattern of confidentiality requirements in Table 5.2.
Table 5.3: A collection of confidentiality requirements from literature

<table>
<thead>
<tr>
<th>Context scenario</th>
<th>Confidentiality property/goal required</th>
</tr>
</thead>
<tbody>
<tr>
<td>a structured general definition for the confidentiality requirement</td>
<td></td>
</tr>
</tbody>
</table>

Context: Sealed bid auction system

The bid value $b$ and the randomiser $r$ must not be revealed to anyone except the bidder until the bid submission phase.

**CR1**

if the user is not the bidder of $b$
then do not reveal the value of $b$ (which represents the bid value)

**CR2**

if the user is not the bidder of $b$
then do not reveal the value of $r$ (which represents the randomiser value)

Context: Clinical information system

“personal information about patients must be kept secure and confidential”.

“identifying information must not be made available to government and health authorities.”

**CR3**

if the user belongs to the government or a health authority
then do not reveal the association between
the patient and his/her medical data

Continued on next page
5.3 Patterns of confidentiality requirements

Table 5.3 – Continued from previous page

“An unauthorized source should never access to the content”. No one apart from authorised personal can have access to personal identifiable health information.

(Juan et al., 2011, p. 26)

| CR4 | if the user does not have the necessary authorisation then do not reveal the association between the patient and his/her medical data |

“The IS shall protect the privacy of patients and their associated medical records. The confidentiality of data shall be guaranteed. Moreover, the database shall be available for allowing the access of patient medical information in case of urgent need”.

“Focusing on the patient data and its relationship with medical data, a privacy goal is associated with them. However medical data (without its associated relationship with patient data) is not constrained by this privacy goal.”

(Mayer et al., 2005, p. 2,10)

| CR5 | if the user does not have the necessary authorisation and the request is not made under a condition of urgency then do not reveal the association between the patient and his/her medical data |

Continued on next page
Table 5.3 – Continued from previous page

“For example, a patient typically gives consent to his/her medical record to physicians in a call group (set of physician sharing a practice), however, patients with particular conditions might only give consent to their family physician”.

Onabajo and Jahnke (2006b)

| **CR6** | if the patient has a particular condition and the user is not the family physician then do not reveal the association between the patient and his/her medical data |

“The context of the confidentiality statement is not only based on the purpose, but also on potential user(s) or stakeholder(s) who require access to data e.g., a patient would normally give access to his/her medical record to a physician for care-delivery, but deny access to non-medical staff”.

“For example, access to sections of a medical record, such as prescription history, might have potential influence on treatment options, particularly during emergencies”.

Onabajo and Jahnke (2006a, p. 34)

| **CR7** | if the user is not a physician then do not reveal the association between the patient and his/her medical data |

Continued on next page
5.3 Patterns of confidentiality requirements

It is important to preserve individual privacy of the persons in healthcare information systems.

| CR8 | if the user is not authorized | then do not reveal the association between the patient and his/her medical data |

“We motivate the need for conditional confidentiality by referring to the doctor example from the introduction. As described before, the system should allow access to the X-ray only if the user is a doctor.”

| CR9 | if the user is not a doctor | then do not reveal the association between a patient and his/her x-ray |

“Protection of the integrity and confidentiality of medical images is an issue in the management of patients’ medical records. Confidentiality states that unauthorized parties should not be granted to access medical images during transmission”.

| CR10 | if the user is not authorized | then for every given $x$, do not reveal whether $x$ is a member of the set $S$ (where $S$ contains the set of x-rays) |

Continued on next page
“the specific association between individual patients and their illnesses is sensitive and should be maintained confidential”.

(De Capitani di Vimercati et al., 2014, p. 214)

**CR11**  
if the user is not authorized  
then **do not reveal the association** between  
a patient and his/her illness

“The last kind of attribute is the confidential attribute, the values of which we have to protect”.

“For example, if the attribute is a HIV test result, then the revelation of a ‘+’ value may cause a serious invasion of privacy”.

(Wang et al., 2007, p. 257)

**CR12**  
if the user is not authorized  
then **do not reveal the association** between  
a patient and his/her HIV result

Continued on next page
“Several items in this system may need protection: (i) User’s private information as known to the local host (ii) Account information provided by User to the applet upon form submission (iii) Order information similar to (ii) (iv) The mere fact that User is engaging in a transaction with Merchant (v) Secrets concerning User’s account possessed by Acquirer such as account balance or credit limit.”

(Dam and Giambiagi, 2000, p. 235)

| CR13 | if the user is not authorized then do not reveal the association between a customer and his/her account balance result |

Continued on next page
The existence of a particular phone number in a phone book should be kept secret.

“The property to be kept secret for the example is whether a particular string, say ‘555-55’ is in the phone book. Let us denote it by secret. We want to verify that the attacker cannot infer whether the secret holds or not based on her knowledge of the program and observation of the outputs (in this case, the variable message)”.

(Cerny and Alur, 2009b, p. 175)

<table>
<thead>
<tr>
<th>CR14</th>
<th>if the user is not authorized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>then for every given ( x ),</td>
</tr>
<tr>
<td></td>
<td>do not reveal whether ( x ) is a member of the set ( S )</td>
</tr>
<tr>
<td></td>
<td>(where ( S ) contains the phone numbers in the phone book)</td>
</tr>
</tbody>
</table>

The confidentiality of the personal data stored in a smart card must be maintained.

“Due to its particular nature, a major concern of smartcards applications is to guarantee confidentiality and integrity of data”.

(Barthe and Dufay, 2005, p. 133,164)

<table>
<thead>
<tr>
<th>CR15</th>
<th>if the user is not authorized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>then do not reveal the value of ( x )</td>
</tr>
<tr>
<td></td>
<td>(where ( x ) is any personal detail stored in the smart card)</td>
</tr>
</tbody>
</table>

Continued on next page
### 5.3 Patterns of confidentiality requirements

<table>
<thead>
<tr>
<th>CR16</th>
<th>if the agent is not the card holder then do not reveal the value of $x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR17</td>
<td>if the agent is not the card holder then do not reveal whether the value $y$ is above/below a certain threshold $n$</td>
</tr>
</tbody>
</table>

**Context: e-Purse**

“In that specific example, agents who are not the card holder should not know that there is at least $v$ in the e-purse (whatever the value of $v$ is).”

“An agent should not know the exact value of some state variable”.

“An agent should not know that the value of some state variable is below/above some threshold”.

(De Landtsheer and Van Lamsweerde, 2005, p. 44)
The paper uses confidentiality requirements from Chandra and Khan (2010, p. 17), which reads, “Confidentiality protects data/information from unauthorised user access. Security demands that sensitive information should not be disclosed publicly. For example, in a bank account management system, function get balance () shows the current balance amount of user. In this case permission should be granted only for authorised user to see the information”.

(Parveen et al., 2015, p. 2)

<table>
<thead>
<tr>
<th>Context : Bank information system</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table 5.3 – Continued from previous page</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CR18</th>
<th>if the user is not authorized then do not reveal the association between a customer and his/her current balance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“This enables us to check confidentiality properties, e.g., that critical data such as credit card information are shared only with authorized partners”.</td>
</tr>
<tr>
<td></td>
<td>“a driver in trouble must be assured that information about his credit card and his location cannot become available to unauthorized users”.</td>
</tr>
<tr>
<td></td>
<td>(Lapadula et al., 2008, p. 713,714)</td>
</tr>
</tbody>
</table>

| CR19 | if the user is not authorized then do not reveal the value of $y$ (where $y$ is a credit card information of the card holder) |

Continued on next page
5.3 Patterns of confidentiality requirements

Table 5.3 – Continued from previous page

Context: Examination system

“No examinee should learn any details of the contents of the examination before the start of the examination”.
“No examinee should learn any details of the contents of any other examinees answer paper between the start of the paper and the end of the examination”.
“No examinee should learn any details of the marking until results are posted”.

(Foley and Jacob, 1995, p. 143)

```
if the user u is registered in an exam e
    and exam e has not started
then for every given x,
    do not reveal whether x is a member of the set S
(where S represents the contents of the exam e)
```

Continued on next page
Table 5.3 – *Continued from previous page*

<table>
<thead>
<tr>
<th>CR21</th>
<th>if the user $u$ currently using the system is not requesting information that belongs to him/her and exam $e$ has started but not ended then do not reveal the set $S$ (where $S$ represents the answers recorded by a user other than $u$ in the exam $e$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR22</td>
<td>if the user $u$ is registered in an exam $e$ and the results of exam $e$ has not been posted then for every given $x$, do not reveal whether $x$ is a member of the set $S$ (where $S$ represents the markings for the exam $e$)</td>
</tr>
<tr>
<td></td>
<td>The crucial information that is confidential in a computer assisted assessment and diagnosis system includes the testing number, the name of the candidates, title, score and user account amongst others. (Cao and Wang, 2009, p. 292)</td>
</tr>
<tr>
<td>CR23</td>
<td>if the user is not authorized then do not reveal the association between a candidate and his/her candidate details</td>
</tr>
</tbody>
</table>

*Continued on next page*

---

2 Cao and Wang (2009) describes a mechanism that protects the confidential data via encryption.
5.3 Patterns of confidentiality requirements

Table 5.3 – Continued from previous page

Context: Java card

The confidentiality of data in a multi-application Java card must be guaranteed as per the applet isolation principle.

if the applet \( A \) is not authorised on the applet \( B \)
then for every given \( x \),
    do not reveal whether \( x \) is a member of the set \( S \)
    (where \( S \) represents the set of data in the applet \( A \))

“Open smart cards let you download code onto cards after their issuance (postissuance)”.

“One of the main issues when deploying these applications is to guarantee to the customer that these applications will be safe i.e., that their execution will not jeopardise the smart card’s integrity or confidentiality.”.

if the applet \( A \) is not authorised on the applet \( B \)
then for every given \( x \),
    do not reveal whether \( x \) is a member of the set \( S \)
    (where \( S \) represents the set of data in the applet \( A \))
“This objective prevents the objects owned by one applet from being used by another applet without explicit sharing. The isolation between the applets covers two security properties: confidentiality and integrity. The confidentiality ensures that during its execution, an applet cannot read the information stored in the other applets”.

(Chetali and Nguyen, 2008, p. 202)

<table>
<thead>
<tr>
<th>CR26</th>
</tr>
</thead>
<tbody>
<tr>
<td>if the applet A is not authorised on the applet B</td>
</tr>
<tr>
<td>then for every given x,</td>
</tr>
<tr>
<td>do not reveal whether x is a member of the set S</td>
</tr>
<tr>
<td>(where S represents the set of data in the applet A)</td>
</tr>
</tbody>
</table>

Continued on next page
5.3 Patterns of confidentiality requirements

Table 5.3 – Continued from previous page

Context: Conference management system

The following confidentiality properties are addressed.

“A group of users learn nothing about a paper unless one of them becomes an author of that paper or a PC member at the paper’s conference.

A group of users learn nothing about a paper beyond the last submitted version unless one of them becomes an author of that paper.

A group of users learn nothing about the content of a review beyond the last submitted version before the discussion phase and the later versions unless one of them is that review’s author.

The author’s learn nothing about the discussion of their paper”.

(Kanav et al., 2014, p. 168)

if the user \( u \) is not the author of the paper \( x \)

and \( u \) is not a PC member at the conference

where \( x \) is submitted

then do not reveal the association between

the paper \( x \) and any details related to that paper

Continued on next page
“Confidentiality is important here because Pubs want to make sure that only paying customers have access to the quotes. We say that a CBPS system provides publication confidentiality if Brokers can neither identify the content of the messages published by Pubs nor infer the distribution of attribute values of the message”.

(Ion et al., 2010, p. 134)

if the user $u$ is a Broker

then do not reveal the set $S$

(where $S$ is the set of contents in the messages published by a publisher)

“We say that a CBPS system provides subscription privacy if Brokers can neither identify what subscriptions $\text{Subs}$ made nor relate a set of subscriptions to a specific $\text{Sub}$”.

(Ion et al., 2010, p. 134)

if the user $u$ is a Broker

then do not reveal the association between a subscriber and his/her set of subscriptions

Continued on next page
5.3 Patterns of confidentiality requirements

Table 5.3 – Continued from previous page

Context: Military information system

“Information about mission plans, strategy, and deployment of troops must remain confidential on the C4I computer system and networks of all type”.

“If an adversary gets access to the logistics information stored in a file or database may be used to misguide the army commander in the deployment of troops”.

(Alghamdi et al., 2010, p. 131)

if the user \( u \) is not authorised
then do not reveal the set \( S \)
(\( S \) is the set of logistics information)

Context: Information retrieval system

The confidentiality of the search query as well as the contents of the retrieved documents should be maintained.

“The proposed method maintains the confidentiality of the query as well as the content of retrieved documents”.

(Swaminathan et al. 2007, p. 12)

if the user is not authorized
then do not reveal the value of \( x \)
(\( x \) is the search query)

if the user is not authorized
then do not reveal the set \( S \)
(\( S \) is the set of contents in the retrieved documents)

Continued on next page
It is important to identify any commonalities that may exist in the formalisation of the patterns in Table 5.2 using BCF in Circus. Such commonalities may be considered as generalized patterns, as they exist across more than one confidentiality pattern. However, before a confidentiality pattern can be formalized, any vagueness within its description must be removed. In Table 5.3, CP3 is identified as having a vague description. The next section discusses the possible different interpretations of CP3 that can be formalised.
5.4 Subtleties in formalizing generic patterns of confidentiality requirements using BCF in Circus

Most of the confidentiality requirements presented in Table 5.3 are not described in a clear and concise manner. Therefore, the patterns derived from these confidentiality requirements can be formalised in more than one way. The subtleties with these different formal definitions translate to different upper limits on the information that can be disclosed to an unauthorised audience.

Such subtleties may be required during the system design stage where there are often trade-offs between functional and non-functional requirements (Yu and Liu, 2001, p. 185; Santen, 2006, p. 154). For example, rather than concealing every single bid in a set of bids $Bids$ it will be acceptable to conceal at least 1 bid. By doing so, the uncertainty about the highest bid in $Bids$ can be maintained as the the concealed bid may be the highest bid.

In this section, we discuss possible different formal definitions of inequality between two sets, as required by pattern CP3. The pattern CP3 states:

\[
do\ not\ reveal\ the\ set\ S
\]

Even though the pattern demands that the set $S$ must not be revealed, the specifics of how much information about $S$ must be concealed has not been defined in the original literature. Table 5.4 lists some possible ways in which the inequality between two sets may be formalized using BCF in Circus.
A systematic literature search for case study material

Possible ways of specifying set inequality

<table>
<thead>
<tr>
<th>Possible ways of specifying set inequality</th>
<th>Syntax in BCF in Circus</th>
</tr>
</thead>
<tbody>
<tr>
<td>do not reveal the exact composition of the set $S$</td>
<td>$S \neq \tilde{S}$</td>
</tr>
<tr>
<td>do not reveal any elements of the set $S$</td>
<td>$S \cap \tilde{S} = \emptyset$</td>
</tr>
<tr>
<td>do not reveal at least $n$ elements from the set $S$</td>
<td>$(#S + n) \geq #\tilde{S}$</td>
</tr>
</tbody>
</table>

Table 5.4: Possible ways of specifying an inequality between two sets $S$ and $\tilde{S}$

The subtleties listed in Table 5.4 is demonstrated using a set of tuples of the function $S$ and $\tilde{S}$ as shown in Figure 5.2. Each function maps a person identifier to his/her salary. Each mapping $(x, y)$ is a tuple, such as (Alex, 1000) in Figure 5.2a, (Fred, 400) in Figure 5.2c and (Casey, 1000) in Figure 5.2h. Table 5.5 shows the differences between the function map in Figure 5.2a and each of the the function maps in Figure 5.2b to Figure 5.2i.
5.4 Subtleties in formalizing generic patterns of confidentiality requirements using BCF in Circus

Figure 5.2: A function map for a function $S$ and possible function maps for different variants of its twin function $\tilde{S}$
A systematic literature search for case study material

<table>
<thead>
<tr>
<th>Function Map</th>
<th>$\text{dom}(S)$</th>
<th>$\text{ran}(S)$</th>
<th>$S \cap \tilde{S} = {}$</th>
<th>$S \neq \tilde{S}$</th>
<th>$(#S + 1) \geq #\tilde{S}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td>$=$</td>
<td>$=$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
</tr>
<tr>
<td>(c)</td>
<td>$\neq$</td>
<td>$\neq$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>(d)</td>
<td>$=$</td>
<td>$=$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
</tr>
<tr>
<td>(e)</td>
<td>$\neq$</td>
<td>$\neq$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
</tr>
<tr>
<td>(f)</td>
<td>$\neq$</td>
<td>$\neq$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>(g)</td>
<td>$\neq$</td>
<td>$\neq$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>(h)</td>
<td>$\neq$</td>
<td>$\neq$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>(i)</td>
<td>$\neq$</td>
<td>$\neq$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
</tbody>
</table>

Table 5.5: A comparison of equality in function maps from Figure 5.2

**NOTE:**
- The $n$ in $(\#S + n) \geq \#\tilde{S}$ has been instantiated to 1, for the purposes of demonstration in Table 5.5.
- The $\checkmark$ denotes that the property defined in the column heading is satisfied while $\times$ denotes that the property is not satisfied.
5.4 Subtleties in formalizing generic patterns of confidentiality requirements using BCF in Circus

5.4.1 Scenarios where different subtleties with inequality between two sets may satisfy a confidentiality requirement

The Table 5.6 shows how effective each possible way of specifying inequality between two sets is against confidentiality requirements, that have been identified in Section 5.3. The set $S$ in Table 5.6 may be any expression that results in a set. The ✓ denotes that the property is satisfied by the type of implementation.

<table>
<thead>
<tr>
<th>Set to be kept secret ($S$)</th>
<th>Specify a set inequality where we conceal the exact contents of the set</th>
<th>Conceal all the elements of the set</th>
<th>Conceal at least $n$ elements from the set</th>
</tr>
</thead>
<tbody>
<tr>
<td>set of salaries in a salary band with $x$ number of employees</td>
<td>may reveal $M$ where $M \subseteq S$. $M$ might contain $x - 1$ salaries. Using $M$ and $x$ we can compute the sum of the hidden salaries thereby revealing the only hidden salary</td>
<td>✓</td>
<td>✓ (if $n &gt; 1$)</td>
</tr>
<tr>
<td>answers of a candidate $c$</td>
<td>May reveal all answers except 1</td>
<td>✓</td>
<td>May reveal ($#S - n$) answers from $S$</td>
</tr>
<tr>
<td>set of medicines in a patient's prescription history</td>
<td>may reveal $M$ where $M \subseteq S$. $M$ might contain a specific medicine implying a specific medical condition of the patient</td>
<td>✓</td>
<td>may reveal $M$ where $M \subseteq S$. $M$ might contain a specific medicine implying a specific medical condition of the patient</td>
</tr>
</tbody>
</table>

Table 5.6: How confidentiality properties are addressed by subtleties in set inequality.

3 Chivers (2006, p. 89) discusses a scenario where the relationship between an employee and his/her salary is confidential. Lunt (1989, p. 104) states that if the set of employees in a group is small enough then the association between a particular employee and his/her salary can be inferred.

4 Foley and Jacob (1995, p. 143) state that the answers recorded by an examinee must not be revealed to any other examinee during the examination.

5 Onabajo and Jahnke (2006a, p. 844) state that restricting the visibility of the prescription history is a patient’s confidentiality requirement.
5.5 Identifying and formalizing generic patterns of confidentiality

This section presents a discussion on extracting generalized patterns from formal specifications of confidentiality requirement patterns that were identified in Section 5.3. The approach involves identifying commonalities that may exist in formalizing these patterns and deriving generalized patterns of confidentiality requirements based on these commonalities. These generalized patterns can also be categorized as property specification patterns since these patterns have been extracted from specifications of properties.

“A property specification pattern describes the essential structure of some aspect of a system’s behaviour and provides expressions of this behaviour in a range of common formalisms” (Dwyer et al., 1998, p. 9).

Konrad et al. (2003) provide a template for specifying security patterns. Further, he suggests a systematic process for simulating as well as model checking the resulting models. Figure 5.3 shows the steps that were followed in deriving these generalized patterns.

| CP1 | The confidentiality pattern CP1 states:

| do not reveal the relation between x and y in S |

Here, the objective of CP1 is to maintain the secrecy regarding the existence of the tuple or pair \((x, y)\) in the set \(S\). Binary relations and functions from the Z notation are used to model such a set of tuples (Spivey, 1989, p. 27).

“A relation is defined to be a set of pairs. A function is a particular form of relation, where each domain element has only one corresponding range element” (ISO/IEC, 2002).
5.5 Identifying and formalizing generic patterns of confidentiality

- **CP1**: Do not reveal the relation between $x$ and $y$ in set $S$
- **CP2**: Do not reveal whether $x$ is a member of $S$
- **CP3**: Do not reveal the set $S$
- **CP4**: Do not reveal the exact value of $x$
- **CP5**: Do not reveal whether the value $x$ is lower/higher than a given threshold $n$

**Generalized patterns of confidentiality properties**

Figure 5.3: Deriving generalized patterns from patterns of confidentiality requirements
Recall from Section 2.5 that Circus combines the Z notation with other notations and techniques. The following are two formalisations of CP1 depending on whether the set $S$ is a relation or a function.

**If $S$ in CP1 is a non-functional relation**

Where $S$ is a non-functional relation, one possible formalisation of CP1 using BCF in Circus can be:

$$(x, y) \in S \Rightarrow (\tilde{x}, \tilde{y}) \not\in \tilde{S}$$

(5.1)

The formalisation used in Equation (5.1) for specifying confidentiality in set $S$ is specific to tuples. However, the formalisation is concerned with maintaining the secrecy of an element of in a given set. This is similar to the requirement pattern captured in CP2.

**If $S$ in CP1 is a function**

Where $S$ is a function, one possible formalisation of CP1 using BCF in Circus can be:

$$S(x) \neq \tilde{S}(\tilde{x})$$

(5.2)

The formalisation in Equation (5.2) can be used in BCF in Circus to conceal the exact value of $S(x)$. This fulfils the confidentiality requirement pattern in CP1 where CP1 is concerned with maintaining the secrecy of the tuple $(x, S(x))$. Here, $S(x)$ represents the variable $y$ is CP1. This is similar to the requirement pattern captured in CP4.
The confidentiality pattern CP2 states:

\[ x \notin S \Rightarrow \tilde{x} \notin \tilde{S} \] (5.3)

While Equation (5.3) is a possible formalisation of CP2, Equation (5.3) is also a more generalized formalisation of Equation (5.1). The following generalized pattern can be derived from Equation (5.3).

**Generalized pattern GP1**: do not reveal whether x is in S

The confidentiality pattern CP3 states:

do not reveal the set S

In Section 5.4, the need for introducing some subtly different interpretations of CP3 were discussed. Further, formalisations for those interpretations were presented and how they address different confidentiality properties were discussed. Due to the lack of clear descriptions of confidentiality properties under CP3, it was important that those interpretations were available for accurately formalising the confidentiality properties. For this reason, the subtly different interpretations of CP3 can be adopted as generalised patterns for maintaining the secrecy of a set. Here, the contents of Table 5.4 in Table 5.7 are being duplicated for easy reference. Further, a unique identifier for each possibility listed in Table 5.7 has been introduced.
A systematic literature search for case study material

<table>
<thead>
<tr>
<th>id</th>
<th>Possible ways of specifying set inequality</th>
<th>BCF in Circus</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>do not reveal the exact composition of the set $S$</td>
<td>$S \neq \tilde{S}$</td>
</tr>
<tr>
<td>S2</td>
<td>do not reveal any elements of the set $S$</td>
<td>$S \cap \tilde{S} = \emptyset$</td>
</tr>
<tr>
<td>S3</td>
<td>do not reveal at least $n$ elements from the set $S$</td>
<td>$(#S + n) \geq #\tilde{S}$</td>
</tr>
</tbody>
</table>

Table 5.7: Possible ways of specifying an inequality between two sets $S$ and $\tilde{S}$

The formalisation of S1 in Table 5.7 is similar to the requirement pattern captured in CP4 where CP4 is a more generic formalisation of S1.

The following generalized pattern can be derived from the formalisation of S2 in Table 5.7.

**Generalized pattern GP2**: *do not reveal any elements of the set $S$*

The following generalized pattern can be derived from the formalisation of S3 in Table 5.7.

**Generalized pattern GP3**: *do not reveal at least $n$ elements from the set $S$*

The confidentiality pattern CP4 states:

do not reveal the exact value of $x$

Here, the objective of CP4 is to maintain the secrecy of the exact value of $x$. One possible formalisation of CP4 using BCF in Circus can be:

$$x \neq \tilde{x}$$ (5.4)
Pattern CP4 with definition “do not reveal the exact value of $x$” is a weak pattern because even after CP4 is applied, sometimes critical information may be revealed, without revealing the exact value that is required to be kept secret. For example, consider the requirement where the exact value of a customer bank account is required to be a secret. If the balance $bal$ of a customer called $bob$ is $500$ and if a user has access to the binary representation of the state variable $bal$ except the last two bits, then the user can infer that $bal$ is definitely between 500 and 503. In this scenario, even though CP4 has been satisfied through bit obfuscation, the inference made by the user might not be acceptable. This is because, through inference, the user is able to learn $bal$ to a high degree of accuracy. Table 5.8 represents the bit representation of numbers from 500 to 503.

<table>
<thead>
<tr>
<th>values</th>
<th>256</th>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>501</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>502</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>503</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.8: Bit representations of bank balances between 500 and 503

When using bit obfuscation for confidentiality, as discussed above, the minimum number of bits that needs to be obfuscated to maintain the secrecy of a value will depend on the confidentiality requirement, where the confidentiality requirement must define the degree to which the value should be kept confidential.
A systematic literature search for case study material

Essentially, both Equation (5.4) and Equation (5.2) are concerned with maintaining the secrecy of the exact value of an expression $expr$, where $expr \equiv x$ in Equation (5.4) and $expr \equiv f(x)$ in Equation (5.2). The expression $\overline{expr}$ is derived by renaming each state variable $y$ in $expr$ with its twin counterpart $\tilde{y}$ as in $expr[y/\tilde{y}]$. Equation (5.5) represents a more generalised formalisation of CP4.

$$expr \neq \overline{expr} \quad (5.5)$$

We may derive the following generalized pattern from Equation (5.5).

**Generalized pattern GP4**: do not reveal the exact value of any expression $expr$

The confidentiality pattern CP5 states:

do not reveal whether the value of $x$ is lower/higher than a given threshold $n$

The objective of CP5 is to maintain the secrecy of the upper bound and the lower bound of a value against some given thresholds. The pattern CP5 has been extracted from the paper (De Landtsheer and Van Lamsweerde, 2005, p. 4). Infact, this pattern contains three generic patterns of confidentiality properties which they have formalised separately in (De Landtsheer and Van Lamsweerde, 2005, p. 4) using epistemic logic. Following are possible formalisations of those patterns using BCF in Circus.

**lower bound** do not reveal whether $x$ is lower than the threshold $m$

$$x \in \{y \mid y < n\} \Rightarrow \tilde{x} \notin \{y \mid y < n\} \quad (5.6)$$
5.6 Generalized patterns of confidentiality requirements

**upper bound** do not reveal whether $x$ is higher than the threshold $m$

$$x \in \{ y \mid y > n \} \Rightarrow \tilde{x} \notin \{ y \mid y > n \}$$ (5.7)

**between** do not reveal whether $x$ is between $m$ and $n$

$$x \in m..n \Rightarrow \tilde{x} \notin m..n$$ (5.8)

The formalisation used in Equation (5.8) specifies that the knowledge that the value of $x$ falls within a certain range must be confidential. In general, the formalisation is concerned with maintaining the secrecy of an element in a given set, which in this case is the set of values in the range $m..n$. This is similar to the requirement pattern captured in CP2.

### 5.6 Generalized patterns of confidentiality requirements

A literature-related set of generalized patterns of confidentiality requirements related to data leakage has been derived through research carried out in this chapter. The set of generalized patterns derived are shown in Table 5.9. The table further shows how to specify each generalized pattern of confidentiality requirement using BCF in Circus.

<table>
<thead>
<tr>
<th>Pattern id</th>
<th>Pattern description</th>
<th>How to specify a property with the pattern using BCF in Circus</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP1</td>
<td>do not reveal whether $x$ is in $S$</td>
<td>$x \in S \Rightarrow \tilde{x} \notin \tilde{S}$</td>
</tr>
<tr>
<td>GP2</td>
<td>do not reveal any elements of the set $S$</td>
<td>$S \cap \tilde{S} = {}$</td>
</tr>
<tr>
<td>GP3</td>
<td>do not reveal at least $n$ elements from the set $S$</td>
<td>$(# S + n) \geq # \tilde{S}$</td>
</tr>
<tr>
<td>GP4</td>
<td>do not reveal the exact value of any expression $expr$</td>
<td>$expr \neq \tilde{expr}$</td>
</tr>
</tbody>
</table>

Table 5.9: A catalogue of generalized patterns of confidentiality requirements

---

181
5.7 Confidentiality requirement patterns in literature

Gurses et al. (2005) list a set of attributes that describe a confidentiality requirement. They include the owner of the confidentiality data, degree of agreement between the stakeholders of the data, the counter-stakeholder or the role from whom the owner wants to hide the data, the information to which the confidentiality requirement refers to, the owners rationale for the confidentiality requirement, the temporal range or how long the confidentiality requirement must be in place and the context or the cluster of system functionality where the requirement must be implemented.

De Landtsheer and Van Lamsweerde (2005) presents a catalogue of patterns on confidentiality along two dimensions. One is concerned with the degree of appropriate knowledge that must be kept confidential. The second is concerned with timing according to which the knowledge must be kept confidential. Similar to De Landtsheer and Van Lamsweerde (2005), BCF in Circus helps to formally specify the degree of confidentiality that must be maintained about a piece of data. BCF in Circus can address timing related confidentiality properties in terms of the order of states.

Both De Landtsheer and Van Lamsweerde (2005) and the generic patterns proposed in this thesis share the pattern “do not reveal the exact value of a variable”. Further, De Landtsheer and Van Lamsweerde (2005) proposed the pattern Confidential lower/upper bound which states that “An agent should not know that the value of some state variable is below/above some threshold”, which has been dissected into two generic patterns and included in the catalogue of generic patterns in Table 5.9. Also, De Landtsheer and Van Lamsweerde (2005) proposed the pattern Fully confidential value which states “An agent should not be able to infer any property about the value of some state variable”. This pattern is just a conjunction of many other patterns proposed in his catalogue.

The timing related confidentiality patterns by De Landtsheer and Van Lamsweerde (2005, p. 45) include Confidential now that states that “In the current state, an agent should not know about some state variable”; Confidential until expiration date that states that “An agent should not know about some state variable until some delay has expired”;
Confidential unless/until condition states that “An agent should not know about some state variable unless or until some condition becomes true”; Confidential forever that states that “An agent should never know about some state variable”. Both confidentiality until a condition is met (see Section 6.2.4) and confidentiality forever are properties that can also be specified using BCF in Circus.

5.8 Patterns in software engineering

In the early 1990s, there was an interest in the software engineering community to identify situations in which design knowledge could be represented and shared between practitioners (Fowler, 1997, p. 5). Engineers explored the possibility of using the pattern language approach by Alexander et al. (1977) to represent knowledge that can possibly be shared. The OOPSLA workshops (Meyrowitz, 1986) provided an early platform for such discussions. Currently, the Pattern Languages of Programs (PLoP) conference series (The Hillside Group, 2017) is one of the annual platforms for such discussions. Alexander et al. (1977) introduced the notion of pattern languages in the context of building architecture where the elements of the language were entities called patterns. He defined the term ‘pattern’ as follows.

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” (Alexander et al., 1977, p. x)

In the field of software engineering, pattern based approaches has been discussed with respect to various stages of a system development process. Some of the different kinds of patterns used in a system development process include music design patterns (Bouaziz et al., 2011), design patterns (Andrews et al., 2008; Angkasaputra and Pfahl, 2004)

---

6 “Capturing expert-knowledge and providing proven solutions for recurring problems is the basic idea of software (design) patterns.” (Schumacher, 2001, p. 4)
Patterns have also been utilized in formal methods technology. Freitas and Whiteside (2014) proposed proof tactics to facilitate less experienced proof engineers to attempt at difficult lemmas, reducing the proof cost and effort in the process.

5.9 Limitations of the study

There are a number of inherent limitations in the process adopted in compiling the catalogue of confidentiality properties. Firstly, during the literature search, it was found that often the search results included papers that discussed legal aspects of privacy and confidentiality while those papers had no discussions on confidentiality requirements. Combining this reality with our inclusion criteria where we select only the first 20 results from each “indexing service × keyword phrase” pair meant that papers discussing confidentiality requirements might have been pushed further down the stack past the first 20 results.

The interpretations proposed for some vague confidentiality requirements may be reasonable. However, those interpretations are still hand-crafted and hence need further evidence to strengthen the argument that the patterns derived from such interpreted confidentiality requirements are important. Rather than reading all 490 search results word for word, the effort was concentrated on locating any discussions of
5.9 Limitations of the study

Confidentiality properties both in the abstract as well as around each paragraph where the keyword ‘confidential’ was located.

The search functionality in Mendeley software was used to locate keywords in each paper. Such an approach was necessary because of the limited time frame available for this Ph.D. However, some useful discussions of confidentiality properties that did not contain the specific word ‘confidentiality’ might have been missed, as a result of following this process.

Finally, documents that cannot be indexed by Mendeley would not have been included in the search results that Mendeley produced. One or more potential discussions on confidentiality properties might have been overlooked because of this. It must be noted here that the decision to use Mendeley was a personal choice and has to do with its ease of use for the author.
5 A systematic literature search for case study material

5.10 Summary

The main objective of this chapter is to derive a set of generic patterns of confidentiality requirements from literature. Such a set is required to fill a vacuum in terms of a benchmark of confidentiality requirement patterns that can be used in testing tools that analyse systems for confidentiality.

In this chapter:

• a catalogue of confidentiality requirements from literature was compiled.
• a literature related set of patterns of confidentiality requirements were derived.
• generic patterns of confidentiality requirements were identified and formalized.
• existing literature specific to patterns of confidentiality requirements, as well as the emergence and use of patterns in software engineering in general was discussed.
6 Evaluation of mechanisation

6.1 Introducing

The objective of this chapter is to demonstrate that the CFAT tool developed under this research is practically applicable. This chapter presents an evaluation of the mechanisation of BCF in Circus described in Chapter 3. The evaluation of the mechanisation will involve analysing systems with different patterns of confidentiality requirements, that has been identified in Section 5.2.

In Chapter 5, a literature search was carried out to identify case studies that discusses systems with a confidentiality requirement. However, none of the papers identified in Chapter 5 contained a full system description or a full formal specification of a system, discussed in that paper. Further, none of the papers presented a confidentiality analysis using BCF. Papers by De Landtsheer and Van Lamsweerde (2005) and Howitt (2008) describe confidentiality analysis of systems based on other modelling techniques. In Section 2.2, a discussion of the differences between those techniques and the approach adopted by BCF in Circus (Banks 2012) has been presented.

In the absence of a literature-supported full formal specifications of systems, hand-crafted possible system requirement specifications for systems had to be developed.
We will work through an example of each identified pattern where we will:

1. introduce a hand-crafted set of requirements for a system.
2. illustrate the functional structure of the system using a use-case diagram.
3. construct a Circus specification of the system manually.
4. specify the confidentiality requirements of the system using BCF in Circus.
5. construct a CFAT specification of the system based on the Circus specification and confidentiality requirement.
6. analyse the CFAT specification using the mechanisation tool to check the consistency of the requirements defined in the system.

There is no formal mapping between the use case and the formal model of the system captured in the Circus specification. The back propagation laws of BCF in Circus (Banks, 2012, p. 138) does not contain a back propagation law that can be used on parallel processes. And so, currently BCF in Circus can only be used to evaluate sequential access systems. Therefore, in all cases it is assumed that the systems modelled in this chapter are single user access.

The confidentiality requirements used in the analysis carried out in this chapter are deduced from the discussion of confidentiality in the published papers. This exercise has been carried out in Chapter 5, and the deduced confidentiality properties are listed in Table 5.3. An informal approach has been used to state the requirements of each system in natural language and then a Circus specification has been presented for that system.

In order to analyse each identified confidentiality requirement pattern CP, we will model a system Y described in one of the referenced papers RP in Table 5.3. Each confidentiality requirement CR stated for Y in Table 5.3 has been derived from the confidentiality properties discussed in RP. A CP represents a generalized definition for each CR. Table 5.2 groups all confidentiality requirements based on the CP which they
6.1 Introducing

The concept of BCF is that at least one alternative similar observation exists for each observation of a state variable in the original system. BCF notation uses the decoration tilde (\(\tilde{\cdot}\)) as in \(\tilde{S}\) to specify a similar state space of a given state schema \(S\). However, this notation is not supported by the CZT tool \cite{Malik2005}, which is used for type and syntax checking the Circus specification of the system.

A similar state space of a state schema \(S\) can be specified in the Circus notation using \LaTeX{} by subscripting the schema name with a number such as by writing \(S_9\). Each variable \(x\) in the new schema \(S_9\) can then be referenced using the notation \(x_9\). It must be highlighted that in all the formal statements presented in this chapter, \(x_9\) has been used to represent each state variable \(x\) in the twin state space.

It must be noted here that the subscript 9 is an arbitrary choice that does not clash with anything else as long as the restrictions on the naming conventions are in place as discussed on Page 100. It must also be noted that using the subscript 9 is a coding trick to enable syntax and type checking by the CZT tool. It is important to declare this only as a coding trick because the semantics of this coding trick does not correctly capture the BCF concept ‘at least one alternative similar observation exists for each observation of a state variable in the original system’.

Each case study analysis in this chapter involves using the mechanisation to verify the consistency of a BCF in Circus based formal model of \(Y\) against an associated CR.

The BCF in Circus based formal specification of a system contains syntactic decorations that are not supported by both CZT as well as the Isabelle theorem prover. Therefore, for the purposes of type and syntax checking the formal specification using CZT and for theorem proving using the Isabelle theorem prover, syntactic renaming of the twin state variables in the formal specification are required. Recall that Section 3.2.4 in
6 Evaluation of mechanisation

Page 100 discusses all the variables of the system specification that must be renamed by the user during the submission of the system specification to the mechanised tool as well as that will be renamed when files are generated by the mechanised tool. The explanation on Page 189 recalls how the twin or shadow state space is encoded in a Circus specification generated by the mechanised tool so that the specification can be type and syntax checked using CZT.

6.2 Mechanised analysis of confidentiality patterns

In Chapter 5, a set of patterns has been extracted from the confidentiality requirements described in a systematically compiled literature set. In this section, we will discuss the mechanised analysis of systems having confidentiality properties with one or more of those patterns.

The following steps will be carried out in each case study.

• Present the system requirement specification of the case study system.

• Develop and describe a possible formal specification of the functionality requirements of the case study system using the Circus notation.

• Discuss a possible formalisation of the confidentiality requirement(s) of the case study system using the extended Circus notation.

• Describe how the mechanisation in Chapter 3 can be used to analyse the formal specification.

• Present and discuss the results of the mechanised analysis of the formal specification that integrates the confidentiality requirement.

The above steps will be discussed in detail for the first pattern. For the subsequent patterns, the pattern and subsequently the results from analysing a system with a confidentiality property having that pattern will be present. Recall from Section 6.1 that BCF in Circus only supports sequential access systems and therefore in all the
analysis carried out in this chapter, systems will be modelled with sequential access. Further, recall that all the derived confidentiality patterns are listed in Table 5.2 and all the identified confidentiality requirements are listed in Table 5.3.

6.2.1 Mechanised analysis of a system having a confidentiality property that reflects pattern CP1

The first case study to evaluate the mechanisation of BCF in Circus involves analysing a system with a confidentiality property that reflects the pattern CP1. The confidentiality requirement CR18 has been chosen as an adhoc choice for this analysis. The confidentiality pattern CP1 states:

\[ \text{do not reveal the relation between } x \text{ and } y \text{ in } S \]

For this analysis, a simple accounts system for a bank will be modelled. The system will be called the Bank information system. The confidentiality property modelled in this system has been borrowed from discussions by both \cite{Parveen et al. 2015} and \cite{Lapadula et al. 2008}.

6.2.2 System requirement specification - Bank information system

A hand-crafted system requirement specification of a fictitious Bank information system is given below. The functions listed in the Bank information system are borrowed from commonly used use cases by \cite{Eckel 2005}, \cite{Skon 2016} and \cite{Pearce 2017} for academic discussions in the context of a Bank information system.

Organisational structure

A \textit{bank balance} is represented by a \textit{number}.

A \textit{customer} is a \textit{person}.
The structures of a system requirement specification may be mapped onto structures of the *Circus* notation. Table 6.1 shows such a mapping. Every system requirement specification presented in this section has been structured accordingly for ease of readability. It must be noted that each developed *Circus* specification will be just one possible implementation of the original system.

<table>
<thead>
<tr>
<th>System requirement specification</th>
<th>Structures of the <em>Circus</em> notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organisational structure (entities and attributes)</td>
<td>data types, data objects, global constants</td>
</tr>
<tr>
<td>Organisational rules and regulations</td>
<td>state invariants</td>
</tr>
<tr>
<td>Operations performed in the organisation</td>
<td>actions</td>
</tr>
<tr>
<td>Who can perform the functions</td>
<td>channelsets (representing user roles)</td>
</tr>
<tr>
<td>Which staff are included in each user role</td>
<td>state invariants that define elements of sets</td>
</tr>
</tbody>
</table>

Table 6.1: Mapping a system requirement specification to structures in a *Circus* specification
6.2 Mechanised analysis of confidentiality patterns

The company uniquely identifies the person who is the manager of the company.

The company uniquely identifies the person who is the current_user of the system.

The company maintains a list that contains every customer of the company.

The company maintains a list that contains every cashier of the company.

The company maintains a list that contains every user of the company.

The company maintains a bank_balance for each existing customer.

Organisational rules

The same person cannot be a customer and a cashier and the manager.

Every user must be either a customer or a cashier or the manager.

The current_user must be included in the user list.

User roles in the system

The banking_staff user role includes the manager and all users who are cashiers.

The cashier user role includes all users who are cashiers.

The customer user role contains all users who are customers.

Operations, user roles and permissions

Table 6.2 lists all the system operations, and the specific permissions on those operations by user role. Figure 6.1 presents a tripartite graph that shows the user-to-role and role-to-permission assignments in the Bank information system. User roles and how it is utilised in BCF in Circus for analysing data leakage in a system is discussed later in Figure 6.3.
### Evaluation of mechanisation

<table>
<thead>
<tr>
<th>Operation identifier</th>
<th>User roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find_own_balance</td>
<td>Banking Staff, Cashier, Customer</td>
</tr>
<tr>
<td>Find_customer_balance</td>
<td>Banking Staff, Cashier, Customer</td>
</tr>
<tr>
<td>Record_new_customer</td>
<td>Banking Staff, Cashier, Customer</td>
</tr>
<tr>
<td>Deposit_to_customer</td>
<td>Banking Staff, Cashier, Customer</td>
</tr>
<tr>
<td>Withdraw_from_customer</td>
<td>Banking Staff, Cashier, Customer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Find_own_balance</th>
<th>Banking Staff, Cashier, Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find the account_balance of oneself.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Find_customer_balance</th>
<th>Banking Staff, Cashier, Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find the account_balance of any customer.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record_new_customer</th>
<th>Banking Staff, Cashier, Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record the personal_details of a new customer.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deposit_to_customer</th>
<th>Banking Staff, Cashier, Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculate and Update the bank balance of a particular customer which is derived using the customers existing bank balance and value deposited by the customer.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Withdraw_from_customer</th>
<th>Banking Staff, Cashier, Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculate and Update the bank balance of a particular customer which is derived using the customers existing bank balance and value withdrawn by the customer.</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Roles and Permissions Matrix of the Bank information system
### 6.2 Mechanised analysis of confidentiality patterns

<table>
<thead>
<tr>
<th>user</th>
<th>user role</th>
<th>permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cashier</td>
<td>Cashier</td>
<td>Record_new_customer</td>
</tr>
<tr>
<td>Manager</td>
<td>Banking staff</td>
<td>Deposit_to_customer</td>
</tr>
<tr>
<td>Customer</td>
<td>Customer</td>
<td>Withdraw_from_customer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Find_customer_balance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Find_own_balance</td>
</tr>
</tbody>
</table>

Figure 6.1: Tripartite graph of user-to-role and role-to-permission assignments in the Bank information system

Figure 6.2 presents a graphical summary of the system requirement specification of the Bank information system using a use case diagram.

![Use case diagram for the Bank information system](image)

Figure 6.2: Use case diagram for the Bank information system
6.2.3 Formal specification - Bank information system

The actions in Table 6.3 addresses the operations of the Bank information system and reflects the system requirement specification of the Bank information system. The basic type in Table 6.5, state variables in Table 6.6 and state invariants in Table 6.7 reflect organisational structures that are described in the system requirement specification of the Bank information system in Section 6.2.2.

It is important to note that the action Init in Table 6.3 is not in the use case diagram in Figure 6.2. The action Init initializes the state variable that records the identifier of the customer whose information is being requested from the system. This is an internal initialization function.

<table>
<thead>
<tr>
<th>Action</th>
<th>Operation performed by the action</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewAccount</td>
<td>Allows a cashier to create a customer bank account in the system.</td>
</tr>
<tr>
<td>DepositMoney</td>
<td>Allows a cashier to deposit money to an existing bank account.</td>
</tr>
<tr>
<td>WithdrawMoney</td>
<td>Allows a cashier to withdraw money from an existing bank account.</td>
</tr>
<tr>
<td>GetMyBalance</td>
<td>Allows a customer to view his/her own account balance. The customer provides his/her person identifier which the operation uses to retrieve the account balance.</td>
</tr>
<tr>
<td>GetAnyCustBalance</td>
<td>Allows a cashier to view the account balance of any customer.</td>
</tr>
</tbody>
</table>
6.2.3.a User roles

As detailed on page 201, user roles are modelled in BCF in Circus using channelsets. Table 6.2 includes the user roles that are assumed as common in a typical Bank information system. These user roles in the specification of the Bank information system has been modelled using channelsets as shown in the Circus specification of the system in Figure 6.10 and these channelsets are listed in Table 6.4 with their descriptions.

Table 6.4: Bank information system - Description of the user roles

<table>
<thead>
<tr>
<th>channelset as user role</th>
<th>Functions allowed for the user role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>a user role where users have the right to view their own bank balances</td>
</tr>
<tr>
<td>Cashier</td>
<td>a user role where users have the right to create a new customer account with the information provided by the customer, carry out a deposit or a withdraw transaction on any customer account and also find the balance of any customer account.</td>
</tr>
<tr>
<td>BankingStaff</td>
<td>a user role where users have the right to view the balance of any customer account.</td>
</tr>
</tbody>
</table>
6 Evaluation of mechanisation

6.2.3.b Types

A basic type that contains the identifiers of all the possible people in the system is defined.

Table 6.5: Bank information system - Description of the basic types

<table>
<thead>
<tr>
<th>Basic type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERSON</td>
<td>The set of all possible person identifiers.</td>
</tr>
</tbody>
</table>

6.2.3.c State variables

The following are organisational components which we believe are common in a banking environment. A state variable has been declared to represent each organisational component within the system specification. The description of these state variables are included in Table 6.6.

Table 6.6: Bank information system - Description of the state variables

<table>
<thead>
<tr>
<th>State variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance</td>
<td>A function that identifies the balance of a customer, if any.</td>
</tr>
<tr>
<td>managers</td>
<td>The set of identifiers of the managers.</td>
</tr>
<tr>
<td>cashiers</td>
<td>The set of identifiers of the cashiers.</td>
</tr>
<tr>
<td>customers</td>
<td>The set of identifiers of the customers.</td>
</tr>
<tr>
<td>loggedIn</td>
<td>The set of identifiers of the users who are logged into the system.</td>
</tr>
</tbody>
</table>
6.2 Mechanised analysis of confidentiality patterns

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loginUser</td>
<td>The identifier of the person who is currently using the system.</td>
</tr>
<tr>
<td>reqCustomer</td>
<td>The identifier of the person about whom information is being requested from the system.</td>
</tr>
</tbody>
</table>

6.2.3.d State invariants

We assume certain system constraints that must be respected throughout the life of the system. These constraints reflect the relevant organisational rules that we believe are typical in an accounts system in a bank. These constraints are defined as state invariants in the system specification.

Table 6.7: Bank information system - Description of the state invariants

<table>
<thead>
<tr>
<th>State invariants</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loginUser ∈ loggedIn</td>
<td>The current user must be from the set of users logged into the system.</td>
</tr>
<tr>
<td>loggedIn ⊆ (customers ∪ cashiers ∪ managers)</td>
<td>The current user of the system must either be from the group of customers, the group of cashiers or the manager.</td>
</tr>
<tr>
<td>(customers ∩ cashiers) = {}</td>
<td>The same person cannot be a customer and a cashier at the same time.</td>
</tr>
<tr>
<td>(customers ∩ managers) = {}</td>
<td></td>
</tr>
</tbody>
</table>

1 A state invariant of a system is a property that holds in every reachable state of the system (Kirby et al., 1999, p. 110).
6 Evaluation of mechanisation

The same person cannot be a customer and the manager at the same time.

\[(\text{cashiers} \cap \text{managers}) = \{\}\]

The same person cannot be a cashier and the manager at the same time.

\[\text{dom}(\text{balance}) \subseteq \text{customers}\]

Every person who has a bank balance recorded in the system must be a customer.
6.2 Mechanised analysis of confidentiality patterns

Data leakage is analysed with respect to the user roles held by a user

A user role is a named instance where a set of permissions are assigned to a group of users to perform a set of tasks. In the Circus notation, a user may perform tasks on a system through communications allowed on a system, that are made possible through channels. In this regard, a set of permissions are synonymous with a set of channels or a channelset in the Circus notation. Therefore, a channelset is used to model a user role in a system. A user may belong to one or more user roles. For example, in Figure 6.3 user x belongs to the user role A whereas user y belongs to both user roles A and B.

The observations that a user can make by executing a function depend on the set of authorized permissions for that user. One way BCF in Circus can be used to analyse a system for data leakage is by comparing the set of prohibited observations for a particular user $u$ against the set of observations the user $u$ can make by executing the system.

![Figure 6.3: User roles and user observations](image)

In the ongoing Bank information system example, the identity of the specific user who is currently logged into the system is represented by the variable `loginUser` and has the data type `PERSON`. The group of users that belong to the user role employees is represented by the variable `employees` and has the data type $\exists$ `PERSON`. 
6 Evaluation of mechanisation

6.2.3.e Formal specification of the Bank information system

The full formal specification of the Bank information system is presented in Figure 6.4.
6.2 Mechanised analysis of confidentiality patterns

(This page intentionally left blank)
[ PERSON ]

State

\[
\text{balance : PERSON } \rightarrow \text{N} \\
\text{reqCustomer, loginUser : PERSON} \\
\text{loggedIn, customers, cashiers, managers : } \mathbb{F} \text{ PERSON}
\]

\[
\text{dom (balance) } \subseteq \text{customers} \\
(\text{customers } \cap \text{ cashiers } \cap \text{ managers}) = \{\} \\
\text{loggedIn } \subseteq (\text{customers } \cup \text{ cashiers } \cup \text{ managers}) \\
\text{loginUser } \in \text{loggedIn}
\]

HideBalCustomer

\[\exists \text{ State }\]

\[
\exists \text{ State }_{9} \bullet \\
\text{loginUser } \neq \text{ reqCustomer } \land \\
\text{loginUser } \notin \text{ managers } \land \\
\text{loginUser } \notin \text{ cashiers } \land \\
\text{reqCustomer } \in \text{ dom (balance)} \\Rightarrow \\
(balance \text{ reqCustomer}) \neq (balance_{9} \text{ reqCustomer}_{9})
\]

channel \text{ newBalanceIn, mrbalOut, showBalanceOut, myBalanceOut : N} \\
channel \text{ withdrawAmountIn, depositAmountIn : N} \\
channel \text{ withdrawCustomerIn, depositCustomerIn, newCustomerIn : PERSON} \\
channel \text{ rubIn, customerIn : PERSON}

channelset BankingStaff == \{ \text{ customerIn, showBalanceOut } \} \\
channelset Cashier == \{ \text{ withdrawAmountIn, customerIn, showBalanceOut, depositAmountIn, depositCustomerIn, withdrawCustomerIn, newCustomerIn, newBalanceIn } \} \\
channelset Customer == \{ \text{ myBalanceOut } \}

process Bank\_information\_system \triangleq \text{ begin} \\

\text{ state State} \\

\text{ Init } \triangleq \text{ reqCustomer := loginUser} \\
\text{ NewAccount } \triangleq \text{ var newBalance : N}; \\
\quad \text{ newCustomer : PERSON } \bullet \\
\quad \text{ newBalanceIn?newBalance } \rightarrow \\
\quad \text{ newCustomerIn?newCustomer } \rightarrow \\
\quad \quad (\text{ (newCustomer } \notin \text{ dom (balance) } \land \text{ loginUser } \in \text{ cashiers) } \& \\
\quad \quad \quad \text{ balance := balance } \oplus \\
\quad \quad \quad \quad \left\{ (\text{newCustomer } \rightarrow \text{newBalance}) \right\})

\text{ DepositMoney } \triangleq \text{ var depositAmount : N}; \\
\quad \text{ depositCustomer : dom (balance) } \bullet \\
\quad \text{ depositAmountIn?depositAmount } \rightarrow \\
\quad \text{ depositCustomerIn?depositCustomer } \rightarrow \\
\quad \quad (\text{ (loginUser } \in \text{ cashiers) } \& \\
\quad \quad \quad \text{ balance := balance } \oplus \\
\quad \quad \quad \quad \left\{ (\text{depositCustomer } \rightarrow \text{balance depositCustomer}) \rightarrow \\
\quad \quad \quad \quad \quad (\text{balance depositCustomer } \rightarrow \text{balance depositCustomer } - \text{depositAmount}) ) \right\})

Figure 6.4: Specification of Bank information system - code block 1 of 2
WithdrawMoney ≜ \textbf{var} withdrawAmount : N;
withdrawCustomer : dom(balance) •
withdrawAmountIn?withdrawAmount →
withdrawCustomerIn?withdrawCustomer →
((loginUser ∈ cashiers) &
balance := balance \n{(withdrawCustomer? →
(balance withdrawCustomer? + withdrawAmount?))})

GetMyBalance ≜ reqCustomer := loginUser;
((loginUser ∈ customers &
loginUser ∈ dom(balance)) &
myBalanceOut!(balance loginUser) → Skip)

GetAnyCustBalance ≜ \textbf{var} customer : PERSON •
customerIn?customer →
reqCustomer := customer?;
((loginUser ∈ cashiers \ loginUser ∈ managers) &
showBalanceOut!(balance customer?) → Skip)

HideBalCustomer ≜ HideBalCustomer

UserOptions ≜ \left(\begin{array}{ll}
\text{NewAccount} & \text{DepositMoney} \\
\text{WithdrawMoney} & \text{GetAnyCustBalance} \\
\text{GetMyBalance} & \\
\end{array}\right); \text{HideBalCustomer}

• (Init); µY • \langle(UserOptions ; Y)\rangle

end
6 Evaluation of mechanisation

6.2.4 Formalising the confidentiality requirement

In this section, a possible formalisation of the confidentiality requirement CR18 is specified using the Circus notation. The general definition of CP1 states:

\[
\text{do not reveal the relation between } x \text{ and } y \text{ in } S
\]

The confidentiality requirement CR18 of the Bank information system reads:

\[
\text{CR18 : permission to view the balance of a customer should be given only to an authorised user.}
\]

The confidentiality requirement CR18 can be re-phrased as follows, to align with the definition of CP1.

\[
\begin{align*}
\text{if} & \quad \text{the user is not authorised} \\
\text{then} & \quad \text{do not reveal the association} \\
& \quad \text{between a customer and his account balance}
\end{align*}
\]

In this particular confidentiality requirement, the condition to hide the association between A and B is satisfied if the user is not authorised. [Tschantz and Wing (2008)] describes such a requirement as a conditional confidentiality requirement that contains a conditional information flow where information flow occurs only when some condition is met at runtime (Tschantz and Wing, 2008, p. 108).

**Identify the condition for confidentiality.** Papers published by [Parveen et al. (2015) p. 2] and [Lapadula et al. (2008) p. 713,714] that contain confidentiality requirements with CP1 does not mention which set of users are authorised to view the data. Therefore, in the case of an accounts system at a bank, it is assumed that the current user of the system loginUser is authorised to view the balance of the bank account B which belongs to the customer reqCustomer, if one of the following conditions are satisfied.
• loginUser is the manager

• loginUser is a cashier

• loginUser and reqCustomer are the same

If none of the above conditions are satisfied, then the balance of the bank account $B$ must never be revealed to the user loginUser. It must also be ensured that the account holder reqCustomer has a bank balance recorded in the bank balance register balance in the system. This combined condition can be written follows.

    loginUser is not the manager
    and loginUser is not a cashier
    and loginUser and reqCustomer are different
    and reqCustomer has an associated bank balance record in balance

The above condition may be formalised using the Circus notation as follows.

    loginUser \notin \text{managers}
    \land loginUser \notin \text{cashiers}
    \land loginUser \neq reqCustomer
    \land reqCustomer \in \text{dom balance}

Identify the confidential data. BCF states that whenever a user can observe the value of a variable $x$ in the normal system state space he/she can assume that the shadow system must also have the same value for its twin variable $\tilde{x}$.

In order to prevent the system from revealing the exact value of a particular variable, we must exclusively state a separation between the value of the variable in the original system and that in the shadow system. In this scenario we may write:

    x \neq \tilde{x}
6 Evaluation of mechanisation

In the case of the Bank information system, the value we must not reveal is:

\[ \text{balance reqCustomer} \]

where \( \text{balance} \) is a function between a customer identifier and the bank balance of that customer if any and \( \text{reqCustomer} \) is the identifier of the customer whose bank balance is being requested. To state a minimum separation between the value \( (\text{balance reqCustomer}) \) and its twin counterpart \( (\text{balance}_9 \text{ reqCustomer}_9) \) we may write:

\[ (\text{balance reqCustomer}) \neq (\text{balance}_9 \text{ reqCustomer}_9) \]

**Build the confidentiality predicate.** The condition for confidentiality and the separation defined for the confidential data are combined to come up with a formal definition for the confidentiality requirement **CR18**. The derived formal definition is shown below as a Z schema. As described in page 189 a subscript 9 is used to represent the state schema of the twin system as well as the variables from the twin system.

\[
\begin{align*}
\exists \text{State} & \ (
\exists \text{State}_9 \ ( \\
\text{loginUser} \neq \text{reqCustomer} \land \\
\text{loginUser} \in \text{customers} \land \\
\text{reqCustomer} \in \text{dom} (\text{balance}) \Rightarrow \\
(\text{balance reqCustomer}) \neq (\text{balance}_9 \text{ reqCustomer}_9)
\))
\end{align*}
\]

The schema \( \text{HideBalCustomer} \) includes a confidentiality constraint that must be enforced on the state space of a particular state or states of the Bank information system where \( \text{HideBalCustomer} \) is observed.

A pre-requisite for using BCF in Circus to analyse systems for confidentiality is to understand how a formalized confidentiality requirement can be integrated into a
6.2 Mechanised analysis of confidentiality patterns

6.2.5 Structure of the Circus specifications used in the mechanised analysis

A confidentiality integrated Circus specification is a Circus specification which includes one or more confidentiality annotations\(^2\). Figure 6.5 shows how a confidentiality integrated Circus specification has been structured for the purposes of the mechanised analysis carried out in this chapter. The structures \[\text{CA\_schema\_name}, \text{CA\_Action}\] and \[\text{CA\_predicate}\] are introduced into a Circus specification to integrate the confidentiality properties of a system within its specification. A detailed description of the Circus notation and how its structures can be used to model a system is given in Appendix A.1

Here, a summarised description of the main structures of a Circus specification are presented, which has been utilized in the case study specifications in this chapter. It must be noted that one can produce other possible specifications of the same system using the Circus notation.

The highlighted words **process**, **begin**, **state** and **end** in Figure 6.5 are Circus keywords. The description of the place-holders in Figure 6.5 are given below.

- **TypeDeclarations**
  The place-holder for the names of the Basic types defined in the specification.

- **StateVariableDeclarations**
  The place-holder for the declarations of state variables in the system.

- **StateInvariants**
  The place-holder for the state invariants that are required in the system state.

- **StateName**
  The place-holder for the name assigned to the schema that encodes the state of the system.

- **ChannelDefinitions**
  The place-holder where channels\(^3\) are defined in the specification.

---

\(^2\) A brief introduction to confidentiality annotations (Banks, 2012, p. 105) is discussed in Section 6.2.6.

\(^3\) The definition of channels and channelsets in the Circus notation are discussed in Freitas (2005, p. 13).
Evaluation of mechanisation

TypeDeclarations

\[ \text{StateName} \]

StateVariableDeclarations

StateInvariants

\[ \text{CA schema name} \]

\[ \Xi \text{StateName} \]

\[ \text{CA predicate} \]

ChannelDefinitions

ChannelsetDefinitions

\textbf{process} ProcessName \cong \textbf{begin}

\textbf{state} StateName

Action_1 \cong \ldots

\ldots

Action_n \cong \ldots

\text{CA Action} \cong \text{CA schema name}

\bullet \langle \text{MainAction} \rangle

\textbf{end}

Figure 6.5: Structure of the Circus specifications used in the mechanised analysis
6.2 Mechanised analysis of confidentiality patterns

cification.

ChannelsetDefinitions The place-holder where channelset are defined in the specification.

ProcessName The place-holder for the name assigned to the Circus process that is defined in the specification.

Action_1, ..., Action_n Place-holders for the names of the Circus actions that are defined in the system.

CA_schema_name The place-holder for the schema name that is defined to embed the confidentiality predicate.

CA_Action The place-holder for the name of the Circus action that is defined to use the CA_schema_name schema as a Circus action.

CA_predicate The place-holder for the formal predicate that represents a confidentiality property of the system being formalised.

MainAction The label MainAction represents the place-holder for the nameless main action that defines the process behaviour. For the purposes of the mechanised analysis in this chapter we define the main action as a composite action that uses Action_1 to Action_n and CA_Action.

6.2.6 Using the mechanised tool to analyse the system

The first step in using the mechanisation developed in this thesis, is to submit the BCF in Circus based formalization of the Bank information system (see Figure 6.10) to the CFAT tool in the CFAT format. Next, follow the steps in Figure 3.6 to analyse the
submitted specification using the mechanisation. The possible results from the analysis and how they can be interpreted are described in Section 3.2.6.

6.2.7 Strengthening a weak specification

Recall from Section 4.3 that a specification is referred as a ‘weak specification’ if the specification is seemingly incorrect but the mechanised back propagation of the specification using BCF in Circus results in a predicate that can be simplified to true. The Bank information system specification in Figure 6.4 has a fundamental weakness in its role specification, where the three-way intersection is not strong enough to rule out the same user playing two roles at once. This subsection explores the specific weakness and its solution.

Assume that a fictitious organisation has the user roles A, B and C. Further, assume that the organisation has a particular piece of confidential data CI.

Figure 6.6: Intersection of user roles A, B and C
6.2 Mechanised analysis of confidentiality patterns

The seemingly correct formalisation of organisational rules. Assume the following organisational rules.

- **Rule 1** - CI can be visible to users having the user role B and the user role C.
- **Rule 2** - CI must not be revealed to users having the user role A.
- **Rule 3** - Every user in the organisation must belong to either A or B or C.
- **Rule 4** - A user cannot occupy more than one user role in the organisation.

**Rule 1** was formalised as:

\[\text{user} \in (B \cup C) \Rightarrow \text{show CI} \quad (6.1)\]

and **Rule 2** was formalised as:

\[\text{user} \in A \Rightarrow \text{conceal CI} \quad (6.2)\]

and **Rule 3** was formalised as:

\[\text{user} \in (A \cup B \cup C) \quad (6.3)\]

and **Rule 4** was formalised as:

\[A \cap B \cap C = \{\} \quad (6.4)\]

where *user* is the user who was requesting access to the confidential information CI, ‘show CI’ is a specification that allows the value of CI to be revealed and ‘conceal CI’ is a specification that states that the value of the state variable CI must not be revealed in the current state. The consistency of the requirements in the specification, that included the formalisations Equation (6.1), Equation (6.2), Equation (6.3) and Equation (6.4), could not be verified using the proposed mechanisation.
**Strengthening the specification.** In order to prove the consistency of the requirements in the formalised specification of the system, the formalisation of Rule 2 had to be strengthened by replacing Equation (6.2) with Equation (6.5) as follows.

\[ \text{user} \notin (B \cup C) \Rightarrow \text{conceal CI} \quad (6.5) \]

Upon further investigation, it was found that the original formalisation of Rule 4 in Equation (6.4) only guaranteed that \( r \) in Figure 6.6 was empty. In this scenario, it may be possible that one or both of the subsets \( x \) and \( z \) might not be empty. In such a scenario, Equation (6.1) and Equation (6.2) will contradict each other. Strengthening the formalisation of Rule 2 worked because Equation (6.5) did not enforce Rule 2 on subsets \( x \) and \( z \). However, this means that there might be the possibility of data leakage should there be a user who belonged to \( x \) or \( z \) and who was requesting for CI based on Rule 1. The mechanisation had detected this data leakage.

**The solution.** The correct formalisation of Rule 4 must define the sets \( x, y, z \) and \( r \) as empty to ensure that a user cannot occupy more than one user role. For this, all possible pairwise intersections for the user roles in the organisation must be defined as empty as shown in Figure 6.7.

\[
\begin{align*}
A \cap B &= \{} \\
A \cap C &= \{} \\
B \cap C &= \{}
\end{align*}
\]

Figure 6.7: Pairwise disjoint statements for users roles \( A, B \) and \( C \)

If the pairwise disjoint statements in Figure 6.7 are used as invariants in the formal specification of the system, then the consistency of the requirements in the system against the confidentiality property in Equation (6.2) can be verified.
6.2.8 An example of strengthening a weak specification

Consider a fictitious hand-crafted bidding system where the value of the highest bid must never be revealed to a customer. Possible formal specifications of the system are presented in Figure 6.8 and Figure 6.9. Figure 6.8 presents a ‘weak specification’ of the system whereas the specification in Figure 6.9 presents a strengthened specification.

In both Figure 6.8 and Figure 6.9, the following representations are followed. The free type PERSON represents the identifiers of all the users who can use the system which includes alice, bob, carol, dave and eve. The loginUser is the identifier of the user who is currently using the system. The groups of users in the system include loggedIn which includes all the users currently using the system, customers that includes the identifiers of all the customers, cashiers that includes the identifiers of all the cashiers, managers that includes the identifiers of all the managers in the system. The system requires that the set of logged in users must always be from the combined set of customers, cashiers and managers. The system has two user roles where Customer includes all the users in customers and Staff includes all users in both cashiers and managers. The system has a function called RecordHighestBid which can be used to record the highest bid and ShowHighestBid which shows the highest bid recorded in the system.

Assume that the confidentiality requirement of the system states that the value of the highest bid highestBidvalue must never be revealed to the customers in the system. The confidentiality requirement may be formalised using the following schema.

\[
\text{HideHighestBidOriginal} \quad \Xi \text{State} \\
\exists \text{State}_9 \bullet \\
\quad \text{loginUser} \in \text{customers} \Rightarrow \\
\quad \text{highestBidvalue} \neq \text{highestBidvalue}_9
\]

The specification that includes HideHighestBidOriginal is shown in Figure 6.8. When simplifying the predicate generated from back propagating the specification Figure 6.8.
the Isabelle theorem prover highlights that there is contradiction in the predicate. This is because the triple intersection invariant
\[ \text{customers} \cap \text{cashiers} \cap \text{managers} = \{\} \]
in the following State schema of the specification in Figure 6.8 does not define the three sets as disjoint sets, as discussed on Page 214.

\[
\begin{align*}
\text{State} \\
\text{loginUser} &: \text{PERSON} \\
\text{loggedIn, customers, cashiers, managers} &: \notin \text{PERSON} \\
\text{highestBidvalue} &: \text{N} \\
\text{loginUser} \in \text{loggedIn} \\
\text{loggedIn} \subseteq (\text{customers} \cup \text{cashiers} \cup \text{managers}) \\
(\text{customers} \cap \text{cashiers} \cap \text{managers}) = \{\}
\end{align*}
\]

In this scenario, the \text{loginUser} who is currently logged into the system may belong to both \text{customers} and either of \text{cashiers} or \text{managers}. The conflicting requirements of the functions allowed for a user role assigned to a \text{manager} or a \text{cashier} and the confidentiality requirements enforced on a user role assigned to a \text{customer} is detected during the simplification of the predicate. This conflict is detected as a potential data leakage.

Therefore, to make the requirements in the specification consistent, an upper limit is defined on the users to whom the confidential data in the confidentiality requirement can be revealed. The change is included in the modified confidentiality requirement \text{HideHighestBidModified} as follows.

\[
\begin{align*}
\text{HideHighestBidModified} \\
\exists \text{State} \\
\exists \text{State} \quad \\
\text{loginUser} \notin (\text{cashiers} \cup \text{managers}) \Rightarrow \\
\text{highestBidvalue} \neq \text{highestBidvalue}_9
\end{align*}
\]
The confidentiality requirement in \textit{HideHighestBidModified} states that \textit{highestBidValue} must never be revealed to anyone except a user who belongs to the combined set of uses from \textit{customers} and \textit{managers}.

### 6.2.9 Strengthening the specification

In order to use \textit{HideHighestBidOriginal}, the original formalisation of the confidentiality requirement that aligns with the description of the confidentiality requirement, we must modify the system state so that the sets \textit{customers}, \textit{cashiers} and \textit{managers} are disjoint from each other. The system state is modified as follows to include pairwise intersections of the sets \textit{cashiers}, \textit{managers} and \textit{customers}.

\begin{verbatim}
State

loginUser : PERSON
loggedIn, customers, cashiers, managers : \# PERSON
highestBidvalue : \# IN

loginUser \in loggedIn
loggedIn \subseteq (customers \cup cashiers \cup managers)
(cashiers \cap managers) = \{}
(customers \cap managers) = \{}
(customers \cap cashiers) = \{}
\end{verbatim}

The modified specification of the system is included in Figure\textsuperscript{6.9}. The consistency of the requirements in this strengthened specification can now be verified.
State

loginUser : PERSON
loggedIn, customers, cashiers, managers : \( \mathbb{F} \) PERSON
highestBidvalue : \( \mathbb{N} \)

\( \text{loggedIn} \subseteq (\text{customers} \cup \text{cashiers} \cup \text{managers}) \)
\( (\text{customers} \cap \text{cashiers} \cap \text{managers}) = \{\} \)

HideHighestBid

\( \exists \text{State}_9 \cdot \)
\( \text{loggedIn} \notin (\text{cashiers} \cup \text{managers}) \Rightarrow \)
\( \text{highestBidvalue} \neq \text{highestBidvalue}_9 \)

channel \( \text{recordBidIn}, \text{showLastBidOut} : \mathbb{N} \)

channelset Customer \( == \{ | \text{showLastBidOut} | \} \)
channelset Staff \( == \{ | \text{recordBidIn}, \text{showLastBidOut} | \} \)

process SecretHighestBid \( \equiv \) begin

state State

RecordHighestBid \( \equiv \) var recordBid : \( \mathbb{N} \) •
recordBidIn?recordBid \( \rightarrow \)
\( ((\text{loggedIn} \in \text{cashiers} \lor \text{loggedIn} \in \text{managers}) \land \)
\( \text{highestBidvalue} := \text{recordBid}?) \)

ShowHighestBid \( \equiv \) \( ((\text{loggedIn} \in \text{cashiers} \lor \text{loggedIn} \in \text{managers}) \land \)
\( \text{showLastBidOut}! (\text{highestBidvalue} \rightarrow \text{Skip}) \)

HideHighestBid \( \equiv \) HideHighestBid

\( \bullet \mu X \bullet (\langle \text{RecordHighestBid} \rangle ; \langle \text{ShowHighestBid} \rangle ; \langle \text{HideHighestBid} \rangle ; X) \)

end

Figure 6.8: Specification of Secret Highest Bid - code block 1 of 1

( The weak specification )
PERSON ::= alice | bob | carol | dave | eve

\begin{align*}
\text{State} & \\
\text{loginUser : PERSON} \\
\text{loggedIn, customers, cashiers, managers : } & \text{\$PERSON} \\
\text{highestBidvalue : } & \text{N} \\
\text{loginUser } & \in \text{loggedIn} \\
\text{loggedIn } & \subseteq (\text{customers } \cup \text{ cashiers } \cup \text{ managers}) \\
\text{(cashiers } & \cap \text{ managers)} = \{\} \\
\text{(customers } & \cap \text{ managers)} = \{\} \\
\text{(customers } & \cap \text{ cashiers)} = \{\} \\
\end{align*}

\begin{align*}
\exists \text{State } 9 & \\
\text{loginUser} \in \text{customers } & \Rightarrow \\
\text{highestBidvalue } & \neq \text{highestBidvalue}_9 \\
\end{align*}

\begin{itemize}
\item channel recordBidIn, showLastBidOut : N
\end{itemize}

\begin{itemize}
\item channelset Customer == \{showLastBidOut\}
\item channelset Staff == \{recordBidIn, showLastBidOut\}
\end{itemize}

\begin{itemize}
\item process SecretHighestBid \equiv \text{begin} \\
\text{state State} \\
\text{RecordHighestBid} \equiv \text{var recordBid : N } \bullet \\
\text{recordBidIn?recordBid } \rightarrow \\
\text{((loginUser } \in \text{ cashiers } \lor \text{ loginUser } \in \text{ managers}) \& \\
\text{highestBidvalue } := \text{recordBid}?) \\
\text{ShowHighestBid} \equiv (\text{((loginUser } \in \text{ cashiers } \lor \text{ loginUser } \in \text{ managers}) \& \\
\text{showLastBidOut !}(\text{highestBidvalue} ) \\ \rightarrow \text{Skip}) \\
\text{HideHighestBid} \equiv \text{HideHighestBid} \\
\bullet \mu X \bullet \left( \square (\text{RecordHighestBid}) \right) ; (\text{HideHighestBid}) ; X \\
\end{itemize}

\text{end}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{image}
\caption{Specification of Secret Highest Bid - code block 1 of 1}
\end{figure}

( The strengthened specification )
6 Evaluation of mechanisation

6.2.9.a Strengthened formal specification of the Bank information system

The strengthened formal specification of the Bank information system is presented in Figure 6.10.
6.2 Mechanised analysis of confidentiality patterns
Evaluation of mechanisation

State

\[ \text{balance} : \text{PERSON} \mapsto \mathbb{N} \]
\[ \text{reqCustomer, loginUser} : \text{PERSON} \]
\[ \text{loggedIn, customers, cashiers, managers} : \mathbb{F} \text{PERSON} \]

\[ \text{dom (balance)} \subseteq \text{customers} \]
\[ (\text{cashiers} \cap \text{managers}) = \{\} \]
\[ (\text{customers} \cap \text{managers}) = \{\} \]
\[ (\text{customers} \cap \text{cashiers}) = \{\} \]
\[ \text{loggedIn} \subseteq (\text{customers} \cup \text{cashiers} \cup \text{managers}) \]
\[ \text{loginUser} \in \text{loggedIn} \]

HideBalCustomer

\[ \exists \text{State}_9 \quad \]
\[ \text{loginUser} \neq \text{reqCustomer} \land \]
\[ \text{loginUser} \in \text{customers} \land \]
\[ \text{reqCustomer} \in \text{dom (balance)} \Rightarrow \]
\[ (\text{balance reqCustomer}) \neq (\text{balance}_9 \text{reqCustomer}_9) \]

channel \(\text{newBalanceIn}, \text{mrbalOut}, \text{showBalanceOut}, \text{myBalanceOut} : \mathbb{N}\)
channel \(\text{withdrawAmountIn}, \text{depositAmountIn} : \mathbb{N}\)
channel \(\text{withdrawCustomerIn}, \text{depositCustomerIn}, \text{newCustomerIn} : \text{PERSON}\)
channel \(\text{rubIn, customerIn} : \text{PERSON}\)

channelset BankingStaff \(== \{\text{customerIn, showBalanceOut} \}\)
channelset Cashier \(== \{\text{withdrawAmountIn, customerIn, showBalanceOut, depositAmountIn, depositCustomerIn, withdrawCustomerIn, newCustomerIn, newBalanceIn} \}\)
channelset Customer \(== \{\text{myBalanceOut} \}\)

process Bank_information_system \(\triangleq \text{begin}\)

state State

\[ \text{Init} \quad \triangleq \ \text{reqCustomer} := \text{loginUser} \]
\[ \text{NewAccount} \quad \triangleq \ \text{var} \ \text{newBalance} : \mathbb{N}; \]
\[ \text{newCustomer} : \text{PERSON} \bullet \]
\[ \text{newBalanceIn?newBalance} \rightarrow \]
\[ \text{newCustomerIn?newCustomer} \rightarrow \]
\[ ((\text{newCustomer} \notin \text{dom (balance)} \land \text{loginUser} \in \text{cashiers}) \land \]
\[ \text{balance} := \text{balance} \oplus \]
\[ \{(\text{newCustomer} \rightarrow \text{newBalance})\}) \]
\[ \text{DepositMoney} \quad \triangleq \ \text{var} \ \text{depositAmount} : \mathbb{N}; \]
\[ \text{depositCustomer} : \text{dom (balance)} \bullet \]
\[ \text{depositAmountIn?depositAmount} \rightarrow \]
\[ \text{depositCustomerIn?depositCustomer} \rightarrow \]
\[ ((\text{loginUser} \in \text{cashiers}) \land \]
\[ \text{balance} := \text{balance} \oplus \]
\[ \{(\text{depositCustomer} \rightarrow \text{balance depositCustomer}) \rightarrow \]
\[ (\text{balance depositCustomer} \rightarrow \text{depositAmount}))\}) \]

Figure 6.10: Specification of Bank information system - code block 1 of 2
WithdrawMoney ≜ var withdrawAmount : N;
withdrawCustomer : dom(balance) •
withdrawAmountIn? withdrawAmount →
withdrawCustomerIn? withdrawCustomer →
((loginUser ∈ cashiers) &
  balance := balance \ {(withdrawCustomer? → (balance withdrawCustomer? + withdrawAmount?))})

GetMyBalance ≜ reqCustomer := loginUser;
  ((loginUser ∈ customers &
    loginUser ∈ dom(balance)) &
    myBalanceOut !(balance loginUser) → Skip)

GetAnyCustBalance ≜ var customer : PERSON •
customerIn? customer →
reqCustomer := customer?;
  ((loginUser ∈ cashiers ∨ loginUser ∈ managers) &
    showBalanceOut !(balance customer?) → Skip)

HideBalCustomer ≜ HideBalCustomer • ⟨Init⟩; µY • ⟨(UserOptions ; Y)⟩

UserOptions ≜

□ NewAccount
□ DepositMoney
□ WithdrawMoney
□ GetAnyCustBalance
□ GetMyBalance

• (Init); µY • ⟨(UserOptions ; Y)⟩

end
6 Evaluation of mechanisation

6.2.10 Results of the analysis

Table 6.8 presents the results of the mechanised analysis of the Bank information system. The total time taken for the mechanised analysis is the combined total of the time it takes to back propagate the specification using the CFAT tool and the time it takes to simplify the generated predicate using the Isabelle theorem prover.

The process of back propagation calculates the user’s inference about the process state at each step of the process execution (Banks 2012, p. 186). The results of the back propagation are dependent on the set of channels that is accessible to a user (see page 201). Since each user role in this system has access to a different set of channels, the results of the mechanised analysis of the system is presented in relation to users in each user role.

<table>
<thead>
<tr>
<th>User role against which the system has being analysed</th>
<th>Cashier</th>
<th>Customer</th>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time taken for back propagation</td>
<td>474 ms</td>
<td>1055 ms</td>
<td>641 ms</td>
</tr>
<tr>
<td>Time taken for predicate simplification</td>
<td>2043 ms</td>
<td>2194 ms</td>
<td>2120 ms</td>
</tr>
<tr>
<td>Total time taken for evaluation</td>
<td>2517 ms</td>
<td>3249 ms</td>
<td>2761 ms</td>
</tr>
<tr>
<td>Result of the mechanised evaluation</td>
<td>Simplified</td>
<td>Simplified</td>
<td>Simplified</td>
</tr>
</tbody>
</table>

Table 6.8: Results of the mechanised analysis of the Bank information system

The results show that the tool saves time by executing the back propagation in a matter of milliseconds. However, as demonstrated with a smaller specification in Section 4.2, a manual application of the back propagation process will be very much slower while also being error prone.

Table 6.8 shows that the analysis of the system with respect to the three user roles in the Bank information system results in the outcome “Simplified”. Recall from Section 3.2.6 that if the predicate generated from back propagating a system specification can
be simplified, then according to BCF in *Circus*, there are no contradictions in the specification of the system being analysed.

### 6.2.11 Negative testing

Positive testing is used to verify the functionality of a product whereas negative testing is used to verify that a product does not do something (Oehlert, 2005, p. 58). In the context of this thesis, positive testing is used to verify that there are no contradictions between the functionality and confidentiality requirements in a system. Negative testing uses cases that are expected to fail (Olan, 2003, p. 320).

"Negative testing is performed to ensure that the system is able to handle inconsistent information. Negative acceptance tests (often expressed in the form of negative scenarios) are increasingly recognized as a powerful way of thinking about requirements, possible conflicts, and identifying threats." (Melnik et al., 2006, p. 41)

In the context of testing systems for data leakage related confidentiality requirements, using BCF in *Circus*, a potential negative test will be to evaluate what will happen in a scenario where one accidentally introduces a side channel that violates access control requirements in a system. To simulate this scenario, an explicit side channel has been introduced to the hand-crafted system specification of the Bank information system. The basic approach is to:

<table>
<thead>
<tr>
<th><strong>introduce</strong></th>
<th>a channel based access to a piece of data $d$ for a user role $ur$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>while</strong></td>
<td>there is already an existing confidentiality requirement that exclusively restricts the user role $ur$ from having access to $d$.</td>
</tr>
</tbody>
</table>

Such a test is expected to fail since there is a contradiction in the defined access parameters.
6 Evaluation of mechanisation

To introduce a side channel, a function with unconstrained access is introduced to the system specification of the Bank information system. This new Circus action, called GetBalance, allows any user using the system to view the account balance of any customer. The formal specification of the action GetBalance is as follows.

\[
\text{GetBalance} \quad \equiv \quad \text{var rub : dom (balance)} \bullet \\
\quad \text{rubin?rub} \rightarrow \\
\quad \text{reqCustomer := rub?;}
\quad \text{mrbalOut !(balance rub?)} \rightarrow \text{Skip}
\]

The confidentiality requirement HideBalCustomer restricts certain users from knowing the balance of any customer while the Circus action GetBalance allows any user to know the balance of any customer. Table 6.9 illustrates the results of a mechanised analysis of a formal specification that includes GetBalance.

Table 6.9 shows that the analysis of the Bank information system specification (that includes the side channel), with respect to the user role “Customer”, results in the outcome “Time-out”. Recall from Section 3.2.6 that if the predicate generated from back propagating a system specification does not reach a conclusion during simplification but rather times-out, then nothing can be concluded about the presence or absence of contradictions in the specification. As stated earlier in Section 3.2.6, the result can be interpreted as provably true or provably false.
6.2 Mechanised analysis of confidentiality patterns

<table>
<thead>
<tr>
<th>User role against which the system has being evaluated</th>
<th>Cashier</th>
<th>Customer</th>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time taken for back propagation</td>
<td>374 ms</td>
<td>344 ms</td>
<td>350 ms</td>
</tr>
<tr>
<td>Time taken for predicate simplification</td>
<td>771 ms</td>
<td>-</td>
<td>930 ms</td>
</tr>
<tr>
<td>Total time taken for evaluation</td>
<td>1145 ms</td>
<td>344 ms</td>
<td>1280 ms</td>
</tr>
<tr>
<td>Result of the simplification by the theorem prover</td>
<td>Simplified</td>
<td>Time-out</td>
<td>Simplified</td>
</tr>
</tbody>
</table>

Table 6.9: Result of the mechanised evaluation of the Bank information system with a side channel

6.2.12 Analysing other confidentiality patterns

Next, the confidentiality patterns CP2, CP3, CP4 and CP5 will be analysed in order.

6.2.12.a Analysing the confidentiality pattern CP2

The following is an analysis of a system with a confidentiality property that reflects the confidentiality pattern CP2. The confidentiality requirement CR14 has been chosen as an adhoc choice for this analysis.

The mechanised analysis of CR14 will be discussed using a hand-crafted system requirement specification of a fictitious Phone book system of a Secret government agency (Cerny and Alur, 2009b, p. 175). The fictitious Phone book system of a Secret government agency has been used as a case study for confidentiality analysis by others such as Lunt (1989) and Jajodia and Meadows (1995).

The full formal specification for the Phone book system is included in Appendix A.5.1. Here, we present the requirement specification of the system and the results of analysing
the presented specification of the system. It must be noted that the formal specification included in Appendix A.5.1 is just one possible implementation of the system using the Circus notation.

6.2.12.a.1 Requirement specification of a system having CP2

The following requirement specification of the Phone book system has been divided into the organisational structure, the organisational rules, the user roles in the system, the operations and the user roles and permissions matrix of the system. Figure 6.11 presents a summarised use case diagram of the Phone book system.

Organisational structure

- Every engineer, secretary, official and the manager is an employee of the agency.
- The agency uniquely identifies the employee who is the manager of the agency.
- The agency maintains a list that contains every engineer of the agency.
- The agency maintains a list that contains every secretary of the agency.
- The agency maintains a list that contains every official of the agency.
- The agency maintains a list that contains every official with a confidential phone number in the agency.
- The agency maintains a phone number for a subset of existing officials in the agency.

Organisational rules

- The same employee cannot be a secretary and an engineer.
- The same employee cannot be a secretary and the manager.
- The same employee cannot be an engineer and the manager.
6.2 Mechanised analysis of confidentiality patterns

- The agency phone book can only be used by either a secretary or an engineer or the manager.

- The set of officials with a confidential phone number must be from the set of officials whose phone numbers have been recorded in the phone book.

User roles in the system

The following are user roles of the system. These user roles reflect the actors that are included in the use case diagram in Figure 6.11. The tasks that can be performed by a user belonging to each user role is described in Table 6.4.

- The Manager user role includes the manager of the agency.
- The Secretary user role includes all users who are secretaries in the agency.
- The Engineer user role contains all users who are engineers in the agency.

Operations, user roles and permissions

Table 6.10 lists all the system operations, and the specific permissions on those operations by user role.
6 Evaluation of mechanisation

<table>
<thead>
<tr>
<th>Functions that can be performed in the Phone book system</th>
<th>User roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record the phone_number of a particular official in the phone book.</td>
<td>✓</td>
</tr>
<tr>
<td>Find the phone number of an official who is not in the list of officials with a confidential phone number.</td>
<td>✓ ✓</td>
</tr>
<tr>
<td>Find the phone number of any official, recorded in the phone book.</td>
<td>✓</td>
</tr>
<tr>
<td>Record the name of an official in the list of officials whose phone numbers are to be kept confidential.</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 6.10: Roles and Permissions Matrix of the Phone book system

6.2.12.a.2 Formalising the confidentiality requirement CR14

In this section, a possible formalisation of the confidentiality requirement CR14 is presented using the Circus notation. The general definition of CP2 states:

\[
\text{do not reveal whether } x \text{ is a member of } S
\]

The confidentiality requirement CR14 of the Phone book system discussed in [Cerny and Alur, 2009a, p. 175] reads:

**CR14**: The property to be kept secret for the example is whether a particular string, say ‘555-55’ is in the phone book.

Recall that the confidentiality requirement CR14 has been rephrased as follows, to align with the definition of CP2.
This is another confidentiality requirement with a conditional information flow (Tschantz and Wing, 2008, p. 108) where the condition to hide the information about the membership of $x$ in $S$ is satisfied if the user is not authorised.

**Identify the condition for confidentiality.** Cerny and Alur (2009b) do not detail the specific users from whom the information is required to be hidden. Therefore, in the case of the Phone book system, it is assumed that the current user of the system $loginUser$ is not authorised to view the phone number of a secret official if the user is not the manager and the name of the requested official $reqOfficial$ is in the list of secret officials. These conditions can be combined and written as follows.

\[
\text{if the user is not authorized then for every given } x, \\
\text{ do not reveal whether } x \text{ is a member of the set } S \\
\text{ (where } S \text{ contains the phone numbers in the phone book)}
\]
6 Evaluation of mechanisation

`loginUser` is not the manager

and `reqOfficial` is in the list of secret officials

The above condition may be formalised using the Circus notation as follows.

\[
loginUser \notin \text{managers} \\
\land \ reqOfficial \in \text{secretList}
\]

Identify the confidential data. In order to hide the membership of a particular variable \(x\) in the set \(S\) in the normal state space, it is required to have the cover story that the twin variable \(x_9\) is not a member of the twin set \(S\) in the shadow state space. BCF states that whenever a user can observe the value of a variable \(x\) in the normal system state space he/she can assume that the shadow system must also have the same value for its twin variable \(x_9\).

In order to prevent the system from revealing the exact value of a particular variable, we must exclusively state a separation between the value of the variable in the original system and that in the shadow system. In this scenario we may write:

\[
x \in S \Rightarrow x_9 \notin S_9
\]

In the case of the Phone book system, where phone number of a secret official `reqOfficial` is requested, information about the membership of the official in the phone book register `PhoneNumbers` must not revealed. This may be formalised as follows:

\[
reqOfficial \in \text{dom } phoneNumbers \\
\Rightarrow reqOfficial_9 \notin \text{dom } phoneNumbers_9
\]

where `PhoneNumbers` is a function between an identifier for an official and a phone number, if any.
6.2 Mechanised analysis of confidentiality patterns

**Build the confidentiality predicate.** Now, we combine the condition for confidentiality and the separation defined for the confidential data to come up with a formal definition for the confidentiality requirement CR14. The derived formal definition is shown below as a Z schema.

\[
\exists \text{State} \quad \exists \text{State 9} \quad \begin{align*}
\text{reqOfficial} & \in \text{secretList} \land \\
\text{reqOfficial} & \neq \text{loginUser} \land \\
\text{loginUser} & \neq \text{manager} \Rightarrow \\
\text{reqOfficial} & \in \text{dom (phoneNumbers)} \Rightarrow \\
\text{reqOfficial}_9 & \notin \text{dom (phoneNumbers)}
\end{align*}
\]

The schema HideSecretNumber includes a confidentiality constraint that must be enforced on the state space of a particular state or states of the Phone book system where HideSecretNumber is observed.

**6.2.12.a.3 Results of analysing the Phone book system**

Table 6.11 shows that the analysis of the system with respect to the three user roles in the Phone book system results in the outcome “Simplified”. Recall from Section 3.2.6 that if the predicate generated from back propagating a system specification can be simplified, then according to BCF in Circus, there are no contradictions in the specification of the system being analysed.
6 Evaluation of mechanisation

<table>
<thead>
<tr>
<th>User role against which the system has being analysed</th>
<th>Secretary</th>
<th>Engineer</th>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time taken for back propagation</td>
<td>338 ms</td>
<td>223 ms</td>
<td>445 ms</td>
</tr>
<tr>
<td>Time taken for predicate simplification</td>
<td>1904 ms</td>
<td>2919 ms</td>
<td>1802 ms</td>
</tr>
<tr>
<td>Total time taken for evaluation</td>
<td>2242 ms</td>
<td>3142 ms</td>
<td>2247 ms</td>
</tr>
<tr>
<td>Result of the simplification by the theorem prover</td>
<td>Simplified</td>
<td>Simplified</td>
<td>Simplified</td>
</tr>
</tbody>
</table>

Table 6.11: Results of the mechanised analysis of the Phone book system

6.2.12.b Analysing the confidentiality pattern CP3

The following is an analysis of a system with a confidentiality property that reflects the confidentiality pattern CP3. The confidentiality requirement CR21 has been chosen as an adhoc choice for this analysis.

The mechanised analysis of CR21 will be discussed using the Secure electronic examination system by Foley and Jacob (1995). For this, a hand-crafted system requirement specification of a fictitious Secure electronic examination system has been developed. Some functions listed in this specification have been borrowed from the description of the Secure electronic examination system by Foley and Jacob (1995).

The full formal specification for the Secure electronic examination system is included in Appendix A.5.2. Here, we present the requirement specification of the system and the results of analysing the presented specification of the system. It must be noted that the formal specification included in Appendix A.5.2 is just one possible implementation of the system using the Circus notation.
6.2 Mechanised analysis of confidentiality patterns

6.2.12.b.1 Requirement specification of a system having CP3

The following requirement specification of the Secure electronic examination system has been divided into the organisational structure, the organisational rules, the user roles in the system, the operations and the user roles and permissions matrix of the system. Figure 6.12 presents a summarised use case diagram of the Secure electronic examination system.

Organisational structure

- A chair is a user.
- A setter is a user.
- A checker is a user.
- A grader is a user.
- The company maintains a chair for each existing subject.
- The company maintains a setter for each existing paper.
- The company maintains a checker for each existing paper.
- The company maintains a grader for each existing paper.
- The company maintains a result for each existing candidate.
- The company maintains a paper status for each existing paper.
- The company maintains a subject of each existing paper.
- The company maintains a list that contains every student of the company.
- The company maintains a list that contains every lecturer of the company.
- The company maintains a list that contains every loggedInUser of the company.
6 Evaluation of mechanisation

- The company uniquely identifies the user who is the current_user of the company.

- The company uniquely identifies the can who is the current_user of the company.

- The company records the relationship between a candidate and a paper.

- The company records the relationship between a candidate and a answer.

- The company records the relationship between a paper and a question.

- The company records the relationship between a paper and a answer.

Organisational rules

- All subject chairs, setters, checkers and graders must be lecturers.

- A lecturer cannot have more than one role. Hence, the lecturer can either be a subject chair, setter, checker or grader.

- The set of users allowed access to the system is a subset of the set of lecturers and students.

- A person cannot be both a lecturer and a student in the system.

- Every paper with a paper status must belong to an announced examination.

- Every paper for which a setter has been assigned must have a paper status.

- Every paper for which a checker has been assigned must have a paper status.

- Every paper for which a grader has been assigned must have a paper status.

- Every paper on which a question is recorded must have a paper status.

- Every paper on which an answer is recorded must have a paper status.

- Every paper registered for, must have a paper status.
6.2 Mechanised analysis of confidentiality patterns

- Every student registered for a paper must be a registered candidate.
- Every student registered as a candidate must be a student recorded in the system.
- Every result must belong to a registered paper.
- Every answer recorded must be for a registered paper.
- Lecturers who record questions in the system must be setters.
- Every question recorded in the system must be for a registered paper.
- Every answer recorded in the system must be for a registered paper.

User roles in the system

The following are user roles of the system. These user roles reflect the actors that are included in the use case diagram in Figure 6.12. The tasks that can be performed by a user belonging to each user role is described in Table 6.12.

- The Chair user role contains all users who are subject chairs in the institution.
- The Setter user role includes all users who are setters of examination papers in the institution.
- The Checker user role includes all users who check examination papers in the institution.
- The Marker user role includes all users who grade the examination papers in the institution.
- The Student user role contains all users who are studying in the institution.

Operations, user roles and permissions

Table 6.12 lists all the system operations, and the specific permissions on those operations by user role.
## Evaluation of mechanisation

### User roles

<table>
<thead>
<tr>
<th>Functions that can be performed on the Phone book system</th>
<th>Chair</th>
<th>Setter</th>
<th>Checker</th>
<th>Marker</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Announce an examination for a particular subject.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appoint a lecturer for the position of a <em>setter</em> from the relevant subject group to set the paper.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appoint a lecturer for the position of a <em>checker</em> from the relevant subject group to check the paper.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appoints a lecturer called a <em>grader</em> from the relevant subject group to grade the candidates who grade the paper.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set a paper if the user is authorised to do so.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release a paper if the user is authorised to do so.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close an examination if the user is authorised to do so.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check a paper if the user is authorised to do so.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grade a paper if the user is authorised to do so.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Publish grades of candidates who set for a particular paper if the user is authorised to do so.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register oneself as a candidate to sit for a particular paper.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record an answer for particular paper if the user is registered as a candidate for that particular paper.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find results of oneself for a particular paper.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cancel oneself registration for a particular paper.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.12: Roles and Permissions Matrix of the Secure electronic examination system

In the paper Secure Electronic Examinations by [Foley and Jacob] (1995) many setters and checkers could be appointed. However, to keep our model simple, we consider a single setter and a single checker for a paper.
6.2 Mechanised analysis of confidentiality patterns

6.2.12.b.2 Formalising the confidentiality requirement CR21

The general definition of confidentiality pattern CP3 reads:

\[
\text{do not reveal the set } S
\]

The confidentiality requirement CR21 identified from the Secure electronic examination system states:

**CR21** : No examinee should learn any details of the contents of any other examinees answer paper between the start of the examination and the end of the examination.
Recall from page 162 that the confidentiality requirement CR21 has been rephrased as follows, to align with the definition of CP3.

\[
\text{if the user } u \text{ currently using the system is not requesting information that belongs to him/her and exam } e \text{ has started but not ended}
\]
\[
\text{then do not reveal the set } S
\]
\[
(\text{where } S \text{ represents the answers recorded by a user other than } u \text{ in the exam } e)
\]

Similar to the earlier discussion in Section 6.2.4, CR21 is also a conditional confidentiality requirement [Tschantz and Wing 2008]. The formalisation of the condition for confidentiality and the confidential data are discussed separately.

**Identify the condition for confidentiality.** In this particular confidentiality requirement, the condition to hide the set containing the answers recorded by other examinees from the current user of the systems is satisfied if the examination for the paper of the requested user has started and has but not ended yet.

In our hand-crafted formal specification for the Secure electronic examination system (see Appendix A.5.2), the possible states of a paper is represented by the free type \textit{PAPERSTATUS}. The description of all these states can be found in Appendix A.5.2. The status of the paper between the start of an examination for that paper and its end is represented by the value \textit{released}.

We may write the conditions required to satisfy the confidentiality requirement using the Circus notation as follows:
6.2 Mechanised analysis of confidentiality patterns

<table>
<thead>
<tr>
<th>condition stated in the confidentiality requirement</th>
<th>one possible specification in Circus</th>
</tr>
</thead>
<tbody>
<tr>
<td>examination of the paper for which the current user is a candidate has started and not ended</td>
<td>$\text{pStatus}(\text{regPaper theCandidate}) = \text{released}$</td>
</tr>
<tr>
<td>the user currently using the system is not requesting information that belongs to him/her</td>
<td>$((\text{regStudent} \sim) \text{loginUser} \neq \text{theCandidate})$</td>
</tr>
</tbody>
</table>

Table 6.13: Formal specification of conditions required to satisfy the confidentiality requirement CR21

The confidentiality of the data must be enforced if all the conditions in Table 6.13 are satisfied. We may write this combined condition as follows.

$$((\text{regStudent} \sim) \text{loginUser} \neq \text{theCandidate}) \land \text{pStatus}(\text{regPaper theCandidate}) = \text{released}$$

**Identify the confidential data.** Here, we are to hide every single answer recorded by the examinee theCandidate from the current user. The set of answers recorded by the examinee theCandidate is represented by the set answersB:

$$\text{answersB} \equiv \{\text{theCandidate}\} \leftarrow \text{ansStudent}$$

where ansStudent is a relation between candidate identifiers and answers they have recorded. In the shadow system, the set of answers recorded by the examinee theCandidate is represented by the set answersB9:

$$\text{answersB}_9 \equiv \{\text{theCandidate}_9\} \leftarrow \text{ansStudent}_9$$
where \( ans_{Student} \) is the twin counterpart of \( ans_{Student} \). To prevent the system from revealing any value in the set \( answers_B \) we state that the sets \( answers_B \) and \( answers_{B9} \) are disjoint.

\[
\left( \{b\} \leq ans_{Student} \right) \cap \left( \{b\} \leq ans_{Student_{9}} \right) = \{\}
\]

**Build the confidentiality predicate.** We now combine the condition and the action to come up with a formal definition for the confidentiality requirement \( CR21 \). The derived formal definition is shown below.

\[
\begin{align*}
\text{HideOthersAnswers} & \leq \text{State} \\
\exists \text{State}_{9} \bullet & \\
& \left( (\text{regStudent} \sim) \ loginUser \neq \text{theCandidate} \land \right. \\
& \text{pStatus} (\text{regPaper theCandidate}) = \text{released} \Rightarrow \\
& \left. (\text{ran}(\{\text{theCandidate}\} \leq ans_{Student}) \cap \right. \\
& \left. \text{ran}(\{\text{theCandidate}_{9}\} \leq ans_{Student_{9}})) = \{\} \right.
\end{align*}
\]

The schema \( HideOthersAnswers \) includes a confidentiality constraint that must be enforced on the state space of a particular state or states of the Secure electronic examination system where \( HideOthersAnswers \) is observed.

**6.2.12.b.3 Results of analysing the Secure electronic examination system**

Table 6.14 shows that the analysis of the system with respect to the user roles in the Secure electronic examination system results in the outcome “Simplified”. Recall from Section 3.2.6 that if the predicate generated from back propagating a system specification can be simplified, then according to BCF in Circus, there are no contradictions in the specification of the system being analysed.
6.2 Mechanised analysis of confidentiality patterns

<table>
<thead>
<tr>
<th>User role against which the system has been evaluated</th>
<th>Setter</th>
<th>Marker</th>
<th>Chair</th>
<th>Checker</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time taken for back propagation</td>
<td>12854 ms</td>
<td>15282 ms</td>
<td>4860 ms</td>
<td>7378 ms</td>
<td>9887 ms</td>
</tr>
<tr>
<td>Time taken for predicate simplification</td>
<td>7772 ms</td>
<td>8808 ms</td>
<td>7371 ms</td>
<td>6201 ms</td>
<td>7558 ms</td>
</tr>
<tr>
<td>Total time taken for evaluation</td>
<td>20626 ms</td>
<td>24090 ms</td>
<td>12231 ms</td>
<td>13579 ms</td>
<td>17445 ms</td>
</tr>
<tr>
<td>Result of the simplification by the theorem prover</td>
<td>Simplified</td>
<td>Simplified</td>
<td>Simplified</td>
<td>Simplified</td>
<td>Simplified</td>
</tr>
</tbody>
</table>

Table 6.14: Results of the mechanised evaluation of the Secure electronic examination system

6.2.12.c Analysing the confidentiality pattern CP4

The following is an analysis of a system with a confidentiality property that reflects the confidentiality requirement pattern CP4. The confidentiality requirement CR16 has been chosen as an adhoc choice for this analysis.

The mechanised analysis of CR16 will be discussed using a hand-crafted system requirement specification of a fictitious CR16. The fictitious ePurse system has been used as a case study for confidentiality analysis by De Landtsheer and Van Lamsweerde (2005, p. 44).

The full formal specification for the ePurse system is included in Appendix A.5.1. Here, we present the requirement specification of the system and the results of analysing the presented specification of the system. It must be noted that the formal specification
Evaluation of mechanisation

included in Appendix A.5.3 is just one possible implementation of the system using the Circus notation.

6.2.12.c.1 Requirement specification of a system having CP4

The following requirement specification of the ePurse system has been divided into the organisational structure, the organisational rules, the user roles in the system, the operations and the user roles and permissions matrix of the system. Figure 6.13 presents a summarised use case diagram of the ePurse system.

Organisational structure

- The ePurse system maintains an ePurse for every buyer identifier from a subset of buyers in the system.
- The ePurse system maintains a balance identifier for every ePurse from a subset of ePurses in the system.
- The ePurse system maintains an ePurse identifier for every transaction from a subset of transactions in the system.
- The ePurse system maintains a terminal identifier for every transaction from a subset of transactions in the system.
- The ePurse system maintains a balance identifier for every transaction from a subset of transactions in the system.
- The ePurse system maintains an ePurse for every buyer from a subset of buyers in the system.
- The ePurse system maintains a validation status for every transaction from a subset of transactions in the system.
- The ePurse system uniquely identifies the agent that is currently requesting to execute a function in the system.
6.2 Mechanised analysis of confidentiality patterns

- The ePurse system uniquely identifies the ePurse, the balance of which is requested from the system.

Organisational rules

- The ePurse, of which a balance is requested from the system, must have a balance recorded in the system.

- The ePurse, of which a balance is requested from the system, must have an associated owner recorded in the system.

- Every transaction that has an associated agent must also have a transaction amount recorded in the system.

- Every transaction that has an associated ePurse must also have a transaction amount recorded in the system.

- Every transaction that has an associated validation status must also have a transaction amount recorded in the system.

- Every ePurse that has a transaction associated with it must also have a balance recorded in the system.

- Every ePurse that has a balance associated with it must belong to an agent in the system.

User roles in the system

The following are user roles of the system. These user roles reflect the actors that are included in the use case diagram in Figure 6.13. The tasks that can be performed by a user belonging to each user role is described in Table 6.15.

- The Buyer user role includes the buyers in the ePurse system.

- The Seller user role includes the sellers in the ePurse system.

- The Terminal user role includes the terminals in the ePurse system.
6 Evaluation of mechanisation

Operations, user roles and permissions

Table 6.15 lists all the system operations, and the specific permissions on those operations by user role.

<table>
<thead>
<tr>
<th>Operations that can be performed on the ePurse system</th>
<th>User roles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seller</td>
</tr>
<tr>
<td>Record a transaction in the system.</td>
<td>✓</td>
</tr>
<tr>
<td>Approve a transaction on the ePurse of a buyer.</td>
<td></td>
</tr>
<tr>
<td>Process the payment for a translation.</td>
<td></td>
</tr>
<tr>
<td>Get the balance in the ePurse of a given buyer.</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.15: Roles and Permissions Matrix of the ePurse system

Figure 6.13: Use case diagram for the e-Purse system
6.2.12.c.2 Formalising the confidentiality requirement CR16

In this section, a possible formalisation of the confidentiality requirement CR16 is specified using the Circus notation. The general definition of CP4 states:

\[
\text{do not reveal the exact value of } x
\]

The confidentiality requirement CR16 of the ePurse system by De Landtsheer and Van Lamsweerde (2005, p. 44) reads:

\[
\text{CR16 : agents who are not the card holder should not know the exact value of some state variable.}
\]

Recall that the confidentiality requirement CR16 has been rephrased as follows, to align with the definition of CP4.

\[
\text{if the agent is not the card holder then do not reveal the value of } x
\]

Once again, like the previous confidentiality requirements, this confidentiality requirement requires confidentiality of information under certain conditions.

**Identify the condition for confidentiality.** De Landtsheer and Van Lamsweerde (2005, p. 44) state that the current agent currAgent using the system is not authorised to view the exact value of a state variable if does not. If that state variable is the balance of a particular ePurse reqPurse, then the requirement states that currAgent must not know the exact balance of reqPurse if currAgent does not own reqPurse. This confidentiality condition can be written as follows.

\[
\text{currAgent is not the owner of the ePurse}
\]

The above condition may be formalised using the Circus notation as follows.
6 Evaluation of mechanisation

\[(\text{currAgent} \mapsto \text{reqPurse}) \notin \text{owns}\]

where \text{own} is a function that identifies the ePurse of a given agent, if any.

**Identify the confidential data.** In order to prevent the system from revealing the exact value of a particular variable, we must exclusively state a separation between the value of the variable in the original system and that in the shadow system. In the case of the ePurse system, such a separation is formalised as follow.

\[(\text{balance reqPurse}) \neq (\text{balance}_{9} \text{reqPurse}_{9})\]

**Build the confidentiality predicate.** Now, the condition for confidentiality and the separation defined for the confidential data must be combined to come up with a formal definition for the confidentiality requirement CR16. The derived formal definition is shown below as a Z schema.

\[
\begin{align*}
\text{HideExactBalance} & \quad \forall \text{State} \\
\exists \text{State}_{9} \cdot \\
\quad & \quad (\text{currAgent} \mapsto \text{reqPurse}) \notin \text{owns} \land \\
\quad & \quad \text{currAgent} \neq \text{terminal} \land \\
\quad & \quad \text{reqPurse} \in \text{dom} (\text{balance}) \Rightarrow \\
\quad & \quad (\text{balance reqPurse}) \neq (\text{balance}_{9} \text{reqPurse}_{9})
\end{align*}
\]

The schema \text{HideExactBalance} includes a confidentiality constraint that must be enforced on the state space of a particular state or states of the ePurse system where \text{HideExactBalance} is observed.
6.2.12.c.3 Results of analysing the ePurse system having CR16

Table 6.16 shows that the analysis of the system with respect to the three user roles in the ePurse system results in the outcome “Simplified”. Recall from Section 3.2.6 that if the predicate generated from back propagating a system specification can be simplified, then according to BCF in Circus, there are no contradictions in the specification of the system being analysed.

<table>
<thead>
<tr>
<th>User role against which the system has being analysed</th>
<th>Seller</th>
<th>Buyer</th>
<th>Terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time taken for back propagation</td>
<td>481 ms</td>
<td>394 ms</td>
<td>653 ms</td>
</tr>
<tr>
<td>Time taken for predicate simplification</td>
<td>1590 ms</td>
<td>1329 ms</td>
<td>1460 ms</td>
</tr>
<tr>
<td>Total time taken for evaluation</td>
<td>2071 ms</td>
<td>1723 ms</td>
<td>2113 ms</td>
</tr>
<tr>
<td>Result of the simplification by the theorem prover</td>
<td>Simplified</td>
<td>Simplified</td>
<td>Simplified</td>
</tr>
</tbody>
</table>

Table 6.16: Results of the mechanised analysis of the ePurse system
6 Evaluation of mechanisation

6.2.12.c.4 Negative test with an automatic counter example

In some scenarios the theorem prover may identify a counter example after running possible combinations to simplify the submitted predicate. Consider the Circus specification of the ePurse system in Figure A.8. Extend the specification in Figure A.8 by introducing the schema $\text{HideIfNotTerminal}$ that represents a confidentiality annotation and the Circus action $\text{ShowAnyBal}$. The schema $\text{HideIfNotTerminal}$ formalises a confidentiality requirement where the system must never reveal the balance of any ePurse if the current user agent is not a $\text{terminal}$. The Circus action $\text{ShowAnyBal}$ formalises a function where the balance of any requested ePurse can be viewed. The descriptions of the variables in both $\text{HideIfNotTerminal}$ and $\text{ShowAnyBal}$ are included in Appendix A.5.3.

$$\text{HideIfNotTerminal} \equiv \exists \text{State} \bullet$$
$$\exists \text{State} \bullet$$
$$\text{currAgent} \neq \text{terminal} \Rightarrow$$
$$(\text{balance reqPurse}) \neq (\text{balance}, \text{reqPurse})$$

$$\text{ShowAnyBal} \equiv \text{var anyPurse} : \text{dom (balance)} \cap \text{ran owns} \bullet$$
$$\text{anyPurseIn?anyPurse} \rightarrow$$
$$\text{reqPurse} := \text{anyPurse};$$
$$\text{showAnyPurseOut} ! (\text{balance anyPurse?}) \rightarrow \text{Skip}$$

The main action of the specification of the ePurse system in Figure A.8 is modified to include $\text{HideIfNotTerminal}$ and $\text{ShowAnyBal}$ as follows.

$$\mu X \bullet \text{SelectAgent} ;$$
$$\begin{align*}
\square \langle \text{ApproveTrnsctn} \rangle \\
\square \langle \text{ShowBalSec} \rangle \\
\square (\langle \text{ShowAnyBal} \rangle ; \langle \text{HideIfNotTerminal} \rangle) \\
\square \langle \text{DoPayment} \rangle \\
\square \langle \text{RecordTrnsctn} \rangle
\end{align*}$$

; X
6.2 Mechanised analysis of confidentiality patterns

Analysing the resulting specification using the mechanisation of BCF in Circus proposed in this research resulted in the Isabelle theorem prover suggesting a counter example. The Isabelle theorem prover command “apply smt” suggested that the variable currAgent might assume the value seller or buyer. In this case, the antecedent in the implication in HideIfNotTerminal will be satisfied resulting in a contradiction between the output action in ShowAnyBal and the consequent in the implication in HideIfNotTerminal.

Figure 6.14: Counter example generated due to an insecure operation
6 Evaluation of mechanisation

6.2.12.d Analysing the confidentiality pattern CP5

The ePurse system discussed earlier contains the confidentiality property CR17 that reflects the confidentiality requirement pattern CP5. Therefore, the same model of that ePurse system will be reused here for analysing a system with a confidentiality property that reflects CP5. As mentioned earlier, the full formal specification for the ePurse system is included in Appendix A.5.3 and it must be noted that this formal specification is just one possible implementation of the system using the Circus notation.

6.2.12.d.1 Formalising the confidentiality requirement CR17

In this section, a possible formalisation of the confidentiality requirement CR17 is specified using the Circus notation. The general definition of CP5 states:

\[
\text{do not reveal the exact value of } x
\]

The confidentiality requirement CR17 of the ePurse system by De Landtsheer and Van Lamsweerde (2005, p. 44) reads:

\[
\text{CR17: agents who are not the card holder should not know whether a state variable is above/below a given threshold.}
\]

Recall that the confidentiality requirement CR17 has been rephrased as follows, to align with the definition of CP5.

\[
\begin{align*}
\text{if} & \quad \text{the agent is not the card holder} \\
\text{then} & \quad \text{do not reveal whether the value } y \text{ is} \\
& \quad \text{above/below a certain threshold } n
\end{align*}
\]

Once again, like the previous confidentiality requirements, this confidentiality requirement requires confidentiality of information under certain conditions. The confidentiality requirement CR17 has the same condition.
The formal definition of the condition for confidentiality has been presented earlier in the analysis for CP4. The confidentiality requirement CR17 has the same condition for confidentiality. In addition, the balance of a requested customer being below a certain threshold \( b_{\text{min}} \) is a condition to be enforced this confidentiality requirement.

\[
\text{balance}_{\text{reqPurse}} \in \{ r : \mathbb{N} \mid r < b_{\text{min}} \}
\]

To hide the set of possible values of \( \text{balance}_{\text{reqPurse}} \) we define the set of possible values of \( \text{balance}_{\text{reqPurse}} \) and \( \text{balance}_{\text{reqPurse}_9} \) to be disjoint.

\[
\text{balance}_{\text{reqPurse}} \in \{ r : \mathbb{N} \mid r < b_{\text{min}} \} \\
\Rightarrow \text{balance}_{\text{reqPurse}_9} \notin \{ r : \mathbb{N} \mid r < b_{\text{min}} \}
\]

The derived formal definition of CR17 is shown below as a Z schema.

\[
\begin{align*}
\text{HideMinBalance} & : \Xi \text{State} \\
\exists \text{State}_{\text{9}} & \bullet \\
& (\text{currAgent} \mapsto \text{reqPurse}) \notin \text{owns} \land \\
& \text{currAgent} \neq \text{terminal} \land \\
& \text{reqPurse} \in \text{dom} (\text{balance}) \land \\
& \text{balance}_{\text{reqPurse}} \in \{ r : \mathbb{N} \mid r < b_{\text{min}} \} \Rightarrow \\
& \text{balance}_{\text{reqPurse}_9} \notin \{ r : \mathbb{N} \mid r < b_{\text{min}} \}
\end{align*}
\]

The schema HideMinBalance includes a confidentiality constraint that must be enforced on the state space of a particular state or states of the ePurse system where HideMinBalance is observed. The main action of the ePurse system has been modified as follows for this analysis.
The ePurse system in the specification in Figure A.8 having the modified main action as above, is analysed using the mechanisation. The results of this analysis is discussed next.

### 6.2.12.d.2 Results of analysing the ePurse system having CR17

Table 6.17 shows that the analysis of the system with respect to the three user roles in the ePurse system results in the outcome “Simplified”. Recall from Section 3.2.6 that if the predicate generated from back propagating a system specification can be simplified, then according to BCF in Circus, there are no contradictions in the specification of the system being analysed.

<table>
<thead>
<tr>
<th>User role against which the system has been analysed</th>
<th>Seller</th>
<th>Buyer</th>
<th>Terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time taken for back propagation</td>
<td>481 ms</td>
<td>394 ms</td>
<td>653 ms</td>
</tr>
<tr>
<td>Time taken for predicate simplification</td>
<td>1590 ms</td>
<td>1329 ms</td>
<td>1460 ms</td>
</tr>
<tr>
<td>Total time taken for evaluation</td>
<td>2071 ms</td>
<td>1723 ms</td>
<td>2113 ms</td>
</tr>
<tr>
<td>Result of the simplification by the theorem prover</td>
<td>Simplified</td>
<td>Simplified</td>
<td>Simplified</td>
</tr>
</tbody>
</table>

Table 6.17: Results of the mechanised analysis of the ePurse system
6.2.13 A comparison of results of the mechanised analysis

In order to compare the results of the mechanised analysis shown in Table 6.8, Table 6.11, Table 6.14, Table 6.16 and Table 6.17, we must first have a way to identify the relative size of each Circus specification that was analysed. The author does not know of any prior work on comparing the relative size of two formal specifications in the Circus notation. Further, the reader must also be informed that the implemented back propagation process in the CFAT tool uses regular expressions and pattern matching for renaming variables in the generated predicate. Alternatively, with the right programming skills, one could have implemented the same mechanisation using a functional programming language. In this scenario, regular expressions and pattern matching might not be required.

BCF laws (see Table 2.4) used in the analysis in this chapter are:

- **bw** external choice
- **bw** sequence
- **bw** assignment
- **bw** guard
- **bw** input prefix
- **bw** output prefix

The **bw** sequence and **bw** external choice laws are for the composite operators, sequential composition and external choice respectively. Apart from that, BCF back propagation laws for the atomic actions in the Circus notation manipulate the incoming predicate during the back propagation process. For example, **bw** guard, **bw** input prefix and **bw** output prefix laws add additional text to the predicate being calculated. However, the **bw** assignment law applies pattern matching through regular expressions to manipulate the predicate being calculated. It is expected that as the predicate grows in size, more computing time is required for pattern matching. Therefore, a correlation can be
expected between the number of “assignment” actions in a given specification and the average time taken for back propagating the same specification. Such a correlation is reflected in Table 6.18. However, this is less true for \textbf{bw guard}, \textbf{bw input prefix} and \textbf{bw output prefix} laws where:

Application of the input and output laws depends on the channelset \( L \) upon which the specification \( S \) is lifted. If there are output actions in \( S \) that uses a channel in \( L \) additional text will be introduced to the back propagated predicate, increasing its size. Likewise, base on whether an input action uses a channel in the channelset \( L \) for communication, the resulting text introduced to the back propagated predicate will be different. Therefore it will be difficult to derive a meaningful correlation between the number of input actions in a \textit{Circus} specification and the time it takes for back propagating that specification.

The back propagation of the guarded action also introduce new text to the back propagated predicate. However, this text depends on the size of the \textit{guard} in a guarded action. Therefore, existence of a guard will have little impact on the time taken for back propagation.

It must be strongly noted that this is too little a dataset to make a strong conclusion regarding the correlation. A much larger dataset from a range of systems must be analysed if we are to make a more accurate conclusion regarding the relation between the specification size and time taken for its mechanised analysis.
## 6.2 Mechanised analysis of confidentiality patterns

<table>
<thead>
<tr>
<th>Context</th>
<th>Bank information system</th>
<th>Phone book system</th>
<th>Secure electronic examination system</th>
<th>ePurse system</th>
<th>ePurse system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality requirement</td>
<td>CR18</td>
<td>CR14</td>
<td>CR21</td>
<td>CR16</td>
<td>CR17</td>
</tr>
<tr>
<td>Confidentiality pattern</td>
<td>CP1</td>
<td>CP2</td>
<td>CP3</td>
<td>CP4</td>
<td>CP5</td>
</tr>
<tr>
<td>Number of Circus actions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Guarded action</strong></td>
<td>5</td>
<td>4</td>
<td>19</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>Input action</strong></td>
<td>8</td>
<td>5</td>
<td>15</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td><strong>Output action</strong></td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Assign action</strong></td>
<td>6</td>
<td>5</td>
<td>34</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>The average back propagation time</td>
<td>434 ms</td>
<td>252 ms</td>
<td>2792 ms</td>
<td>1477 ms</td>
<td>1587 ms</td>
</tr>
<tr>
<td>The average simplification time</td>
<td>1271 ms</td>
<td>1656 ms</td>
<td>2095 ms</td>
<td>1095 ms</td>
<td>1219 ms</td>
</tr>
</tbody>
</table>

Table 6.18: A comparison of the average analysis times for different systems
6 Evaluation of mechanisation

6.3 Summary

The main contribution of this chapter is the analysis carried out to evaluate the mechanisation of BCF in Circus, developed under this research. This fulfills the objective of this chapter by demonstrating that the mechanisation of BCF in Circus is practically applicable.

In this chapter:

- the mechanisation of BCF in Circus has been used to analyse systems with confidentiality requirements that reflect five different types of confidentiality requirement pattern identified in this research.
- the issue with a weak specification and how the weak specification maybe strengthened has be demonstrated.
- the concept of negative testing has been discussed and the results of carrying out a negative test has been discussed.
- the results obtained from executing different sizes and complexities of case studies has been critiqued.
- a possible correlation between the number of “assignment” actions in a given specification and the average time taken for back propagating the same specification has been identified. It must be noted that this maybe due to the programming style used for mechanising BCF where pattern matching has been used text replacement in a predicate.
7 Evaluation

This chapter presents an overall evaluation of the research presented in this thesis.

7.1 Introduction

Recall from Chapter 1 that the hypothesis of this research states that:

\[ a \text{ practically applicable approach exists that supports the process of analysing system models using Banks's Confidentiality Framework (BCF) (Banks, 2012) to verify if those models respect the integrated confidentiality requirements pertaining to data leakage through legitimate channels. } \]

The author argues that the hypothesis has been fully satisfied through the proposed mechanisation in this thesis. The argument for the practical applicability of the proposed mechanisation approach has been justified by showing the time taken for analysing specifications of different systems with varying sets of functions. A possible approach for calculating the efficiency of the mechanisation has been discussed through a comparison between the time taken for a manual run of a BCF in Circus based analysis of a system and a mechanised run of the same analysis.

The author is quick to acknowledge that the current mechanisation does not support the analysis of recursion and parallel processes in systems. This is due to the current limitations in BCF in Circus, as discussed in the next section. For this reason, the mechanisation supports only sequential access systems.
7 Evaluation

7.2 Factors that could have influenced the quality of the analysis

The following factors could have strengthened the overall value of the outcome of the analysis carried out in this thesis.

Availability of real Circus specifications of Systems. If the case studies conducted in this research were based on real Circus specifications of systems or on real confidentiality requirements relating to data leakage then the justification of the advantages brought by the proposed mechanisation could have been stronger. Having to invent and hand-craft these specifications has been one of the limitations of this research as detailed in Section 7.4.

Availability of descriptions of real-life systems. The research conducted in this thesis required a catalogue of system descriptions of real-life systems, that could be used as a benchmark for testing the mechanisation of BCF in Circus. However, the author was not able to find a literature backed catalogue of formal specifications or even system descriptions of real-life systems having a confidentiality requirement. Therefore, such a catalogue has been compiled through a literature search, as part of this research. Many limitations have shaped the catalogue that was compiled eventually. However, if such a comprehensive catalogue was available, results from analysing systems in that catalogue would have contributed towards evaluating the value that the mechanisation can bring to the system engineering discipline. The author believes that the catalogue provided in Chapter 5 can be a starting point for such requirements for a benchmark.

Support for recursion in BCF in Circus. Banks (2012, p. 148) discussed a method for deriving an invariant obligation that is required to back propagate a loop body in a recursion. However, he stated that the method cannot be used in all scenarios and therefore sometimes one must resort to intuition to identify the invariant obligation.
7.3 Benefits derived from the mechanisation

“We leave the problem of devising more sophisticated techniques for identifying invariant obligations for future work.”

(Banks, 2012, p. 148)

In some scenarios, the user may identify an invariant obligation. However, Banks does not provide a mechanisable law for these scenarios. Hence, the current version of the CFAT tool is not designed to provide support for back propagating recursion constructs. Section 3.4.3 discusses a scenario where a data leakage may occur through recursion.

Support for parallel processes in BCF in Circus. BCF in Circus does not have a mechanizable back propagation law for parallel processes that can be used for automating the back propagation of parallel constructs. This is a limitation of BCF in Circus as discussed in Section 2.8. Concurrent access to systems is an important aspect of information systems. A comprehensive analysis of confidentiality in a concurrent access system must involve analysing the parallel process blocks in the system specification.

7.3 Benefits derived from the mechanisation

The proposed mechanisation was intended to provide a number of benefits that would enhance the value of BCF in Circus. The intended benefits have been discussed in Section 3.1. The following is a discussion on the extent to which these intended benefits have been realized.

Practicality. The evaluation of the proposed mechanisation has shown that the proposed mechanisation is practically applicable. This has been demonstrated by presenting the time taken for analysing a number of hand crafted systems that are based on scenarios supported by literature. Based on the However, the performance of the mechanisation can be improved on many fronts. For example,
Evaluation

currently the \textsc{LaTeX} model of the system and the HOL based back propagated predicate needs to be manually submitted to the CZT and the Isabelle theorem prover respectively. This process could be automated to achieve a seamless process for analysing systems using the proposed mechanisation of BCF in \textit{Circus}.

\textbf{Suitability.} Apart from practicality, another intended purpose of the mechanisation of BCF in \textit{Circus} is to show whether systems with different types of confidentiality requirements can be analysed. Two types of confidentiality properties that are supported by BCF in \textit{Circus} has been identified. The suitability of the mechanisation to detect both types of confidentiality requirements supported by BCF in \textit{Circus} has been demonstrated through the use of negative testing (see Section \textit{6.2.11}). It must be noted that the testing for suitability carried out in this thesis is not strong as the proposed mechanisation does not support recursion and parallel constructs, due to limitations with BCF in \textit{Circus}. Therefore, in an ideal world, these tests will fall short of reflecting real world scenarios of the consistency of requirements in a system specification. Developing mechanizable laws for recursion and parallel constructs of the \textit{Circus} notation is a further work required.

\textbf{Efficiency.} The comparison between the mechanised versus the manual analysis of a trivial system in Section \textit{3.5.1} has shown that the proposed mechanisation of BCF in \textit{Circus} in this thesis is multifold efficient than the manual approach. While the relative efficiency has been shown for a trivial example, there is no way to conclude whether the derived ratio of the Temporal Efficiency (see Section \textit{3.5.1}) will be constant as the size and complexity of the specifications change. The author believes that the discussion on relative efficiency between the manual versus the mechanized analysis approach in Section \textit{3.5} has demonstrated that an efficient approach for analysing systems using BCF in \textit{Circus} has been achieved through the mechanisation.
7.4 Contributions and Limitations

In this section, a discussion of the contributions and some of the limitations of those contributions are presented.

Catalogue of case studies with a confidentiality requirement. The catalogue of case studies with a confidentiality requirement is an original contribution made in this thesis as far as the author is aware of. As an initial catalogue, this provides a good collection of case studies that has been derived through a systematic process. Further, this collection can be used by other researchers to analyse other dimensions in relation to a confidentiality requirement literature. For example, context where security is most sought after. Researchers working on vertical areas of research such as eliciting confidentiality requirements and various formalisms of confidentiality requirement specifications can use scenarios from the case studies in this catalogue to align their research with this existing literature discussing different scenarios with a confidentiality requirement. This is because there is a vacuum of such real-world case studies.

“there is a vacuum of real-world case studies and experience reports on how confidentiality requirements are dealt with in practice”

(Gurses et al., 2005, p. 102).

The major limitation of the proposed catalogue in this thesis is its derivation process. Because the scenarios were identified by focussing on pages in a paper where a certain keyword was found, prospective descriptions of confidentiality related requirements in other pages of the paper might have been missed, if such a discussion did not contain the keyword ‘confidential’. Because of the limited set of relevant papers that were identified, it was necessary to consider discussions where confidentiality was mentioned in a very general way without giving much details about the context except a reference for the system environment, such as ‘a banking system’, ‘an electronic voting system’, etc.
**Generalized patterns of confidentiality requirements.** Generalized patterns of confidentiality requirements is an original contribution as far as the author is aware of. These patterns will greatly help the confidentiality engineering community as the pattern catalogue can be used as a baseline benchmark to address confidentiality properties in the literature. However, while using the catalogue as a benchmark, the user must also be aware that the catalogue could be improved on many grounds. The pattern catalogue can also be useful for comparing different formalisms that can be used to analyse systems with a confidentiality requirement.

Assumptions were made when standardizing some confidentiality requirements described in some papers. Those assumptions were necessary because some of the requirements were not clearly stated in terms of:

- the exact data that needed concealing,
- the user roles that were supposed to be unauthorised, with regards to a particular piece of data in a given context.

Some patterns might be weak because the confidentiality requirements from which those patterns were derived were partially hand-crafted. The author acknowledges this weakness as a limitation. This limitation can only be overcome when systems and their confidentiality requirements, considered as source material for such a catalogue, are specified in a clear and unambiguous manner.

Further, because of the above mentioned limitation, the author anticipates further patterns, should there be a more in-depth literature search exercise for confidentiality properties in systems.

**Confidentiality Framework Application Tool.** The CFAT tool is an original contribution of this research. A mechanised back propagation apparatus does not exists to the author’s knowledge. Banks acknowledges this as an impediment to BCFs deployment ([Banks] 2012, p. 187). With the CFAT tool, this has been addressed. The CFAT tool has been developed as part of this research. The
7.4 Contributions and Limitations

biggest advantage of the CFAT tool is that it makes the task of back propagation practically applicable.

The ability to generate a LaTeX based Circus specification of a system being analysed is an important feature of the CFAT tool. Engineers can type check and syntax check the generated Circus specification to provide a degree of certainty about the validity of specification.

The major limitation hindering the use of the CFAT tool is that the user must be equipped with expertise in using the CFAT notation. One further improvement required is the ability to automatically submit the generated LaTeX based Circus specification and the Isabelle theorem file to CZT and Isabelle theorem prover respectively.

Further to the limitations in the contributions that has been made with this thesis, there were other limitations that confined the landscape available for this research. Following are such limitations.

**Programming style not scalable.** The use of an imperative programming approach to code the back propagation logic used in the proposed mechanisation tool is not scalable. This is one of the shortcomings of the tool proposed in this thesis. A more appropriate approach would have been to use functional programming. However, a functional programming approach was not pursued because the author does not have the relevant background.

**Having to hand-craft specifications.** Since formal system specifications were not available for the systems modelled in the case studies in this thesis, specifications of those systems had to be hand-crafted. The resulting specifications might not have reflected the real life scenarios in those contexts.

**Unable to analyse systems with parallel processes.** If BCF in Circus supported parallel processes then the mechanisation of BCF in Circus could have been extended and used to analyse multi-user parallel processing environments. Such an analysis
would have reflected a much closer view about the consistency of the requirements in the system, as many systems designed for use by members of an organisation allow parallel access.

**Unable to analyse the recursion construct.** BCF in *Circus* does no have a back propagation law for recursion. Because of this, the recursion construct had to be bypassed during the back propagation stage of every analysis carried out in this research. Banks (2012, p. 148) discussed how one may approach to back propagate the recursion construct in *Circus*. However, this discussion is in its early stages and needs further research to produce a mechanizable back propagation law for recursion. As such, this endeavour is not within the scope of this thesis.

**The global state promotion dilemma.** The technique adopted by BCF for reasoning about confidentiality in systems is to make specifiers define the level of separation between the real and the twin system and use the result of the back propagation application on the system specification to prove this separation. A separation for a state variable can be defined if that variable exists in the global state space of the system. This becomes an issue if the engineer wants to protect the value of a particular function application such as $f(x)$ where $f$ is a function in the global state space whereas $x$ is a runtime variable whose value is provided by the user during the execution of the program. Therefore, in order to define the required separation, the value of $x$ must be assigned to a global state variable $y$ and subsequently the formalisation of the confidentiality requirement must use $f$ and $y$ rather than $f$ and $x$. 

266
7.5 Mechanization vs. manual back propagation

Banks (2012) did not present any analysis on the feasibility of the manual application of BCF. Rather, he did state that:

“The lack of any dedicated tool support for our platform is arguably the main impediment to its deployment.” (Banks, 2012, p. 187)

The benefit of the mechanisation can be visible when the results of a manual back propagation of a system and the mechanised back propagation of the same system are compared side by side. The manual back propagation shown in Section 4.2 is for a system formalized with a Circus specification that contains one input action, one assignment action, one guarded action and one output action. Manual analysis of that system using BCF in Circus takes roughly one hour while the mechanized analysis of the same system takes 120 milliseconds as stated on Section 4.2.

A more promising set of results of a mechanised analysis that includes the time taken for back propagating non-trivial system models is shown in Table 6.18. For example, the mechanized back propagation of the secure examination system with 71 atomic actions takes an average of 2792 milliseconds or 2.79 seconds.

7.6 A critical analysis of the adopted mechanisation approach

Circus is not ideal as a formalism to adapt for instantiating BCF. The most important issue is that there is no official BNF for the Circus notation. Some researchers do include variations of what they say is the BNF of Circus such as in the papers by Woodcock and Cavalcanti (2001a, P. 292), Woodcock and Cavalcanti (2002a, P. 185), Sampaio et al. (2003, P. 149), Freitas (2005, P. 13), Cavalcanti and Woodcock (2002, P. 149), Oliveira et al. (2006, P. 3) and Oliveira et al. (2009, P. 5). However, unlike the ISO standardisation of the Z notation (ISO/IEC, 2002) there is no single standardisation of the Circus notation nor...
there is a working group that identifies themselves as the caretaker of this formalism.

Another issue with the Circus notation is that there are no reliable and stable tool support that can cater for type checking and model checking Circus specifications. The only type checking tool available for type checking a Circus specification is CZT. However, it does not contain useful error messages, syntax highlighting or navigation capabilities for Circus specific constructs in the specification. Ye and Woodcock (2017) has proposed an approach for model checking Circus specifications by linking Circus to CSP||B. Their approach involves transforming the state part of a Circus specification to a B machine while converting the behavioural part to CSP and finally using ProB (Leuschel and Butler 2003) to model check the resulting CSP||B specification. The problem here is that the limitations in CSP||B confine the type of Circus specifications that are supported in this approach. Ye and Woodcock (2017, p. 94) discusses these limitations. Tools such as CRefine (Oliveira et al., 2008) and JCircus (Barrocas and Oliveira, 2012) have been developed in the past to cater for a specific need of a research conducted in the past. However, to the author’s knowledge there is no evidence that either these tools or any other Circus tools are actively being developed and maintained in such a away that other researchers can build upon its code-base. Further, research on the Circus notation has yet to be taken up by the broader research community rather than being confined to a small cohort of collaborating academics from a limited set universities, as is the case at present.

Isabelle theorem prover is not ideal for simplifying the predicate generated from back propagating a formal specification based on Circus. One reason is that mechanical theorem proving used in Isabelle theorem prover requires guidance by an expert. On the contrary, finite-state verification techniques such as model checking can fully be automated.

---

1 “CSP||B is a combination of CSP and B aiming to introduce behavioural specification into state-based B machines. The B method characterises abstract state, operations with respect to their enabling conditions and their effect on the abstract state, while CSP specifies overall system behaviour. But different from Circus, the CSP specification and B machine in CSP||B are always orthogonal. They are individually complete specifications and can be checked separately” (Ye and Woodcock 2017, p. 76).

2 “Finite-state verification refers to a set of techniques for proving properties of finite-state models of computer systems” (Dwyer et al., 1998, p. 7).
7.6 A critical analysis of the adopted mechanisation approach

“In contrast to mechanical theorem proving, which often requires guidance by an expert, most finite-state verification techniques can be fully automated, thus relieving the user of the need to understand the inner workings of the verification process” (Dwyer et al., 1998, p. 7).

When compared to theorem proving, the advantages of model checking include full automation and the ability to generate counter examples that help in debugging (Ye and Woodcock, 2017, p. 73). However, a proof based approach excels in certain aspects such as being able to handle very complex systems because it does not have to directly check every state and also because its logics are typically more expressive (Amjad, 2004, p. 16). The ideal scenario would be to use a platform that has an efficient combination of model checking and theorem proving. Such combinations have been proposed in the past researchers such as Arkoudas et al. (2004), Amjad (2004), Aagaard et al. (1999), Bjørner et al. (1997), Dingel and Filkorn (1995), Shankar (1996), McMillan (1999) and Rajan et al. (1995).

Generating and submitting the back propagated predicate in a format that is supported by such a combined platform will be a valuable further improvement for the mechanisation approach proposed in this thesis.

The CZT platform was considered for building custom extensions on top of it, to support the mechanisation of the calculations required by BCF in Circus and subsequently for code generation. However, the architecture and inner workings of the CZT editor was very complex that demand a steep learning curve before the CZT editor could be extended. Kimber (2007) considered the CZT editor as an input interface which he planned to extend for code generation to produce PerfectDeveloper (Crocker, 2003) code from Object-Z specifications. However, after reviewing the CZT editor, Kimber (2007) concluded that the DTDs and schemas of CZT projects were quite impenetrable.

HiVe Mathematical Tool-kit is not ideal for the purpose of the mechanisation proposed in this thesis. This is because the theory package does not provide a set of tactics which can be used in dispatching automatic proofs for theories.
7 Evaluation

7.7 Critical factors that would have altered the direction of this research

The following are some critical factors that would have altered the direction of this research, if they were available at the beginning of this research.

Knowledge of Isabelle/Isar and Standard ML. If the author had been well versed with Isabelle/Isar\footnote{Isabelle/Isar provides an interpreted language support for interactive theorem proving in the Isabelle theorem prover whereas the host language of the Isabelle theorem prover is standard ML\cite{Wenzel2013}.} and standard ML\cite{Wenzel2013} then the author could have embedded Circus and the block structure\cite{Banks2012} of BCF in Circus inside the Isabelle theorem prover, like HOL-Z\cite{Brucker2003}, Z and HOL\cite{Bowen1994}, CML\cite{Woodcock2012} or Isabelle/Circus\cite{Feliachi2012}. Further, using ML, the author could have mechanised the back propagation logic and subsequently simplify the predicate within the Isabelle theorem prover.

Detailed developer guide for CZT. If there was a clear documentation regarding the architecture of CZT and how one may extended it, the author could have extended CZT to generate the theorem file from the standard \LaTeX\ format supported by the CZT tool. In this scenario, the user could have specified the system using the Circus notation rather than having to get expertise in using the CFAT notation which is required in the current mechanisation proposed in this thesis. The author attempted to extend CZT during the course of this research. However, similar to Kimber\cite{Kimber2007}, the author found it impenetrable within the time frame of this research.

"Indeed, the DTDs and schemas downloaded from existing projects such as CZT were quite impenetrable."\cite{Kimber2007}
7.8 Further work

The contributions made in this thesis has enabled many interesting directions of further research that could be explored in future.

Comparative analysis of BCF in Circus. The proposed mechanisation in this thesis and other similar mechanisations proposed in the literature, for analysing systems with a confidentiality requirement, can be compared side by side on the basis of analysis time and competence required. Such as exercise may reveal the comparative strengths and weaknesses of the compared approaches. BCF in Circus supports certain sub classes of confidentiality properties as stated in Section 5.3. Therefore, it is important to confine this comparative analysis to systems with properties that are supported by all compared approaches.

Extension of the mechanisation of BCF in Circus to support additional constructs. If further research on BCF in Circus results in the introduction of support for data leakage analysis of parallel processes, then such extensions can easily be adapted in the mechanisation proposed in this thesis. Further, results of such an analysis will more closely reflect the information flows in a real life execution of a system with parallel processes rather than an exercise that cannot analyse a system with parallel processes.

Embedding in Higher Order Logic (HOL). An on-going work (Zeyda et al., 2017) at the University of York is concerned with embedding Circus in Isabelle/UTP. This work further involves translating Circus notations into corresponding operators within this embedding. The confidentiality framework can be mechanized as an Isabelle/HOL theory by extending this embedding, so that Circus models with a confidentiality requirement can be directly written in the Isabelle theorem prover similar to HOL-Z and Isabelle/Circus. Tactics can be developed to facilitate the simplification of these system models.
7 Evaluation

**Prototyping for black-box testing.** *Circus* formal models may be translated to executable OCAML programs for rapid prototyping. These prototypes may be used by security experts to conduct data secrecy tests.

**BCF and Event-B.** BCF can be adopted for the Event-B notation provided that there is a UTP semantics for the notation. The back propagation laws of BCF can then be mechanized by extending the RODIN tool. The RODIN tool already has support for modelling and refinement of Event-B specifications. This effort may result in a more integrated tool that supports confidentiality verification.

**Improving the efficiency and effectiveness of the mechanisation.** The proposed mechanisation can be improved on many grounds so that engineers can save time when using the mechanisation. Some potential areas of the mechanisation that can be improved to achieve a better efficient and effective performance from the tool include:

- Building support in the CFAT tool for a Controlled Natural Language (CNL) as detailed in page 275-276. CNL can lower the barrier of entry for the CFAT tool by allowing users to specify systems using a natural language rather than having to use a structured notation.

- Automating the submission of the back propagated predicate to the Isabelle theorem prover. This can be possible through the use of command line tools of the Isabelle theorem prover.

- Automating the validation of the generated \LaTeX specification using CZT. This can be possible through the use of command line tools of CZT.

- Develop tactics in the Isabelle theorem prover for improving the automation of the simplification. Most probably, this will be an extension to the HiVe mathematical toolkit.
Techniques to improve the identification of data leakage risks. The process followed in this thesis, for analysing systems with a confidentiality requirement, can be strengthened by improving the confidentiality requirement elicitation stage of the process. The current approach involves formalizing the given confidentiality requirement and including it as part of the formal specification of the system. However, if the system requirements could be analysed using techniques for identifying security requirements from business goals, it can only help to improve the set of security requirements of the system. The following are some useful techniques in this domain that future researchers can utilize.

Abuse cases
Abuse cases utilizes use case models to model complete interactions between a systems and one or more actors (McDermott and Fox, 1999). In comparison to normal use cases, the result of the interaction in an abuse case is harmful to one of the actors or stakeholders of the system.

“In a requirements phase, abuse case models can be used to increase both user and customer understanding of the security features of a proposed product.” (McDermott and Fox, 1999, p. 63)

Misuse cases
Sindre and Opdahl (2005) introduced misuse cases, which are inverted use cases that describe functions that the system should not allow. Use cases and misuse cases are included in the same diagram of the system model, rather than being on different diagrams like use cases and Abuse cases. Misuse-case diagrams link regular use cases to both threats and potential countermeasures which aids in prioritization of requirements since the real cost of implementing a use case includes the protection needed to mitigate all serious threats to it (Sindre and Opdahl, 2005, p. 41).
7 Evaluation

Deviational techniques  A security analysis using deviational techniques such as HAZOP (Kletz, 1999) on use cases may identify security requirements which otherwise might have slip through unnoticed. Srivatanakul (2005) demonstrated that deviational techniques can be used on use cases in UML to identify security requirements.

Anti-goals  Another technique for identifying security requirements is through the use of anti-models. An anti-model includes anti-goals, which are attackers own goals that are intentional obstacles to security goals, set up by the attackers to threaten security goals (Elahi et al., 2010, p. 21). Lam-sweerde (2004) extended KAOS, a goal-oriented security requirements engineering methodology to capture anti-goals and other obstacles that capture exceptional behaviour.

Tool-supported automation of translations. From the initial business goals to the formal specification of confidentiality integrated systems, there are many stages where there are requirements for syntactic translations of requirements between stages. Figure 7.1 shows some of these stages with a T. Further work on automating the translation at each stage is recommended to introduce a systematic process for each translation. One such technique for enabling such translations is to use a Controlled Natural Language (CNL) for describing systems. A small introduction to CNL is given on page 275-276.
Two important issues that one must address when writing a system requirement specification for an information system are that:

- the organisational structure and rules that are to be incorporated into the information system must be described in a language that can be understood by all stakeholders.
- the specification should be unambiguous so that traceability can be established between the system requirement specification and its eventual implementation.

Unambiguity is also required in a system requirement specification in order to avoid inconsistencies between stakeholders as well as in order to formally reason about system models. One way to ensure unambiguity in a system requirement specification is to use a Controlled Natural Language (CNL) (Schwitter, 2010).

A CNL is a subset of a natural language where the grammar and vocabulary are restricted in a systematic way to reduce the ambiguity and complexity found in a full natural language (Schwitter, 2010, p. 1113). Some CNLs include INCOSE (Condamines and Warnier, 2014, p. 36), Attempto Controlled English (ACE) (Fuchs et al., 2008) and Computer-Processable Language (CPL) (Clark et al., 2005).

RuleCNL by Feuto Njonko et al. (2014) is a domain independent CNL that systematically embeds domain facts and terms in grammatically correct sentences. In an organisational setting these domain fact and terms can be synonymous with organisational rules and structure.
“An ontology models a part of the world. Such a model can be used by humans and computers in order to establish a common ‘understanding’ of relevant concepts and the relations between them.”

(Schumacher, 2001, p. 32)

A future area of work can be to work on a CNL that adopts the concept of embedding organisational rules and structures similar to RuleCNL. This CNL can be designed to provide a set of necessary patterns that can be used to describe a simple system in such a way that the system description can be systematically translated into a Circus specification. Table 7.1 shows the mapping between some components of a system requirement specification and a Circus specification. A set of rules can be proposed that can be used to describe the organisational structure, rules and regulations using natural language. Subsequently, a mapping for translating compliant sentences to the Circus notation can be provided.

<table>
<thead>
<tr>
<th>System requirement specification</th>
<th>Structures of the Circus notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organisational structure</td>
<td>data types, data objects, global constants</td>
</tr>
<tr>
<td>Organisational rules and regulations</td>
<td>state invariants</td>
</tr>
<tr>
<td>Operations performed in the organisation</td>
<td>actions</td>
</tr>
<tr>
<td>Who can perform the functions</td>
<td>channelsets (representing user roles)</td>
</tr>
<tr>
<td>Which staff are included in each user role</td>
<td>state invariants that define elements of sets</td>
</tr>
</tbody>
</table>

Table 7.1: Mapping a system requirement specification to structures in a Circus specification
A comprehensive set of sentence patterns are not required for using a CNL, but rather a minimal set of patterns that can be used to derive unambiguous and grammatically correct sentences to describe the structure, rules and regulations of an organisation is enough as a start. Nonetheless, the grammar can be extended later to support a richer set of patterns of sentences in order to describe other identified components of a system requirement specification as listed in Table 7.1.
7 Evaluation

![Diagram of system requirements](image)

Figure 7.1: Some possible stages and paths that can be taken when moving from business goals to Circus specifications
A Appendix

A.1 Modelling a system using the Circus notation

One way to model a system using the Circus notation, is to model the individual components of the system and then combine them into a single system definition called a Circus specification. The specifications of individual components are called Circus paragraphs\(^1\). A Circus specification may contain zero, one or many Circus paragraphs. This section describes how the Circus notation has been used to formally model the range of systems discussed in this thesis.

The application of the Circus constructs, discussed in this section, are illustrated using the specification of an Online expenditure tracker system (see Figure A.1). The components of this specification are explained one-by-one, as we progress through the following sub-sections. In summary, this system allows an on-line customer to purchase items. Money spent on every purchase is tallied with his/her previous purchases. The system also allows a customer to check his/her total expenditure.

A.1.1 Defining the data types and state variables

A.1.1.a Data types

In this thesis, systems are modelled as abstract formal specifications. Significant examples of system scenarios are taken from the literature. The data structures of these systems are modelled as abstract sets of elements or relations and functions.

---

\(^1\) A Circus paragraph is a syntactic structure of the Circus specification language (Sampaio et al., 2003).
A Appendix

Basic types In the Circus notation, an abstract data type can be defined by providing the name of the type enclosed in a pair of square brackets. Such type definitions are called basic types (Spivey, 1989, p. 47). Similarly, multiple types can be defined by providing a comma separated sequence of type names.

\[
\text{e.g. } [\text{CUSTOMER}, \text{ITEM}]
\]

In the example, the basic types CUSTOMER and ITEM are defined using the statement \([\text{CUSTOMER}, \text{ITEM}].\) The basic type CUSTOMER represents the set of all the customer identifiers and the basic type ITEM represents the set of all the item identifiers used in the system.

Free types A type with a finite number of elements can be defined in Circus using the free type construct. A free type can be used to define a set of distinct constants (Spivey, 1989, p. 82).

\[
\text{e.g. } \text{Status} ::= \text{InUse} \mid \text{NotInUse}
\]

The statement \(\text{Status} ::= \text{InUse} \mid \text{NotInUse}\) defines the free type \(\text{Status}\) to be a set containing exactly two distinct elements that represent two values. The free type definitions in this thesis use simple free types. More complex free types can be defined as show in (Spivey, 1989, p. 82).

A.1.1.b State variables and invariants

State variables are variables that are declared in the global scope of the system. For example, the data structures \(\text{spent}, \text{price}\) and \(\text{currentCustomer}\) need to be accessible throughout the scope of the process \(\text{Online_expenditure_tracker}\) (see Figure A.1) and so they are defined as state variables. The description of the state variables in the specification are included in the Table A.1.
A.1 Modelling a system using the Circus notation

Table A.1: Online expenditure tracker - Description of the state variables

<table>
<thead>
<tr>
<th>State variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spent</td>
<td>A function that identifies the amount spent by a customer, if any.</td>
</tr>
<tr>
<td>price</td>
<td>A function that identifies the price of an item, if any.</td>
</tr>
<tr>
<td>currentCustomer</td>
<td>The user who is currently logged into the system.</td>
</tr>
</tbody>
</table>

The system enforces constraints encoded as state invariants on the values that can be assumed by the state variables. These constraints encode the policies of the parent organisation. The description of these state invariants are included in Table A.2

Table A.2: Online expenditure tracker - Description of the state invariants

<table>
<thead>
<tr>
<th>State invariants</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentCustomer ∈ (dom (spent))</td>
<td>The current user who is logged into the system must have a spent value recorded in the system.</td>
</tr>
<tr>
<td>CUSTOMER, ITEM</td>
<td></td>
</tr>
</tbody>
</table>

**State**

- spent : CUSTOMER → N
- currentCustomer : CUSTOMER
- price : ITEM → N
- currentCustomer ∈ (dom (spent))

**channel**

- mySpentAmountOut : N
- buyItemIn : ITEM

**channelset**

- Customer == \{| buyItemIn, mySpentAmountOut | \}

**process**

`Online_expenditure_tracker ≡ begin`

```
state State
  RecordMyReciept ≡ var buyItem : ITEM •
  buyItemIn?buyItem→
  ((buyItem? ∈ dom (price))∧
    spent := spent ⊕ \{ (currentCustomer →
      (spent currentCustomer + price buyItem?)) \})

  GetMySpent ≡ mySpentAmountOut !(spent currentCustomer) → Skip

  • μX • (( □ RecordMyReciept ) ; X)
```

**end**

**Figure A.1:** Specification of Online expenditure tracker - code block 1 of 1
A.1 Modelling a system using the Circus notation

A.1.2 Establishing the communication channels

The system must have communication channels so that the customer can interact with the system, by way of inputs and outputs.

Channels  A channel can be declared by using the syntax \texttt{channel } x^+ : T, where \texttt{channel} is a keyword, \(x^+\) denotes a single or a comma separated sequence of channel names and \(T\) denotes the type of data that can be communicated through the channels. If a \textit{Circus} channel is declared with this syntax, it can be used by the system for communicating with the environment by way of inputs and outputs. For example, the statement;

\begin{verbatim}
channel itin : ITEM
\end{verbatim}

in the running example defines a channel called \textit{itin} with the data type \textit{ITEM} to input item identifiers. It is important to note here that channels are strongly typed in \textit{Circus}, and hence a channel will restrict the data it communicates based on its data type\(^2\)

In a system environment, confidentiality requirements may be enforced on all users or various requirements may be enforced on various sub-groups of users. BCF (see Section 2.3) provides constructs to enforce confidentiality requirements on a defined sets of channels. In this regard, a confidentiality requirement may be enforced on a user by enforcing the confidentiality requirement on the set of channels through which the user communicates with the system.

Channel sets  A set of channels with a given name is called a channelset. A channelset is defined using the notation \texttt{channelset ::= } \{ x^+ \}, where \texttt{channelset} is a keyword and \(x^+\) denotes a single or a comma separated sequence of channel names. For example, the statement;

\begin{verbatim}
channelset Customer == \{ buyItemIn, mySpentAmountOut \}
\end{verbatim}

\(^2\) A channel can also be defined without a data type. Such channels are called synchronisation points and they do not communicate any value (Freitas, 2005).
A Appendix

in the running example defines a channelset called Customer that represents the set of channels allowed to a customer of the system.

A.1.3 Defining the system operations

The term ‘action’, in the Circus terminology refers to any operation of a system and hence, from now on this term will be used to refer to system operations accordingly. Here, we define the syntax of the common actions used in the Circus specifications discussed in this thesis.

**Primitive actions.** The action Skip terminates immediately and does not make any changes to the system state (Oliveira et al., 2009, p. 7).

**Declaring a variable.** A variable can be declared using the statement \texttt{var \( v \) : \( T \)} where \( v \) denotes a single or a comma separated sequence of variable names and \( T \) denotes the datatype of \( v \) that represents the values that \( v \) can assume.

The statement \texttt{var \( n \) : \( N \); \( c \) : CUSTOMER} declares the variable \( n \) that can store data of type \( N \) and the variable \( c \) that can store data of type CUSTOMER.

**Input and output events.** A prefixed action is defined to achieve input or output communications whereby the communication event defined in the prefix takes place before the action starts. A prefixed action is defined using the format \texttt{Comm \( \rightarrow \) A} as found in the BNF in Figure A.2 where \texttt{Comm} is a communication event and \texttt{A} is a Circus action.

**Inputting a value**

A prefixed action for input is written \texttt{c?\( x \) \( \rightarrow \) A}, whereby the system accepts a value to the variable \( x \) through the channel \( c \) and then behaves like \( A \) and \( x \) is in the scope of \( A \) (Oliveira, 2005).
Outputting a value

A prefixed action for output is written $c!y \rightarrow A$ whereby the system outputs the value of the expression $y$ through the channel $c$ and then behaves like $A$.

Assigning a value to a variable. A value can be assigned to a variable using the action statement $a := E$ where $a$ is a variable defined in the scope of the assignment action and $E$ is an expression that evaluates to a value of the same type as the state variable $a$.

A.1.4 Defining the overall behaviour of the system

In the Circus notation, a system may be defined with one or more processes. Each process may contain one or more actions and a main action. However, all the systems analysed in this thesis are based on systems composed of multiple actions within a single process. This is because our work uses BCF in Circus ([Banks, 2012](#)) where the verification laws developed so far have been restricted to single processes.

A Circus specification supports multiple actions through action composition using the CSP operators in the Circus BNF in Figure A.2. The following constructs are utilized in this thesis for process composition.

Recursion. The recursion construct $\mu X \cdot A; X$ recursively executes the action $A$ where $X$ is a recursion label defined using the construct $\mu X$. We may define multiple recursions in a single main action.

Sequential composition. The construct $A; B$ defines a sequential composition of two actions whereby ‘;’ is the sequential composition operator and the execution of the action $A$ is immediately followed by the execution of the action $B$.

In the example the statement $(RecordMyReceipt \diamond GetMySpent); X$ defines a sequential composition where the composite action $(RecordMyReceipt \diamond GetMySpent)$ is immediately followed by $X$, a recursive invocation of the process starting at the label $X$. 
Figure A.1: Circus BNF as published in Freitas (2005) doctoral thesis


External choice. The logged in user is allowed to either increment his expenditure total by recording a new receipt in the system or view his/her total expenditure recorded in the system.

RecordMyReceipt ⊕ GetMySpent

These are multiple execution paths which the user can choose from. The external
A.1 Modelling a system using the Circus notation

choice operator $\square$ can be used to define a composite action with multiple execution paths where the selected path depends on the external input.

**Main action.** The overall behaviour of a Circus process is defined by defining a nameless action within the process scope. The overall behaviour of the system in the running example is defined using the following Circus action.

$$\mu X \bullet \left( \begin{array}{c}
RecordMyReceipt \\
\square GetMySpent
\end{array} \right) \; X$$

An implementation of the system specification in Figure A.1 will behave as follows. In the beginning state of the system, it is assumed that the state variables price and spent have already been populated with the necessary data. While in this state, the Circus action RecordMyReceipt or GetMySpent can be chosen by the environment. At the end of the run, the recursive label $X$ locates the program counter to the state at the beginning of the program where the label $X$ is defined.

A.1.5 Recursion

The recursion construct in the Circus notation is of the form $\mu X \bullet F(X)$. To left justify the lifted body $\langle F(X) \rangle$ with respect to the confidentiality annotation $\theta$, we have to identify an invariant obligation $\theta_i$ where $\theta \sqsubseteq \theta_i$ (Banks, 2012, p. 148). Consider the original confidentiality annotation $\theta$ as $\theta_0$. The following calculation is used to identify a $\theta_n$ such that $\theta_n = \theta_{n+1}$.

$$\theta_{i+1} = \theta_i \land \text{bw}(\langle F(X) \rangle, \theta_i)$$

Banks’s argues that this method is not guaranteed to identify a finite $n$ such that $\theta_n = \theta_{n+1}$ (Banks, 2012, p. 148) in the general case. He further states that, in this case one may resort to intuition by analysing patterns of obligations to derive an invariant obligation.
A Appendix

The calculation of the invariant obligation needs further research to derive any useful laws that can be used for back propagation calculations and as such this endeavour is not within the scope of this thesis. For this reason, we do not calculate invariant obligations when back propagating the systems in the case studies in this thesis. I note that omitting the calculation of the invariant obligation is a limitation of the case study analysis done in this thesis.

A.2 A comparison of the tools that provide any form of support for specifying systems in the Circus notation

Tool support for the Circus notation is limited. To the author’s knowledge, the only tools that provide any form of support for specifying systems in the Circus notation are Symphony IDE supporting COMPASS Modelling Language (CML), CZT and CRefine. The following describes the extent to which each tool supports specifying systems in the Circus notation and the effort required to modify the tool to support an extension to the Circus notation.

Symphony IDE supporting COMPASS Modelling Language (CML). Symphony IDE supports the domain specific language CML that is designed for modelling and analysing systems of systems (Woodcock and Miyazawa, 2012). CML is based on VDM (Gulati and Singh, 2012), CSP (Hoare, 1980) and Circus (Oliveira et al., 2006). Symphony IDE (Coleman et al., 2014) is a tool that utilizes CML models to generate Isabelle theorem files to reason about certain properties of those models. Hence, CML, Symphony IDE and the Isabelle theorem prover provide a clear path from formal models based on Circus to Isabelle theorem prover, where we can reason about various properties of those models. However, the generated theorem files are based on the Isabelle/UTP framework (Woodcock et al., 2015). The foundation of the Isabelle/UTP framework is based on a custom definition for the UTP variable and a value model that complements the typing requirements for those variables. In order to extend the Isabelle/UTP to support the extended twin semantics of BCF we need to code a new definition for every
A.3 Decisions regarding the development of a custom tool for BCF application

UTP notation and function of the theory of designs in the Isabelle/UTP. Therefore, CML is not a viable alternative for the purpose of extending it to support BCF.

Symphony IDE generates Isabelle theorem files based on CML system models to reason about certain properties of these models. The theorem files are based on a theory package that needs extensive extension from the ground up if twin semantics was to be supported.

Community Z Tools (CZT). CZT (Malik and Utting, 2005) supports the parsing of Circus specifications. However, the architecture and inner workings of the CZT editor is very complex that demand a steep learning curve before the CZT editor could be extended to support reasoning about confidentiality requirements in a given Circus specification. Kimber (2007) considered the CZT editor as an input interface which he planned to extend for code generation to produce PerfectDeveloper (Crocker, 2003) code from Object-Z specifications. However, after reviewing the CZT editor, Kimber (2007) concluded that the DTDs and schemas of CZT projects were quite impenetrable.

CRefine. CRefine (Oliveira et al., 2008) is a tool that supports the use of Circus refinement calculus (Sampaio et al., 2003). It has an inbuilt proof obligation manager, that automatically dispatch proofs for some of its refinement steps (Oliveira et al., 2008). However, since CRefine extends CZT (Gurgel et al., 2008), it suffers from the same architectural complexities.

A.3 Decisions regarding the development of a custom tool for BCF application

For design decisions regarding the development of a BCF application and code generation tool, it is best to learn how the past researchers have approached it. CRefine, Symphony IDE and Perfect Developer (Crocker, 2003) are some of the tools that have been developed in the past for the formal specification and verification of systems. Perfect Developer (Crocker, 2003) is a tool that allows a user to define and later refine a formal specification to object oriented code. In addition, it generates “proof obligations”
A Appendix

d from pre-conditions, invariants, etc to verify the correctness of the system model being defined (Kimber, 2007). These three tools allow the formal specification systems and dispatch proofs about certain properties of those systems. CRefine and Symphony IDE both support Circus or a close variant of it. Our requirements for a formal tool for reasoning about confidentiality, also share these characteristics.

Even-though CRefine accepts a LaTeX document, it has support for a unicode (on-screen pretty printing) display format because the author’s of the CRefine tool believe that its target audience is mostly not familiar with LaTeX and also that “pretty printing is a success among researchers, since it unconditionally makes the presentation of the development more user friendly” (Oliveira et al., 2008). Perfect developer adopts a non-LaTeX notation similar to any object oriented language, to make it more “accessible for software developers with limited mathematical knowledge”. CML is also a non-LaTeX notation.
A.4 Translating CFAT notation to HOL

The following subsections describe how the some CFAT structures that represent Z data structures are translated into equivalent HOL data structures in the Isabelle theorem prover. The following presents a discussion of how the the data structures of CFAT notation is translated to similar HOL data structures.

**Variable definition.** Z is a strongly typed language and hence a variable declaration in Z implies that the universal set that contains all elements for that type (Kolyang et al., 1996) restricts the possible values of that variable. This implication is the type invariant for that type. For, e.g., The Z statement `user : EMPLOYEE;` defines a state variable “user” with an arbitrary type EMPLOYEE (type definition), but it also

- implies that “user” belongs to an arbitrary set EMPLOYEE of the same type (type invariant).

However, in HOL notation, the type definition and the type invariant must be stated separately. To standardize the translation from Z to HOL and also for ease of readability, the convention we have adopted is that we write;

- the lowercase of the given Z type name EMPLOYEE, as the equivalent arbitrary type name in HOL (type EMPLOYEE in Z written as type employee in HOL)

- the given type EMPLOYEE (also the assumed arbitrary set) in Z as name of the universal set representing the given type in HOL (arbitrary set EMPLOYEE is Z written as arbitrary set EMPLOYEE in HOL)

For example, Equation (A.1) defines a variable in the Z notation where the variable name is `user` and its type is EMPLOYEE.

\[
user : EMPLOYEE \quad \text{(A.1)}
\]
The variable definition in Equation (A.1) can be written in equivalent HOL term as in Equation (A.2).

\[ \forall (\text{user} :: \text{employee}). \text{user} \in \text{EMPLOYEE} \]  

where \text{employee} is an arbitrary type, \text{user} is universally quantified and \text{EMPLOYEE} is a set of the same type \text{employee}.

Each basic type defined in the Z notation is considered to be synonymous with a universal set of labels that represent that type. Hence, we translate every basic type defined in the Z notation as the Isabelle theorem prover type \text{string}, that represent a set of strings. In the Isabelle theorem prover, we defined this type synonym with the statement;

\text{type synonym employee = string}

Subsequently, we define the arbitrary sets of basic types as sets of strings as shown below.

\[
\text{definition EMPLOYEE :: "string set" where "EMPLOYEE == } v. \exists (s :: \text{string}). \text{v = s"}
\]

Relations and functions. A relation is a set of cartesian products of some given types. Throughout this thesis, we restrict ourselves to binary relations which are relations between two given types. In Z notation, a relation is enforced between two types via the use of the infix operator \(\leftrightarrow\). Given two types \text{DOCTOR} and \text{PATIENT}, Equation (A.3) defines a relation between the two.

\[ \text{GP : DOCTOR } \leftrightarrow \text{ PATIENT} \]  

Equation (A.3) can be read as: \text{the relation GP relates doctors to patients}. The HOL notation equivalent for the Equation (A.3) is shown as Equation (A.4).

\[ \forall (\text{GP} :: \text{doctor } \leftrightarrow \text{ patient}). \text{GP } \in \text{(DOCTOR } \leftrightarrow \text{ PATIENT)} \]  

292
Here, *doctor* and *patient* are two type variables representing types of the elements in the two universal sets *DOCTOR* and *PATIENT* defined by the two given types and `<=>` is the type constructor for the “relation type” in the Mathematical tool-kit. The invariant \( GP \in (DOCTOR \ <=> PATIENT) \) defines the valid elements that belong to the set of the cartesian products of the relation \( GP \). The set of elements *DOCTOR* and *PATIENT* of the arbitrary types *doctor* and *patient* are defined as follows.

\[
\text{definition DOCTOR} :: \text{"string set "} \\
\text{where } \text{"DOCTOR} \equiv v. \exists (s::\text{string}) . \ v = s\]

Set definition for basic type DOCTOR

\[
\text{definition PATIENT} :: \text{"string set "} \\
\text{where } \text{"PATIENT} \equiv v. \exists (s::\text{string}) . \ v = s\]

Set definition for basic type PATIENT

Functions are special types of relations. The toolkit by [Bowen and Gordon 1994](#1994) defines functions that restrict the subset of relations that are healthy with respect to various function definitions. Appendix A.4 presents operators in the toolkit that are defined for functions over relations and components of those relations.
### A Appendix

<table>
<thead>
<tr>
<th>Function type</th>
<th>Z infix notation</th>
<th>HiVe notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary relation</td>
<td>↔</td>
<td>&lt;-</td>
</tr>
<tr>
<td>Partial function</td>
<td>↦→</td>
<td>-</td>
</tr>
<tr>
<td>Total function</td>
<td>→</td>
<td>-</td>
</tr>
<tr>
<td>Function application</td>
<td>%</td>
<td>%^</td>
</tr>
<tr>
<td>Cartesian product</td>
<td>×</td>
<td>&gt;&gt;</td>
</tr>
<tr>
<td>Maplet</td>
<td>⇐</td>
<td></td>
</tr>
</tbody>
</table>

Table A.3: Operators in HiVe for the Z mathematical constructs

Following are the function definitions for partial and total functions.

\[
X \rightarrow Y = \{ f : (S \leftrightarrow R) \land (\forall x y1 y2. (x, y1) : f \land (x, y2) : f \Rightarrow (y1 = y2)) \} \\
X \rightarrow Y = \{ s : S \rightarrow R \land \text{dom } s = S \}
\]

In a system where only some doctors are assigned patients, a partial relation exists between the doctors and the patients. In such a case, the relation \(GP\) can be defined as;

\[
GP : \text{DOCTOR} \rightarrow \text{PATIENT}
\]

whereas the HOL equivalent definition is,

\[
\forall (GP : \text{doctor} \leftrightarrow \text{patient}). GP \in (\text{DOCTOR} -|-> \text{PATIENT})
\]
A.5 Description and formal specification of systems

A.5.1 Case study - Phone book system

A hand-crafted documentation with a formal specification of the Phone book system of a secret government agency is presented. A *Circus* action is defined for each use case in the use case diagram of the Phone book system shown in Figure A.3. The operations performed by these actions are described in Table A.5. It is important to note that the action *Init* in Table A.5 is not in Figure A.3. This is because *Init* initializes system variables and is not executed through external action. The following table of contents may be useful for easy navigation within this case study.

<table>
<thead>
<tr>
<th>Contents of the case study</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation of the Phone book system case study</td>
<td>295</td>
</tr>
<tr>
<td>- Operations</td>
<td>295</td>
</tr>
<tr>
<td>- Data types of system entities</td>
<td>297</td>
</tr>
<tr>
<td>- State variables</td>
<td>297</td>
</tr>
<tr>
<td>- Formal specification of the system in the <em>Circus</em> notation</td>
<td>300</td>
</tr>
</tbody>
</table>

A.5.1.a Operations

The operations of the system are formalized as *Circus* actions and described in Table A.5.
Appendix

![Use case diagram for the Phone book system](image)

**Figure A.3:** Use case diagram for the Phone book system

**Table A.5:** Phone book system - Description of the *Circus* actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Operation performed by the action</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>AddPhoneNumber</em></td>
<td>Adds a phone number to the phone book.</td>
</tr>
<tr>
<td><em>SetSecretPhoneNumber</em></td>
<td>Adds an official into the list of high ranking officials whose phone numbers should never be revealed.</td>
</tr>
<tr>
<td><em>GetPhoneNumberNoSecret</em></td>
<td>Outputs the phone number of a given official, as long he is not included in the list of high ranking officials.</td>
</tr>
<tr>
<td><em>GetPhoneNumberAny</em></td>
<td>Outputs the phone number of a given official.</td>
</tr>
</tbody>
</table>
A.5.1.b Data types of system entities

The entities of the system are employees and their phone numbers. These are represented using basic types. These basic types are described in Table A.6.

Table A.6: Phone book system - Description of the basic types

<table>
<thead>
<tr>
<th>Basic type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE</td>
<td>set of all possible identifiers for employees</td>
</tr>
<tr>
<td>TELEPHONE</td>
<td>set of all possible telephone numbers</td>
</tr>
</tbody>
</table>

A.5.1.c State variables

The state of the system is recorded by the schema State. The state variables that record and maintain various information about the system state are described in Table A.7.

Table A.7: Phone book system - Description of the state variables

<table>
<thead>
<tr>
<th>State variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phoneNumbers</td>
<td>A relation that identifies the telephone number allocated to each unique employee.</td>
</tr>
<tr>
<td>secretList</td>
<td>The set of unique identifiers that represents the set of employees whose phone numbers are considered secret.</td>
</tr>
<tr>
<td>loggedIn</td>
<td>The set of identifiers of the users who are logged into the system.</td>
</tr>
</tbody>
</table>
A Appendix

The identifier that represents the user who is currently logged into the system.

The identifier of the official whose phone number is being requested from the system.

The set of identifiers that represents all the officials whose phone numbers have been recorded in the system.

The set of unique identifiers that represents the secretaries of the organisation.

The set of identifiers that represents all the engineers whose phone numbers have been recorded in the system.

The identifier that represents the manager of the organisation.

A.5.1.d State invariants

Further, the system is designed to respect the following constraints.

<table>
<thead>
<tr>
<th>State invariants Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dom ( (phoneNumbers) \subseteq officials )</td>
</tr>
</tbody>
</table>

The set of people whose phone numbers are recorded in the system must be from the set of officials recorded in the system.

<table>
<thead>
<tr>
<th>State invariants Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secretList \subseteq dom ( (phoneNumbers) )</td>
</tr>
</tbody>
</table>
The set of officials whose numbers are considered confidential must be from the set of officials whose phone numbers have been recorded in the system.

\[ req_{\text{Official}} \in \text{dom}(\text{phoneNumbers}) \]

The official whose phone number is being requested from the system must have his/her phone number recorded in the system.

\[ login_{\text{User}} \in \text{loggedIn} \]

The current user must be from the set of users logged into the system.

\[ \text{loggedIn} \subseteq \text{officals} \]

The set of users who are logged into the system must be from the set of officials in the agency.

\[ \text{officals} \subseteq (\text{secretaries} \cup \text{engineers} \cup \{\text{manager}\}) \]

The set of officials of the agency must be from the group of secretaries, the group of engineers or the manager.

\[ (\text{engineers} \cap \{\text{manager}\}) = \{\} \]

The same person cannot be an engineer and the manager at the same time.

\[ (\{\text{manager}\} \cap \text{secretaries}) = \{\} \]

The same person cannot be the manager and a secretary at the same time.

\[ (\text{engineers} \cap \text{secretaries}) = \{\} \]

The same person cannot be an engineer and a secretary at the same time.
State
phoneNumbers : EMPLOYEE \rightarrow TELEPHONE
manager, reqOfficial, loginUser : EMPLOYEE
engineers, secretaries, officials, loggedIn, secretList : \mathbb{F} \text{ EMPLOYEE}

\[
\begin{align*}
(\text{engineers} \cap \text{secretaries}) &= \emptyset \\
(\{\text{manager}\} \cap \text{secretaries}) &= \emptyset \\
(\text{engineers} \cap \{\text{manager}\}) &= \emptyset \\
\text{officials} &\subseteq (\text{secretaries} \cup \text{engineers} \cup \{\text{manager}\}) \\
\text{loggedIn} &\subseteq \text{officials} \\
\text{loggedIn} &\subseteq \text{officials} \\
\text{reqOfficial} &\in \text{dom} (\text{phoneNumbers}) \\
\text{secretList} &\subseteq \text{dom} (\text{phoneNumbers}) \\
\text{dom} (\text{phoneNumbers}) &\subseteq \text{officials}
\end{align*}
\]

HideSecretNumber
\begin{align*}
\exists \text{State}_9 \cdot \\
\text{reqOfficial} &\in \text{secretList} \land \\
\text{reqOfficial} &\neq \text{loginUser} \land \\
\text{loginUser} &\neq \text{manager} \implies \\
\text{reqOfficial} &\in \text{dom} (\text{phoneNumbers}) \implies \\
\text{reqOfficial} &\notin \text{dom} (\text{phoneNumbers}_9)
\end{align*}

channel \ addPhoneNumberIn, anyPhoneOut, noSecPhoneOut : TELEPHONE
channel \ noSecOfficialIn, setSecretOfficialIn, addPhoneOfficialIn : EMPLOYEE
channel \ anyOfficialIn : EMPLOYEE

channelset \ Engineers \ := \ { \text{noSecPhoneOut, noSecOfficialIn} } \\
channelset \ Manager \ := \ { \text{setSecretOfficialIn, anyPhoneOut, anyOfficialIn} } \\
channelset \ Secretaries \ := \ { \text{addPhoneOfficialIn, noSecPhoneOut, noSecOfficialIn, addPhoneNumberIn} }

process \ Phone\_book\_system \ = \ \mathsf{begin} \\
\text{state} \ State \\
\text{Init} \ = \ reqOfficial := \text{loginUser} \\
\text{AddPhoneNumber} \ = \ \mathsf{var} \ addPhoneNumber : \text{TELEPHONE}; \\
\text{addPhoneOfficial} : \text{officials} \bullet \\
\text{addPhoneNumberIn?addPhoneNumber} \rightarrow \\
\text{addPhoneOfficialIn?addPhoneOfficial} \rightarrow \\
((\text{loggedIn} \in \text{secretaries}) \land \\
\text{phoneNumbers} := \text{phoneNumbers} \oplus \\
\{(\text{addPhoneOfficial}? \rightarrow \text{addPhoneNumber}?)\}) \\
\text{SetSecretPhoneNumber} \ = \ \mathsf{var} \ setSecretOfficial : \text{dom} (\text{phoneNumbers}) \bullet \\
\text{setSecretOfficialIn?setSecretOfficial} \rightarrow \\
((\text{loggedIn} = \text{manager}) \land \\
\text{secretList} := \text{secretList} \cup \{\text{setSecretOfficial}\})
\end{align*}

Figure A.4: Specification of Phone book system - code block 1 of 2
GetPhoneNumberNoSecret \equiv \textbf{var} \ noSecOfficial : \text{dom} \ (\text{phoneNumbers}) \ \bullet \\
\ noSecOfficialIn?\noSecOfficial \rightarrow \\
\ reqOfficial := \noSecOfficial?; \\
\ ((\noSecOfficial? \notin \text{secretList} \lor \\
\ loginUser = \text{manager}) \& \\
\ noSecPhoneOut !(\text{phoneNumbers} \noSecOfficial?) \rightarrow \text{Skip}) \\

GetPhoneNumberAny \equiv \textbf{var} \ anyOfficial : \text{dom} \ (\text{phoneNumbers}) \ \bullet \\
\ anyOfficialIn?\anyOfficial \rightarrow \\
\ reqOfficial := \anyOfficial?; \\
\ ((\loginUser = \text{manager}) \& \\
\ anyPhoneOut !(\text{phoneNumbers} \anyOfficial?) \rightarrow \text{Skip}) \\

HideSecretNumber \equiv \text{HideSecretNumber} \\

\text{Options} \equiv \left( \begin{array}{c}
\text{AddPhoneNumber} \\
\text{SetSecretPhoneNumber} \\
\text{GetPhoneNumberNoSecret} \\
\text{GetPhoneNumberAny}
\end{array} \right) ; \text{HideSecretNumber} \\

\bullet \ (\text{Init}) ; \mu Y \bullet \ (\text{Options} ; Y) \\

end
A Appendix

A.5.2 Case study - Secure electronic examination system

A hand-crafted documentation with a formal specification of the Secure electronic examination system is presented. A Circus action is defined for each use case in the use case diagram of the Secure electronic examination system shown in Figure A.5. The operations performed by these actions are described in Table A.14. It is important to note that the action Init in Table A.14 is not in Figure A.5. This is because Init initializes system variables and is not executed through external action. The following table of contents may be useful for easy navigation within this case study.

<table>
<thead>
<tr>
<th>Contents of the case study</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation of the Secure electronic examination system case study</td>
<td>310</td>
</tr>
<tr>
<td>- Operations</td>
<td>310</td>
</tr>
<tr>
<td>- Data types of system entities</td>
<td>302</td>
</tr>
<tr>
<td>- State variables</td>
<td>305</td>
</tr>
<tr>
<td>- Formal specification of the system in the Circus notation</td>
<td>312</td>
</tr>
</tbody>
</table>

A.5.2.a Data types of system entities

The entities of the system including the actors Chair, Student, Setter, Checker and Grader as well as the components Paper, Question, Answer and Candidate are represented using basic types. These basic types are described in Table A.10.
A.5 Description and formal specification of systems

Figure A.5: Use case diagram for a Secure electronic examination system
Table A.10: Secure electronic examination system - Description of the basic types

<table>
<thead>
<tr>
<th>Basic type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMINER</td>
<td>The set of all possible identifiers for examiners</td>
</tr>
<tr>
<td>USER</td>
<td>The set of all users who are allowed access to the system</td>
</tr>
<tr>
<td>SUBJECT</td>
<td>The set of all identifiers for subjects</td>
</tr>
<tr>
<td>PAPER</td>
<td>The set of all identifiers for all subject examination papers announced and managed in the system</td>
</tr>
<tr>
<td>QUESTION</td>
<td>The set of all identifiers for questions in all the subject examination papers</td>
</tr>
<tr>
<td>ANSWER</td>
<td>The set of all identifiers for answers recorded by students against the subject examination papers</td>
</tr>
<tr>
<td>CANDIDATE</td>
<td>The set of all identifiers representing candidates who are students who have registered in the system to sit on examinations</td>
</tr>
</tbody>
</table>

The login status of users and the status of the examination papers are attributes that define values for certain properties of some entities in the system. These attributes are declared as free types in the system and are described in Table A.11.

Table A.11: Secure electronic examination system - Description of the free types

<table>
<thead>
<tr>
<th>Free type</th>
<th>Item</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPERSTATUS</td>
<td>announced</td>
<td>The paper has been announced.</td>
</tr>
</tbody>
</table>
A.5 Description and formal specification of systems

setting  The paper is currently being set.

checked  The paper has been checked.

released  The paper has been released for candidates to take the paper.

paperclosed  The examination has ended and answers will not be accepted from candidates for that paper.

grading  The paper is currently being graded.

gradepublished  The results have been published.

---

A.5.2.b State variables

The state of the system is recorded by the schema State. The state variables that record and maintain various information about the system state are described in Table A.12.

Table A.12: Secure electronic examination system - Description of the state variables

<table>
<thead>
<tr>
<th>State variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chair</td>
<td>A function that identifies the lecturer appointed as the chair for a subject, if any.</td>
</tr>
<tr>
<td>exams</td>
<td>A relation between subjects and examination papers initiated for those subjects.</td>
</tr>
<tr>
<td>setter</td>
<td>A function that identifies the lecturer appointed as the setter for a paper, if any.</td>
</tr>
</tbody>
</table>
A Appendix

checker
A function that identifies the lecturer appointed as the checker for a paper, if any.

grader
A function that identifies the lecturer appointed as the marker for a paper, if any.

pStatus
A function that identifies the status of a paper, if any.

regStudent
A function that identifies the student for a given candidate, if any.

regPaper
A function that identifies the paper registered by a candidate, if any.

questions
A relation between papers and questions that belong to those papers.

ansPaper
A relation between papers and answers recorded on those papers.

ansStudent
A relation between candidate and answers they have recorded.

result
A function that represents the grade obtained by a candidate, if any.

students
The set of identifiers of the students.

lecturers
The set of identifiers of the lecturers.

users
The set of identifiers of the users who are allowed access to the system.

loginUser
The user who is currently logged into the system.

theCandidate
The candidate whose grade or answer is being requested from the system.
A.5 Description and formal specification of systems

A.5.2.c State invariants

Further, the system is designed to respect the following constraints.

Table A.13: Secure electronic examination system - Description of the state invariants

<table>
<thead>
<tr>
<th>State invariants Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>((\text{ran\ chair}) \cup (\text{ran\ setter}) \cup (\text{ran\ checker}) \cup (\text{ran\ grader})) \subseteq \text{lecturers} )</td>
<td>The set of lecturers who are appointed as chairs of subjects a subset of the set of chairs recorded in the system.</td>
</tr>
<tr>
<td>((\text{ran\ chair}) \cap (\text{ran\ setter})) = {} )</td>
<td>A lecturer cannot be a subject chair and a setter at the same time.</td>
</tr>
<tr>
<td>((\text{ran\ chair}) \cap (\text{ran\ checker})) = {} )</td>
<td>A person cannot be both a chair and a checker in the system.</td>
</tr>
<tr>
<td>((\text{ran\ chair}) \cap (\text{ran\ grader})) = {} )</td>
<td>A person cannot be both a chair and a grader in the system.</td>
</tr>
<tr>
<td>((\text{ran\ setter}) \cap (\text{ran\ checker})) = {} )</td>
<td>A person cannot be both a setter and a checker in the system.</td>
</tr>
<tr>
<td>((\text{ran\ setter}) \cap (\text{ran\ grader})) = {} )</td>
<td>A person cannot be both a setter and a grader in the system.</td>
</tr>
<tr>
<td>((\text{ran\ checker}) \cap (\text{ran\ grader})) = {} )</td>
<td>A person cannot be both a checker and a grader in the system.</td>
</tr>
</tbody>
</table>
Appendix

A person cannot be both a checker and a grader in the system.

\[ \text{users} \subseteq (\text{lecturers} \cup \text{students}) \]

The set of users allowed access to the system is a subset of the set of lecturers and students.

\[ (\text{lecturers} \cap \text{students}) = \{\} \]

A person cannot be a lecturer and a student at the same time.

\[ \text{loginUser} \in \text{users} \]

The user currently using the system must be from the set of users allowed access to the system.

\[ \text{dom}(\text{pStatus}) \subseteq \text{dom}(\text{exams}) \]

A paper for which a paper status has been recorded must belong to a particular subject.

\[ \text{dom}(\text{setter}) \subseteq \text{dom}(\text{pStatus}) \]

A paper for which a setter has been assigned must have a paper status.

\[ \text{dom}(\text{checker}) \subseteq \text{dom}(\text{pStatus}) \]

A paper for which a checker has been assigned must have a paper status.

\[ \text{dom}(\text{grader}) \subseteq \text{dom}(\text{pStatus}) \]

A paper for which a grader has been assigned must have a paper status.

\[ \text{dom}(\text{questions}) \subseteq \text{dom}(\text{pStatus}) \]

Every paper on which a question is recorded must have a paper status.
A.5 Description and formal specification of systems

\[ \text{dom} (\text{ansPaper}) \subseteq \text{dom} (\text{pStatus}) \]

Every paper on which an answer is recorded must have a paper status.

\[ \text{ran regPaper} \subseteq \text{dom} (\text{pStatus}) \]

Every paper registered for must have a paper status.

\[ \text{dom} (\text{regPaper}) \subseteq \text{dom} (\text{regStudent}) \]

Every student registered for a paper must be a registered candidate.

\[ \text{ran regStudent} \subseteq \text{students} \]

A student registered as a candidate must be a student recorded in the system.

\[ \text{dom} (\text{result}) \subseteq \text{dom} (\text{regPaper}) \]

Every result recorded in the system must be for a registered paper.

\[ \text{dom} (\text{ansStudent}) \subseteq \text{dom} (\text{regPaper}) \]

Every answer recorded by a student must be for a registered paper.

\[ \text{dom} (\text{questions}) \subseteq \text{dom} (\text{setter}) \]

Every paper for which questions are recorded must have a setter assigned to it.

\[ \text{dom} (\text{questions}) \subseteq \text{ran regPaper} \]

Every question recorded in the system must be for a registered paper.

\[ \text{dom} (\text{ansPaper}) \subseteq \text{ran regPaper} \]

Every answer recorded in the system must be for a registered paper.
A Appendix

A.5.2.d Operations

The operations of the system are formalized as Circus actions and described in Table A.14.

Table A.14: Secure electronic examination system - Description of the Circus actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Operation performed by the action</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnnouncePaper</td>
<td>Allows a subject chair to announce a subject examination.</td>
</tr>
<tr>
<td>AppointSetter</td>
<td>Allows a subject chair to appoint a subject lecturer to set the exam paper.</td>
</tr>
<tr>
<td>ResignSetter</td>
<td>Allows an appointed setter to resign himself or herself from the post.</td>
</tr>
<tr>
<td>AppointChecker</td>
<td>Allows a subject chair to appoint a subject lecturer to check the exam paper.</td>
</tr>
<tr>
<td>ResignChecker</td>
<td>Allows an appointed checker to resign himself or herself from the post.</td>
</tr>
<tr>
<td>AppointMarker</td>
<td>Allows a subject chair to appoint a subject lecturer to set the exam paper.</td>
</tr>
<tr>
<td>ResignMarker</td>
<td>Allows an appointed grader to resign himself or herself from the post.</td>
</tr>
<tr>
<td>CloseExam</td>
<td>Allows a setter to close the paper to conclude the end of the examination.</td>
</tr>
</tbody>
</table>
### A.5 Description and formal specification of systems

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckPaper</td>
<td>Allows a checker to check the paper and record his approval by changing the paper status to checked.</td>
</tr>
<tr>
<td>GradePaper</td>
<td>Allows a grader to record the grade awarded to a particular candidate for a particular paper.</td>
</tr>
<tr>
<td>PublishGrades</td>
<td>Allows a grader to publish the grades obtained by all candidates who set for a paper.</td>
</tr>
<tr>
<td>TakePaper</td>
<td>Allows a student to record answers for a particular paper during the examination.</td>
</tr>
<tr>
<td>GetMyAnswers</td>
<td>Allows a user to view the answers he/she has recorded in the system.</td>
</tr>
<tr>
<td>RegisterforExam</td>
<td>Allows a student to register for a particular paper using his/her candidate number.</td>
</tr>
<tr>
<td>CancelRegistration</td>
<td>Allows a student to cancel his/her registration for a particular paper.</td>
</tr>
<tr>
<td>GetOwnResult</td>
<td>Allows a student to view his/her grade for a particular paper he/she took.</td>
</tr>
<tr>
<td>GetAnyResult</td>
<td>Allows a lecturer to view the grade of any candidate for any paper he/she has sat.</td>
</tr>
<tr>
<td>SetPaper</td>
<td>Allows a setter to set a paper by recording questions for that paper.</td>
</tr>
<tr>
<td>ReleasePaper</td>
<td>Allows a setter to release the paper for candidates to take the examination.</td>
</tr>
</tbody>
</table>
Figure A.6: Specification of Secure electronic examination system - code block 1 of 6
channel gradeAwardedIn, anyResultOut, ownResultOut : N
channel takePaperAnswerIn : ANSWER
channel appCheckerSubjectIn, appointPsubjectIn, paperSubjectIn : SUBJECT
channel appMarkerSubjectIn : SUBJECT
channel resignPsetterIn, appointPSetterIn, newPaperIn : PAPER
channel gradePaperIn, chkPaperIn, closePaperIn, resMarkerPaperIn : PAPER
channel regExamPaperIn, takePaperIn, pubGradePaperIn : PAPER
channel anyResultPaperIn, ownResultPaperIn, cancelRegPaperIn : PAPER
channel releasePaperIn, setPaperIn : PAPER
channel ownAnswerCandidateIn, takePaperCandidateIn, gradeCandidateIn : CANDIDATE
channel ownResultCandidateIn, cancelRegCandidateIn, regExamCandidateIn : CANDIDATE
channel anyResultCandidateIn : CANDIDATE
channel appMarkerLecturerIn, appCheckerLecturerIn, appointPlecturerIn : USER
channel ownAnswerOut : F ANSWER
channel setQuestionPaperIn : QUESTION

channelset Chair == { appMarkerSubjectIn, appMarkerLecturerIn, resMarkerPaperIn, appCheckerLecturerIn, resCheckerPaperIn, appMarkerPaperIn, resignPsetterIn, appCheckerSubjectIn, appCheckerPaperIn, appointPSetterIn, appointPsubjectIn, appointPlecturerIn }
channelset Checker == { resCheckerPaperIn, chkPaperIn }
channelset Marker == { gradeCandidateIn, gradeAwardedIn, pubGradePaperIn, gradePaperIn }
channelset Setter == { resignPsetterIn, releasePaperIn, closePaperIn, setPaperIn, setQuestionPaperIn }
channelset Student == { takePaperCandidateIn, ownAnswerCandidateIn, ownAnswerOut, ownResultOut, takePaperIn, takePaperAnswerIn, cancelRegCandidateIn, ownResultPaperIn, ownResultCandidateIn, regExamPaperIn, regExamCandidateIn, cancelRegPaperIn }

process Secure_electronic_examination_system ≜ begin

state State

Init ≜ theCandidate := ((regStudent ~) loginUser

AnnouncePaper ≜ var paperSubject : dom (chair);
newPaper : PAPER •
paperSubjectIn?paperSubject →
newPaperIn?newPaper →
((chair paperSubject? = loginUser) &
exams := exams ⊕ \{(newPaper? → paperSubject?)\};
pStatus := pStatus ⊕ \{(newPaper? → announced)\})

AppointSetter ≜ var appointPlecturer : lecturers;
appointPsubject : dom (chair);
appointPSetter : dom (pStatus) •
appointPlectorIn?appointPlecturer →
appointPsubjectIn?appointPsubject →
appointPSetterIn?appointPSetter →
((loginUser ∈ lecturers ∧ (appointPsubject?, loginUser) ∈ chair) &
setter := setter ⊕ \{(appointPSetter? → appointPlecturer?)\})

Figure A.6 (cont.) : Specification of Secure electronic examination system - code block 2 of 6
ResignSetter ≜ var resignPsetter : dom(setter) •
resignPsetterIn?resignPsetter−→
((loginUser ∈ lecturers ∧ (resignPsetter?, loginUser) ∈ setter) &
setter := setter \ { (resignPsetter? → loginUser)})

AppointChecker ≜ var appCheckerLecturer : lecturers;
appCheckerSubject : dom (chair);
appCheckerPaper : dom (pStatus) •
appCheckerLecturerIn?appCheckerLecturer−→
appCheckerSubjectIn?appCheckerSubject−→
appCheckerPaperIn?appCheckerPaper−→
((loginUser ∈ lecturers ∧
(appCheckerSubject?, loginUser) ∈ chair) &
checker := checker \ {(appCheckerPaper? → appCheckerLecturer?)})

ResignChecker ≜ var resCheckerPaper : dom (checker) •
resCheckerPaperIn?resCheckerPaper−→
((loginUser ∈ lecturers ∧
(resCheckerPaper?, loginUser) ∈ checker) &
checker := checker \ {(resCheckerPaper? → loginUser)})

AppointMarker ≜ var appMarkerLecturer : lecturers;
appMarkerSubject : dom (chair);
appMarkerPaper : dom (pStatus) •
appMarkerLecturerIn?appMarkerLecturer−→
appMarkerSubjectIn?appMarkerSubject−→
appMarkerPaperIn?appMarkerPaper−→
((loginUser ∈ lecturers ∧
(appMarkerSubject?, loginUser) ∈ chair) &
grader := grader \ {(appMarkerPaper? → appMarkerLecturer?)})

ResignMarker ≜ var resMarkerPaper : dom (grader) •
resMarkerPaperIn?resMarkerPaper−→
((loginUser ∈ lecturers ∧ (resMarkerPaper?, loginUser) ∈ grader) &
grader := grader \ {(resMarkerPaper? → loginUser)})

CloseExam ≜ var closePaper : dom (setter) ∩ dom (pStatus) •
closePaperIn?closePaper−→
((loginUser ∈ lecturers ∧
(closePaper?, loginUser) ∈ setter ∧
(closePaper?, released) ∈ pStatus) &
pStatus := pStatus ⊕ { (closePaper? → paperclosed) })

CheckPaper ≜ var chkPaper : dom (checker) ∩ dom (pStatus) •
chkPaperIn?chkPaper−→
((loginUser ∈ lecturers ∧
(chkPaper?, loginUser) ∈ checker ∧
(chkPaper?, setting) ∈ pStatus) &
pStatus := pStatus ⊕ { (chkPaper? → checked) })

Figure A.6 (cont.) : Specification of Secure electronic examination system - code block 3 of 6
GradePaper \triangleright= \text{var} \; \text{gradeAwarded} : \mathbb{N};
\text{gradePaper} : \text{dom (grader)} \cap \text{dom (pStatus)};
\text{gradeCandidate} : \text{dom (regPaper)} \triangleright
\text{gradeAwardedIn?gradeAwarded} \rightarrow
\text{gradePaperIn?gradePaper} \rightarrow
\text{gradeCandidateIn?gradeCandidate} \rightarrow
((\text{loginUser} \in \text{lecturers} \land
(\text{gradeCandidate}?, \text{gradePaper}?) \in \text{regPaper} \land
(\text{gradePaper}?, \text{paperclosed}) \in \text{pStatus} \land
(\text{gradePaper}?, \text{loginUser}) \in \text{grader}) \&
\text{result := result } \oplus \{(\text{gradeCandidate}? \rightarrow \text{gradeAwarded}?)\})

PublishGrades \triangleright= \text{var} \; \text{pubGradePaper} : \text{dom (grader)} \cap \text{dom (pStatus)} \triangleright
\text{pubGradePaperIn?pubGradePaper} \rightarrow
((\text{loginUser} \in \text{lecturers} \land
(\text{pubGradePaper}?, \text{paperclosed}) \in \text{pStatus} \land
(\text{pubGradePaper}?, \text{loginUser}) \in \text{grader}) \&
\text{pStatus := pStatus } \oplus \{(\text{pubGradePaper}? \mapsto \text{gradepublished})\})

TakePaper \triangleright= \text{var} \; \text{takePaperAnswer} : \text{ANSWER};
\text{takePaperCandidate} : \text{dom (regPaper)};
\text{takePaper} : \text{ran (regPaper)} \triangleright
\text{takePaperAnswerIn?takePaperAnswer} \rightarrow
\text{takePaperCandidateIn?takePaperCandidate} \rightarrow
\text{takePaperIn?takePaper} \rightarrow
\text{theCandidate := takePaperCandidate?};
((\text{loginUser} \in \text{students} \land
((\text{regStudent}) \sim) \text{loginUser} = \text{takePaperCandidate}? \land
(\text{takePaper}?, \text{released}) \in \text{pStatus} \land
(\text{takePaperCandidate}?, \text{takePaper}?) \in \text{regPaper}) \&
\text{ansPaper := ansPaper } \oplus \{(\text{takePaper}? \mapsto \text{takePaperAnswer}?\})

GetMyAnswers \triangleright= \text{var} \; \text{ownAnswerCandidate} : \text{dom (ansStudent)} \triangleright
\text{ownAnswerCandidateIn?ownAnswerCandidate} \rightarrow
\text{theCandidate := ownAnswerCandidate?};
((\text{loginUser} \in \text{students} \land
((\text{regStudent}) \sim) \text{loginUser} = \text{ownAnswerCandidate}? \&
\text{ownAnswerOut} !
(\text{ansStudent } \{\text{ownAnswerCandidate}\} )) \rightarrow \text{Skip})

RegisterforExam \triangleright= \text{var} \; \text{regExamCandidate} : \text{dom (regStudent)};
\text{regExamPaper} : \text{dom (pStatus)} \triangleright
\text{regExamCandidateIn?regExamCandidate} \rightarrow
\text{regExamPaperIn?regExamPaper} \rightarrow
\text{theCandidate := regExamCandidate?};
((\text{loginUser} \in \text{students} \land
((\text{regStudent}) \sim) \text{loginUser} = \text{regExamCandidate}? \&
\text{regPaper := regPaper } \oplus \{(\text{regExamCandidate}? \mapsto \text{regExamPaper}?\}))

Figure A.6 (cont.) : Specification of Secure electronic examination system - code block 4 of 6
CancelRegistration $\deq \var cancelRegCandidate : \dom(\regStudent);
\begin{align*}
\text{cancelRegPaper} & \colon \ran(\regPaper) \bullet \\
\text{cancelRegCandidateIn} \?	ext{cancelRegCandidate} & \rightarrow \\
\text{cancelRegPaperIn} \?	ext{cancelRegPaper} & \rightarrow \\
\text{theCandidate} & := \cancelRegCandidate?
\end{align*}
\begin{align*}
(\text{loginUser} \in \students \land \\
(\regStudent \not\sim) \text{loginUser} = \cancelRegCandidate? \land \\
(\cancelRegCandidate?, \cancelRegPaper?) \in \regPaper) \& \\
\text{regPaper} & := \text{regPaper} \setminus \{(\cancelRegCandidate? \mapsto \cancelRegPaper?)\}
\end{align*}
\begin{align*}
\text{theCandidate} & := \cancelRegCandidate? \\
(\text{loginUser} \in \students \land \\
(\regStudent \not\sim) \text{loginUser} = \cancelRegCandidate? \land \\
(\cancelRegCandidate?, \cancelRegPaper?) \in \regPaper) \& \\
\text{ownResultOut} & \not\rightarrow (\text{result ownResultCandidate})
\end{align*}

GetOwnResult $\deq \begin{aligned}
\var ownResultPaper & \colon \ran(\regPaper) \\
\text{ownResultCandidate} & \colon \dom(\text{result}) \bullet \\
\text{ownResultPaperIn} \?	ext{ownResultPaper} & \rightarrow \\
\text{ownResultCandidateIn} \?	ext{ownResultCandidate} & \rightarrow \\
\text{theCandidate} & := \text{ownResultCandidate}?
\end{aligned}
\begin{align*}
(\text{loginUser} \in \students \land \\
(\regStudent \not\sim) \text{loginUser} = \text{ownResultCandidate}? \land \\
(\text{ownResultCandidate}?, \text{ownResultPaper}?) \in \text{regPaper} \land \\
(\text{ownResultPaper}?, \text{gradepublished}) \in \pStatus) \& \\
\text{ownResultOut} & \not\rightarrow (\text{result ownResultCandidate}) 
\end{align*}

GetAnyResult $\deq \begin{aligned}
\var anyResultPaper & \colon \ran(\regPaper) \\
\text{anyResultCandidate} & \colon \dom(\text{result}) \bullet \\
\text{anyResultPaperIn} \?	ext{anyResultPaper} & \rightarrow \\
\text{anyResultCandidateIn} \?	ext{anyResultCandidate} & \rightarrow \\
\text{theCandidate} & := \text{anyResultCandidate}?
\end{aligned}
\begin{align*}
(\text{loginUser} \in \lecturers \land \\
(\text{anyResultCandidate}?, \text{anyResultPaper}?) \in \text{regPaper}) \& \\
\text{anyResultOut} & \not\rightarrow (\text{result anyResultCandidate}) 
\end{align*}

SetPaper $\deq \begin{aligned}
\var setQuestionPaper & : \text{QUESTION}; \\
\text{setPaper} & : \dom(\text{setter}) \cap \dom(\pStatus) \bullet \\
\text{setQuestionPaperIn} \?	ext{setQuestionPaper} & \rightarrow \\
\text{setPaperIn} \?	ext{setPaper} & \rightarrow \\
(\text{loginUser} \in \lecturers \land \\
(\text{setPaper}?, \text{loginUser}) \in \text{setter} \land \\
(\text{setPaper}?, \text{checked}) \in \pStatus) \& \\
\text{questions} & := \text{questions} \oplus \\
\text{pStatus} & := \text{pStatus} \oplus \{(\text{setPaper}? \mapsto \text{setQuestionPaper}?)\}; \\
\end{aligned}
\begin{align*}
(\text{loginUser} \in \lecturers \land \\
(\text{releasePaper}?, \text{loginUser}) \in \text{setter} \land \\
(\text{releasePaper}?, \text{checked}) \in \pStatus) \& \\
\text{pStatus} & := \text{pStatus} \oplus \{(\text{releasePaper}? \mapsto \text{released})\}
\end{align*}

ReleasePaper $\deq \begin{aligned}
\var releasePaper & : \dom(\text{setter}) \cap \dom(\pStatus) \bullet \\
\text{releasePaperIn} \?	ext{releasePaper} & \rightarrow \\
(\text{loginUser} \in \lecturers \land \\
(\text{releasePaper}?, \text{loginUser}) \in \text{setter} \land \\
(\text{releasePaper}?, \text{checked}) \in \pStatus) \& \\
\text{pStatus} & := \text{pStatus} \oplus \{(\text{releasePaper}? \mapsto \text{released})\}
\end{aligned}$

HideOthersAnswers $\deq \text{HideOthersAnswers}$

StudentOptions $\deq \text{RegisterforExam, CancelRegistration, TakePaper, GetOwnResult, GetMyAnswers}$

Figure A.6 (cont.): Specification of Secure electronic examination system - code block 5 of 6
LecturerOptions \equiv \left( \begin{array}{c}
\text{AppointSetter} \\
\text{ResignSetter} \\
\text{AppointChecker} \\
\text{ResignChecker} \\
\text{CloseExam} \\
\text{CheckPaper} \\
\text{AppointMarker} \\
\text{ResignMarker} \\
\text{SetPaper} \\
\text{ReleasePaper} \\
\text{GradePaper} \\
\text{PublishGrades} \\
\text{GetAnyResult}
\end{array} \right)

\text{Options} \equiv \left( \begin{array}{c}
\text{LecturerOptions} \\
\text{StudentOptions}
\end{array} \right) \; \text{HideOthersAnswers}

\bullet (\text{Init}) \; \mu X \bullet (\text{Options} ; X)

\text{end}
A Appendix

A.5.3 Case study - ePurse system

A hand-crafted documentation with a formal specification of the ePurse system is presented. A Circu action is defined for each use case in the use case diagram of the ePurse system is shown in Figure A.7. The operations performed by these actions are described in Table A.16. It is important to note that the action Init in Table A.16 is not in Figure A.7. This is because Init initializes system variables and is not executed through external action. The following table of contents may be useful for easy navigation within this case study.

<table>
<thead>
<tr>
<th>Contents of the case study</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation of the ePurse system case study</td>
<td></td>
</tr>
<tr>
<td>- Operations</td>
<td>319</td>
</tr>
<tr>
<td>- Data types of system entities</td>
<td>319</td>
</tr>
<tr>
<td>- State variables</td>
<td>320</td>
</tr>
<tr>
<td>- Formal specification of the system in the Circu notation</td>
<td>324</td>
</tr>
</tbody>
</table>

![Use case diagram for the ePurse system](image)

Figure A.7: Use case diagram for the ePurse system
A.5 Description and formal specification of systems

A.5.3.a Operations

The operations of the system are formalized as Circus actions and listed in Table A.16.

<table>
<thead>
<tr>
<th>Action</th>
<th>Operation performed by the action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SelectAgent</td>
<td>Allows the user to select the current agent who is using the system.</td>
</tr>
<tr>
<td>RecordTrnsctn</td>
<td>Allows the seller to record the value of a transaction in the system.</td>
</tr>
<tr>
<td>ApproveTrnsctn</td>
<td>Allows the ePurse holder to approve charging the value of the transaction from his/her ePurse.</td>
</tr>
<tr>
<td>ShowBalSec</td>
<td>Allows only the ePurse owner to check the balance of the ePurse.</td>
</tr>
<tr>
<td>DoPayment</td>
<td>Allows the terminal to process the transaction by adjusting value of the transaction from the ePurse.</td>
</tr>
</tbody>
</table>

A.5.3.b Data types of system entities

These basic types declared in the system are described in Table A.17.

<table>
<thead>
<tr>
<th>Basic type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION</td>
<td>set of all possible transaction identifiers</td>
</tr>
<tr>
<td>EPURSE</td>
<td>set of all possible epurse identifiers</td>
</tr>
</tbody>
</table>
The free types declared in the system are described in Table A.18.

Table A.18: ePurse system - Description of the free types

<table>
<thead>
<tr>
<th>Free type</th>
<th>Item</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT</td>
<td>buyer</td>
<td>The current agent is the buyer.</td>
</tr>
<tr>
<td></td>
<td>seller</td>
<td>The current agent is the seller.</td>
</tr>
<tr>
<td></td>
<td>terminal</td>
<td>The current agent is the terminal.</td>
</tr>
<tr>
<td>VALIDATED</td>
<td>yes</td>
<td>The transaction has been validated.</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>The transaction has not been validated.</td>
</tr>
</tbody>
</table>

A.5.3.c State variables

A state variable has been declared to represent each organisational component within the system specification. The description of these state variables are included in Table A.19.

Table A.19: ePurse system - Description of the state variables

<table>
<thead>
<tr>
<th>State variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owns</td>
<td>A function that identifies the ePurse owned by a buyer, if any.</td>
</tr>
<tr>
<td>balance</td>
<td>A function that identifies the balance in an ePurse, if any.</td>
</tr>
</tbody>
</table>
A.5  Description and formal specification of systems

transPurse: A function that identifies the ePurse to which a transaction is charged, if any.

transTerminal: A function that identifies the terminal on which a transaction is recorded, if any.

transAmount: A function that identifies the value of a transaction, if any.

transValid: A function that identifies the validation status of a transaction, if any.

currAgent: The current agent who is using the system.

reqPurse: The ePurse of which a balance is being requested.

A.5.3.d State invariants

We assume certain system constraints that must be respected throughout the life of the system. These constraints reflect the relevant organisational rules that we believe are typical in a phone book system. These constraints are defined as state invariants ([Woodcock and Davies, 1996], p. 168) in the system specification.

Table A.20: ePurse system - Description of the state invariants

<table>
<thead>
<tr>
<th>State invariants</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dom(balance) ⊆ ran owns</td>
<td>Every ePurse which has a balance must also have an owner.</td>
</tr>
<tr>
<td>ran transPurse ⊆ dom(balance)</td>
<td></td>
</tr>
</tbody>
</table>
Every ePurse on to which a transaction is recorded must also have a balance recorded.

\[ \text{dom} \{\text{transValid}\} \subseteq \text{dom} \{\text{transAmount}\} \]

Every transaction that has a validation status must also have a transaction value associated with it.

\[ \text{dom} \{\text{transPurse}\} \subseteq \text{dom} \{\text{transAmount}\} \]

Every transaction of an ePurse must also have a transaction value associated with it.

\[ \text{dom} \{\text{transTerminal}\} \subseteq \text{dom} \{\text{transAmount}\} \]

Every transaction recorded in a terminal must also have a transaction value associated with it.

\[ \text{reqPurse} \in \text{ran} \text{owns} \]

The ePurse of which a balance is being requested must be owned by someone.

\[ \text{reqPurse} \in \text{dom} \{\text{balance}\} \]

The ePurse of which a balance is being requested must have a balance recorded.
A.5 Description and formal specification of systems

(This page intentionally left blank)
\[
\begin{align*}
\text{AGENT} &::= \text{terminal} \mid \text{seller} \mid \text{buyer} \\
\text{VALIDATED} &::= \text{no} \mid \text{yes} \\
\text{bmin} : \mathbb{N}
\end{align*}
\]

\textbf{State}

\begin{align*}
\text{transPurse} &: \text{TRANSACTION} \mapsto \text{EPURSE} \\
\text{transTerminal} &: \text{TRANSACTION} \mapsto \text{AGENT} \\
\text{currAgent} &: \text{AGENT} \\
\text{transValid} &: \text{TRANSACTION} \mapsto \text{VALIDATED} \\
\text{balance} &: \text{EPURSE} \mapsto \mathbb{N} \\
\text{transAmount} &: \text{TRANSACTION} \mapsto \mathbb{N} \\
\text{owns} &: \text{AGENT} \mapsto \text{EPURSE} \\
\text{reqPurse} &: \text{EPURSE}
\end{align*}

\[\text{reqPurse} \in \text{dom (balance)}\]
\[\text{reqPurse} \in \text{ran owns}\]
\[\text{dom \{transTerminal\}} \subseteq \text{dom \{transAmount\}}\]
\[\text{dom \{transPurse\}} \subseteq \text{dom \{transAmount\}}\]
\[\text{dom \{transValid\}} \subseteq \text{dom \{transAmount\}}\]
\[\text{ran transPurse} \subseteq \text{dom \{balance\}}\]
\[\text{dom \{balance\}} \subseteq \text{ran owns}\]

\textbf{HideExactBalance}

\[\exists \text{State } \left( \begin{array} {c}
\text{(currAgent \mapsto reqPurse)} \notin \text{owns} \land \\
\text{currAgent} \neq \text{terminal} \land \\
\text{reqPurse} \in \text{dom \{balance\}} \land \\
\text{balance reqPurse} \neq \text{(balance} \text{reqPurse})
\end{array} \right)\]

\textbf{HideMinBalance}

\[\exists \text{State } \left( \begin{array} {c}
\text{(currAgent \mapsto reqPurse)} \notin \text{owns} \land \\
\text{currAgent} \neq \text{terminal} \land \\
\text{reqPurse} \in \text{dom \{balance\}} \land \\
\text{balance reqPurse} \in \{ r : \mathbb{N} \mid r < \text{bmin} \} \land \\
\text{balance} \text{reqPurse} \notin \{ r : \mathbb{N} \mid r < \text{bmin} \}
\end{array} \right)\]

\textbf{channel}

\begin{align*}
\text{recordTransAmountIn, showBalEpurseOut, bsout} &: \mathbb{N} \\
\text{epurseOwnerIn, bybln, approveTransAgentIn, changeAgentIn} &: \text{AGENT} \\
\text{payTransEpurseIn, showEpurseIn, epbln, approveTransEpurseIn} &: \text{EPURSE} \\
\text{payTransIn, approveTransIn, recordTransactionIn} &: \text{TRANSACTION}
\end{align*}

\textbf{channelset}

\begin{align*}
\text{Buyer} &= \{ \text{approveTransIn, approveTransAgentIn, approveTransEpurseIn} \} \\
\text{Seller} &= \{ \text{recordTransAmountIn, recordTransactionIn} \} \\
\text{Terminal} &= \{ \text{showBalEpurseOut, showEpurseIn, epurseOwnerIn, payTransIn, payTransEpurseIn} \}
\end{align*}

Figure A.8: Specification of ePurse system - code block 1 of 2
process ePurse_system \equiv \begin{align*}
\text{state State} \\
\text{SelectAgent} & \equiv \text{var changeAgent : AGENT} \\
& \text{var changeAgentIn?changeAgent} \rightarrow \text{currAgent} := \text{changeAgent}\?\\n\text{RecordTrnsctn} & \equiv \text{var recordTransaction : TRANSACTION} \\
& \text{var recordTransAmount : N} \\
& \text{var recordTransactionIn?recordTransAmount} \rightarrow \text{recordTransAmountIn?recordTransAmount} \rightarrow \text{transAmount} := \text{recordTransAmount} \oplus \{(\text{recordTransAmount} \rightarrow \text{recordTransAmount})\}\} \\
\text{ApproveTrnsctn} & \equiv \text{var approveTransEpurse : EPURSE} \\
& \text{var approveTransAgent : AGENT} \\
& \text{var approveTrans : TRANSACTION} \\
& \text{var approveTransEpurseIn?approveTransEpurse} \rightarrow \text{approveTransEpurseIn?approveTransEpurse} \rightarrow \text{approveTransAgentIn?approveTransAgent} \rightarrow \text{approveTransIn?approveTrans} \rightarrow \text{transPurse} := \text{transPurse} \oplus \{(\text{approveTrans} \rightarrow \text{approveTransEpurse})\}; \\
& \text{transValid} := \text{transValid} \oplus \{(\text{approveTrans} \rightarrow \text{yes})\}; \\
& \text{transTerminal} := \text{transTerminal} \oplus \{(\text{approveTrans} \rightarrow \text{approveTransAgent})\}; \\
\text{ShowBalSec} & \equiv \text{var epurseOwner : dom(owns)}; \\
& \text{var showEpurse : dom(balance) \cap ran owns} \\
& \text{var epurseOwnerIn?epurseOwner} \rightarrow \text{showEpurseIn?showEpurse} \rightarrow \text{reqPurse} := \text{showEpurse}?; \\
& \{(\text{currAgent} = \text{buyer} \land \text{currAgent} = \text{epurseOwner}\? \land \text{epurseOwner} = \text{showEpurse}?; \\
& \text{showBalEpurseOut}(\text{balance showEpurse}?) \rightarrow \text{Skip}\}
\end{align*}

\text{DoPayment} \equiv \text{var payTransEpurse : EPURSE} \\
\quad \text{var payTrans : TRANSACTION} \\
\quad \text{var payTransEpurseIn?payTransEpurse} \rightarrow \text{payTransIn?payTrans}\rightarrow \text{balance} := \text{balance} \oplus \{(\text{payTransEpurse} \rightarrow (\text{balance payTransEpurse}? - \text{transAmount payTrans}?)\})

\text{HideExactBalance} \equiv \text{HideExactBalance} \\
\text{HideMinBalance} \equiv \text{HideMinBalance} \\
\mu X. \left(\langle\text{SelectAgent}\rangle ; \left(\begin{array}{c}
\square (\langle\text{ApproveTrnsctn}\rangle) \\
\square (\langle\text{ShowBalSec}\rangle ; \langle\text{HideExactBalance}\rangle) \\
\square (\langle\text{DoPayment}\rangle) \\
\square (\langle\text{RecordTrnsctn}\rangle)
\end{array}\right) ; X\right)
\end{align*}

\text{end}
**Abbreviations**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANTLR</td>
<td>ANother Tool for Language Recognition</td>
</tr>
<tr>
<td>ATP</td>
<td>Automatic Theorem Prover</td>
</tr>
<tr>
<td>BCF</td>
<td>Banks’s Confidentiality Framework</td>
</tr>
<tr>
<td>BNF</td>
<td>Backus-Naur Form</td>
</tr>
<tr>
<td>CA</td>
<td>Confidentiality Annotation</td>
</tr>
<tr>
<td>CML</td>
<td>COMPASS Modelling Language</td>
</tr>
<tr>
<td>CNL</td>
<td>Controlled Natural Language</td>
</tr>
<tr>
<td>CONCHITA</td>
<td>CONfidentiality CHecker for Incremental Threat Analysis</td>
</tr>
<tr>
<td>CSCW</td>
<td>Computer Supported Cooperative Work</td>
</tr>
<tr>
<td>CSP</td>
<td>Communicating Sequential Processes</td>
</tr>
<tr>
<td>CZT</td>
<td>Community Z Tools</td>
</tr>
<tr>
<td>DLP</td>
<td>Data Leak Prevention</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>E.U.</td>
<td>European Union</td>
</tr>
<tr>
<td>FDR</td>
<td>Failures-Divergence Refinement</td>
</tr>
<tr>
<td>FTC</td>
<td>Federal Trade Commission</td>
</tr>
</tbody>
</table>
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
<td>Graphical user interface</td>
</tr>
<tr>
<td>HAZOP</td>
<td>HAZard and OPerability</td>
</tr>
<tr>
<td>HiVE</td>
<td>Hierarchical Verification Environment</td>
</tr>
<tr>
<td>HOL</td>
<td>Higher-Order Logic</td>
</tr>
<tr>
<td>IBAC</td>
<td>Identity-Based Access Control</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>KAOS</td>
<td>Knowledge Acquisition in autOmated Specification</td>
</tr>
<tr>
<td>LCF</td>
<td>Logic for Computable Functions</td>
</tr>
<tr>
<td>LTL</td>
<td>Linear Temporal Logic</td>
</tr>
<tr>
<td>ML</td>
<td>Meta Language</td>
</tr>
<tr>
<td>OCAML</td>
<td>Object Categorical Abstract Machine Language</td>
</tr>
<tr>
<td>OCL</td>
<td>Object Constraint Language</td>
</tr>
<tr>
<td>OOPSLA</td>
<td>Object-Oriented Programming Systems Languages &amp; Applications</td>
</tr>
<tr>
<td>PII</td>
<td>Personally Identifiable Information</td>
</tr>
<tr>
<td>RBAC</td>
<td>Role-Based Access Control</td>
</tr>
<tr>
<td>RHODIN</td>
<td>Rigorous Open Development Environment for Complex Systems</td>
</tr>
<tr>
<td>SMT</td>
<td>Satisfiability Modulo Theories</td>
</tr>
<tr>
<td>SPIN</td>
<td>Simple Promela INterpreter</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
</tbody>
</table>
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-RT</td>
<td>Unified Modelling Language - Realtime</td>
</tr>
<tr>
<td>UTP</td>
<td>Unifying Theories of Programming</td>
</tr>
<tr>
<td>ZF</td>
<td>Zermelo-Fraenkel set theory</td>
</tr>
</tbody>
</table>


331
List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


List of References


Skon, Jim. Use Case Example Solution, 2016. URL http://cs.mvnu.edu/twiki/bin/view/Main/SsE12014Prob1Sol


List of References


List of References


List of References


List of References


List of References


Wing, J M. A specifier’s introduction to formal methods. Computer, 23(9):8–22, sep 1990. ISSN 0018-9162. doi: 10.1109/2.58215.


