

SA1 Site Safety System

Module: Formal Methods

Name: Ruaidhri Cook

Student Number: P148970

Lecturer: Gordon Dickers

Course: Software Engineering

Contents

1	Introduction	2
2	Types and Globals	2
3	The State of the System	3
4	Initialisation	3
5	Change State Operations	4
5.1	Add Staff	4
5.2	Remove Staff	5
5.3	Change Area	6
5.4	Leave Area	7
5.5	Is Staff In Area	8
5.6	Is Staff On Site	9
6	Formal Methods	10
6.1	Levels of Formal specification	10
6.2	Formal specification in relation to the solution	11
7	References	11

1 Introduction

The security system at Steves Friendly Nuclear Company has work areas that can be occupied that are associated with the site: Reactor Core, Control Room, Admin Building, On Site and Off Site. The company also has multiple types of security level: Tech, Science, Management and Office Staff. Each type of staff has a security level:

	Reactor Core	Control Room	Admin Building	On Site	Off Site
Tech		x	x	x	x
Science	x	x		x	x
Management		x	x	x	x
Office			x	x	x

2 Types and Globals

[STAFF] set of all uniquely identifiable member of staff

RESPONSE Type for Error Messages

RESPONSE::= staffEntered | staffExited | staffAdded | staffRemoved | staffNotPresent | staffInArea | staffNotInArea | accessViolation | staffNotOnSite | staffNotOnSite

Types for relations

WORKAREA::= ReactorCore | ControlRoom | AdminBuilding | OnSite | OffSite

SECURITYLEVEL::= Tech | Science | Management | OfficeStaff

The following relations describe the relationships between SECURITYLEVEL and WORKAREA. This relation defines which security level can access which work areas

Axiomatic box declaration for global

| *canAccess* : *SECURITYLEVEL* $\leftrightarrow$ *WORKAREA*

3 The State of the System

<i>Computer</i>
$staff : \mathbb{P} STAFF$
$belongsTo : STAFF \leftrightarrow SECURITYLEVEL$
$staffEntered : STAFF \leftrightarrow WORKAREA$

4 Initialisation

<i>InitComputer</i>
<i>Computer'</i>
$staffEntered' = OffSite$
$staff' = \emptyset$
$belongsTo' = \emptyset$
$canAccess' = \{ Tech \mapsto ControlRoom, Tech \mapsto AdminBuilding,$
$Tech \mapsto OnSite, Tech \mapsto OffSite,$
$Science \mapsto ReactorCore, Science \mapsto ControlRoom,$
$Science \mapsto OnSite, Science \mapsto OffSite,$
$Management \mapsto ControlRoom, Management \mapsto AdminBuilding,$
$Management \mapsto Onsite, Management \mapsto OffSite,$
$OfficeStaff \mapsto AdminBuilding, OfficeStaff \mapsto Onsite,$
$OfficeStaff \mapsto OffSite\}$

5 Change State Operations

Partial Operations defined here are:

- i) Add Staff Member
- ii) Remove Staff Member
- iii) Change Area
- iv) Leave Area
- v) Is Staff In Area
- vi) Is Staff On Site

5.1 Add Staff

Adds a new staff member. This is composed of two schema $AddStaff_0$ and $AddStaffError$.

$AddStaff_0$
$\Delta Computer$ $p? : STAFF$ $sec? : SECURITYLEVEL$ $area? : WORKAREA$
$p? \notin staff$ $staff' = staff \cup \{p?\}$ $staffEntered' = staffEntered$ $belongsTo' = belongsTo \cup \{sec?\}$

$AddStaffError$ completes the $AddStaff$ operation by specifying how to deal with any errors.

$AddStaffError$
$\Xi Computer$ $p? : STAFF$ $reply! : RESPONSE$
$p? \notin staff$ $reply! : staffRemoved$

Disjunction of the two schemas and the addition of an $staffAdded$ response to the $AddStaff_0$ schema completes the $AddStaff$ schema:

$AddStaff == (AddStaff_0 \wedge [reply! : RESPONSE \mid reply! = staffAdded]) \vee AddStaffError$

5.2 Remove Staff

Removes a new Staff Member. $p?$ must be a Staff member, must be off site.
No other state changes occur.

$RemoveStaff_0$	$\Delta Computer$ $p? : STAFF$ $area? : WORKAREA$
	$p? \in staff$ $staff' = staff \cup \{p?\}$ $p? \in area$ $staffEntered' = staffEntered$
$RemoveStaffError$	$\exists Computer$ $p? : STAFF$ $reply! : RESPONSE$
	$p? \notin staff \wedge reply! = staffNotPresent$ $\vee$ $p? \in staff \wedge p? \in staffEntered \wedge staffEntered =! OffSite \wedge response! = staffInArea$

$RemoveStaff == (RemoveStaff_0 \wedge [reply! : RESPONSE \mid reply! = staffRemoved]) \vee RemoveStaffError$

5.3 Change Area

Changes Area of Staff Member. $p?$ must be a Staff member, must have a security level and must have access to the Work Area. No other state changes occur.

$ChangeArea_0$ $\Delta Computer$ $p? : STAFF$ $area? : WORKAREA$	
$p? \in staff$ $\{p? \mapsto area?\} \in belongsTo \S canAccess$ $staff' = staff$	

An error should be flagged up if someone who is not a staff member tries to enter an area or if $p?$ tries to enter an area they do not have access rights to.

$ChangeAreaError$ $\Xi Computer$ $p? : STAFF$ $area? : WORKAREA$ $reply! : RESPONSE$	
$p? \notin staff \wedge reply! = staffNotPresent$ $\vee$ $p? \in staff \wedge \{p? \mapsto area?\} \notin belongsTo \S canAccess \wedge reply! = accessViolation$	

$ChangeArea == (ChangeArea_0 \wedge [reply! : RESPONSE \mid reply! = staffEntered]) \vee ChangeAreaError$

5.4 Leave Area

Leaves Area. $p?$ must be a Staff member, and must be in an must be in the On Site area. No other state changes occur.

$LeaveArea_0$	_____
$\Delta Computer$ $p? : STAFF$ $area? : WORKAREA$	
$p? \in staff$ $\{p? \mapsto area?\} \in belongsTo \circ canAccess \circ staffEntered = OnSite$ $staffEntered' = OffSite$	

An error should be flagged up if someone who is not a staff member tries to leave an area or if $p?$ tries to leave any area other than On Site.

$LeaveAreaError$	_____
$\exists Computer$ $p? : STAFF$ $area? : WORKAREA$ $reply! : RESPONSE$	
$p? \notin staff \wedge reply! = staffNotPresent$ $\vee$ $p? \in staff \wedge \{p? \mapsto area?\} \in belongsTo \circ canAccess \circ staffEntered = OnSite \wedge reply! = accessViolation$	

$LeaveArea == (LeaveArea_0 \wedge [reply! : RESPONSE \mid reply! = staffExited])$
 $\vee LeaveAreaError$

5.5 Is Staff In Area

Checks if staff member is in a work area building. $p?$ must be a Staff member, and the StaffEntered (current location) must not be either On Site or Off Site. No other state changes occur.

$ \begin{array}{l} \text{IsStaffInArea}_0 \\ \Delta \text{Computer} \\ p? : \text{STAFF} \\ \text{area?} : \text{WORKAREA} \end{array} $	_____
$ \begin{array}{l} p? \in \text{staff} \\ \text{staffEntered} \neq \text{OnSite} \vee \text{OffSite} \end{array} $	

An error should be flagged up if $p?$ is not in a work area building.

$ \begin{array}{l} \text{IsStaffInAreaError} \\ \exists \text{Computer} \\ p? : \text{STAFF} \\ \text{area?} : \text{WORKAREA} \\ \text{reply!} : \text{RESPONSE} \end{array} $	_____
$ \begin{array}{l} p? \notin \text{staff} \wedge \text{reply!} = \text{staffNotPresent} \\ \vee \\ p? \in \text{staff} \wedge \text{staffEntered} = \text{OnSite} \vee \text{OffSite} \wedge \text{reply!} = \text{staffNotInArea} \end{array} $	

$\text{IsStaffInArea} == (\text{IsStaffInArea}_0 \wedge [\text{reply!} : \text{RESPONSE} \mid \text{reply!} = \text{staffInArea}]) \vee \text{IsStaffInAreaError}$

5.6 Is Staff On Site

Checks if staff member is on Site. $p?$ must be a Staff member, and the StaffEntered (current location) must not be Off Site. No other state changes occur.

$IsStaffOnSite_0$	_____
$\Delta Computer$ $p? : STAFF$ $area? : WORKAREA$	
$p? \in staff$ $staffEntered \neq OffSite$	

An error should be flagged up if $p?$ is not on site.

$IsStaffOnSiteError$	_____
$\exists Computer$ $p? : STAFF$ $area? : WORKAREA$ $reply! : RESPONSE$	
$p? \notin staff \wedge reply! = staffNotPresent$ $\vee$ $p? \in staff \wedge staffEntered = OffSite \wedge reply! = staffNotOnSite$	

$IsStaffOnSite == (IsStaffOnSite_0 \wedge [reply! : RESPONSE \mid reply! = staffOnSite]) \vee IsStaffOnSiteError$

6 Formal Methods

Formal Methods are often categorised as mathematical based techniques coupled with tools that can be used to synthesise and analyse software systems, this allows users using formal methods to look at software systems and engineer requirements, create specifications and conceptualize design & architecture, to name just a few of its uses. Formal Methods also allows software engineering to be integrated with mathematical theory as well as with physics and other fields that use formulae and calculation.[1][2][3]

6.1 Levels of Formal specification

Formal methods can also take on multiple forms, and can varying levels of formal verification. Haxthausen[1] framed this in the following grading system:

- Formal specification
- Formal specification and semi-formal verification
- Formal specification and formal verification

The most basic formal methods classification is a solution that only uses a formal specification without any kind of verification. A formal specification often accompanies informal specification methods such as natural language or Pseudocode to describe a solution. Unlike the informal specification, the formal specification will use advanced languages that allow the use of precise and mathematical syntax, this solutions can be presented in their pure mathematical form and are used to maximise precision and to turn abstract solutions into something that is easy to understand.

These formal specifications can further be built upon through the use of verification which can come in the form of semi-formal or just the standard formal. However these specifications are only enabled for use because of the use of formal specification. Standard, more informal, methods of verification would be the use of testing and inspection of the code, however these methods are only as useful as the individual(s) employing them and they are also limited in their scope because large projects would be difficult to comb through for errors. The formal verification is done through the construction of mathematical proofs, which are a compelling and objective argument made through the use of mathematics. The semi-formal approach uses a mixture of natural language combined with mathematics to provide a proof that can easily interpreted by the individual reading it, however it can come in varying degrees with some semi formal specification being very centric around natural language but with others being more focused on the mathematical syntax using natural language more sparingly, meaning it is more appropriate to be read by specialist in the field. The formal approach differs as the whole proof is written through the use of mathematical syntax, and is often unreadable to non-specialist.[1][2][3][4]

6.2 Formal specification in relation to the solution

The solution above will fall into one of the formal methods classifications, judging by just the appearance of the solution it's clear to see there is a mixture of both natural language and advanced mathematical syntax, this would suggest that the solution comes under the formal specification and semi-formal verification category, moreover the solution is also very dependant on natural language which may denote that the solution has a lower degree of formality and could easily be interpreted by specialist and may also be easy to understand by the layman.

7 References

- [1] A. E. Haxthausen, An Introduction to Formal Methods for the Development of Safety-critical Applications, pp. 132, 2010.
- [2] J. Rushby, Automated Formal Methods Enter the Mainstream, J. Univers. Comput. Sci., vol. 13, no. 5, pp. 650660, 2007.
- [3] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, Formal methods, ACM Comput. Surv., vol. 41, no. 4, pp. 136, 2009.
- [4] Universitt Tbingen, Semiformal Verification of embedded Software. [Online]. Available: <http://www.ti.uni-tuebingen.de/Semiformal-Verification.357.0.html>. [Accessed: 05-May-2018].