

Steve's Friendly Nuclear Company Formal Specification

Jonathan Evans

May 8, 2018

Contents

1	Introduction	3
2	Table of Security Level access	3
3	Types and declarations	3
4	State of system	4
5	Initialisation of system	4
6	Operations	5
6.1	Add Staff	5
6.2	Remove Staff	6
6.3	Change Area	6
6.4	Leave Area	7
6.5	Is Staff in Area	7
6.6	Is Staff on Site	8
6.7	Formal methods	8

1 Introduction

Steve's Friendly Nuclear Company has a need for strict security. A system is needed to manage the areas a type of user can enter. This specification details the different states the system will be in when different actions are triggered. The reason for this specification is that it can help identify any potential errors that may occur before the development stage of this program.

Implementing this system will allow the company to account for every member of staff in an emergency situation where an employee may be stuck inside the building.

2 Table of Security Level access

	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_Staff			x	x	x

3 Types and declarations

[STAFF] contains all kinds of employee displayed in the above table:
{ Tech, Science, Management, Office_Staff }

Work_area ::= Reactor_Core | Control_Room | Admin_Building | On_Site | Off_Site

Security_level ::= Tech | Science | Management | Office_Staff

MESSAGES ::= staff_entered | staff_exited | staff_removed | staff_not_present | staff_in_area | staff_not_in_area | access_violation

Axiomatic declaration:

| $canAccess : Security_level \leftrightarrow Work_area$

4 State of system

Security_system
 $users : \mathbb{P} STAFF$
 $belongsTo : STAFF \leftrightarrow Security_level$
 $staff_entered : STAFF \leftrightarrow Work_area$

5 Initialisation of system

InitSecurity_System
Security_System'
 $staff_entered' = \emptyset$
 $users' = \emptyset$
 $belongsTo' = \emptyset$
 $canAccess' = \{$
 $Tech \mapsto Control_Room,$
 $Tech \mapsto Admin_Building,$
 $Tech \mapsto On_Site,$
 $Tech \mapsto Off_Site,$
 $Science \mapsto Reactor_Core,$
 $Science \mapsto Control_Room,$
 $Science \mapsto On_Site,$
 $Science \mapsto Off_Site,$
 $Management \mapsto Control_Room,$
 $Management \mapsto Admin_Building,$
 $Management \mapsto On_Site,$
 $Management \mapsto Off_Site,$
 $Office_Staff \mapsto Admin_Building,$
 $Office_Staff \mapsto On_Site,$
 $Office_Staff \mapsto Off_Site$
 $\}$

6 Operations

- Add Staff
- Remove Staff
- Change Area
- Leave Area
- Is Staff in Area
- Is Staff on Site

6.1 Add Staff

AddStaff

$\Delta Security_System$

$p? : STAFF$

$p? \notin users$

$users' = users \cup \{p?\}$

$staff_entered' = staff_entered$

$belongsTo' = belongsTo$

AddStaffError

$\Xi Security_System$

$p? : STAFF$

$reply! : MESSAGE$

$p? \in users \wedge MESSAGE! = staff_in_area$

$\vee$

$p? \notin Security_level \wedge MESSAGE! = access_violation$

6.2 Remove Staff

RemoveStaff	$\Delta \text{Security_System}$ $p? : \text{STAFF}$
	$p? \in \text{users}$ $\text{users}' = \text{users} \setminus p?$ $p? \notin \text{staff_entered}$ $p? \notin \text{belongsTo}$ $\text{staff_entered}' = \text{staff_entered}$ $\text{belongsTo}' = \text{belongsTo}$
RemoveStaffError	$\Xi \text{Security_System}$ $p? : \text{STAFF}$ $\text{reply!} : \text{MESSAGE}$
	$p? \notin \text{users} \wedge \text{reply!} = \text{staff_not_present}$

6.3 Change Area

ChangeArea	$\Delta \text{Security_System}$ $p? : \text{STAFF}$ $a? : \text{Work_area}$
	$p? \in \text{users}$ $\text{users}' = \text{users}$ $\text{belongsTo}' = \text{belongsTo} \cup \{p? \mapsto a?\}$
ChangeAreaError	$\Xi \text{Security_System}$ $p? : \text{STAFF}$ $a? : \text{Work_area}$ $\text{reply!} : \text{MESSAGE}$
	$p? \notin \text{users} \wedge \text{reply!} = \text{staff_not_present}$

6.4 Leave Area

LeaveArea
$\Delta \text{Security_System}$
$p? : \text{STAFF}$
$a? : \text{Work_area}$
$p? \in \text{users}$
$p? \in \text{belongsTo}$
$\text{users}' = \text{users}$
$\text{belongsTo}' = \text{belongsTo} \{p? \mapsto a?\}$
$p? \notin \text{staff_entered}$
$\text{staff_entered}' = \text{staff_entered}$

LeaveAreaError
$\Xi \text{Security_System}$
$p? : \text{STAFF}$
$a? : \text{Work_area}$
$\text{reply!} : \text{MESSAGE}$
$p? \notin \text{users} \wedge \text{reply!} = \text{staff_not_present}$
$\vee$
$p? \in \text{users} \wedge p? \notin \text{belongsTo} \wedge \text{reply!} = \text{staff_not_present}$

6.5 Is Staff in Area

IsStaffInArea
$\Delta \text{Security_System}$
$p? : \text{STAFF}$
$p? \in \text{staff_in_area}$

6.6 Is Staff on Site

$IsStaffonSite$	_____
$\Delta Security_System$	
$p? : STAFF$	
$p? \in On_Site$	

6.7 Formal methods

One way code can be tested for errors is by manually looking through the code for errors and using a debugging tool. However, it may be ideal to prove that a system will work before reaching the development stage, as planning a program can be easier than restructuring the code-base if the program is big and complicated.

By incorporating formal methods, code can be mathematically verified as correct before programming has begun. This can be useful when creating a system where an error could be catastrophic, such as a nuclear power plant room access system.

There are two kinds of methods for proving that code is mathematically correct - semi-formal verification and formal verification. This formal specification uses semi-formal verification [1]. Formal proofs allows for code to be mathematically verified as complete by using solely a proofing language and mathematical language. Semi-formal proofs incorporate natural language with mathematics to prove that a program is correct.

The given specification uses the Z-Specification language for creating mathematical proofs - this incorporates the use of set theory to determine how the state of a system is affected through various changes.

This semi-formal specification can be moved to a formal specification that uses formal proofs by removing all natural language using a proofing language alone. When doing so, the mathematical language must be more descriptive

of the system. The natural language and mathematics included in this specification should contain enough information to convert this specification into a formal specification.

When creating a formal specification, use of a theorem proving tool could aid the creation by checking if a program is mathematically correct. According to Hauxthausen [1], a notable theorem proving tool is MetaMath. As described on it's website, this is a language "that can express theorems in abstract mathematics, accompanied by proofs that can be verified by a computer program" [2]

References

- [1] An introduction to formal methods for the development of safety-critical applications. [Online]. Available: <http://orbit.dtu.dk/ws/files/137536957/FormalMethodsNoteTS.pdf>
- [2] Metamath. [Online]. Available: <http://us.metamath.org/>