EECS4312 Isolette Assignment

Juan Loja (lojag95@cse.yorku.ca)
Sadman Sakib Hasan (cse23152@cse.yorku.ca)

November 6, 2017

Prism account used for submission: cse23152@cse.yorku.ca

Requirements Document: Temperature control for an Isolette

Revisions

Date	Revision	Description
22 October 2017	1.0	Initial requirements document
31 October 2017	1.1	Apply changes to initial function tables
2 November 2017	2.0	Insert updated function tables
3 November 2017	2.1	Add Use Cases
4 November 2017	2.2	Add Acceptance Tests
5 November 2017	3.0	Add Appendix

Contents

List of Figures

List of Tables

1. System Overview

The System Under Development (SUD) is a computer controller for the thermostat of an Isolette.[1] An Isolette is an incubator for for an infant that provides controlled temperature, humidity and oxygen (Fig. 1). Isolettes are used extensively in Neonatal Intensive Care Units for the care of premature infants.

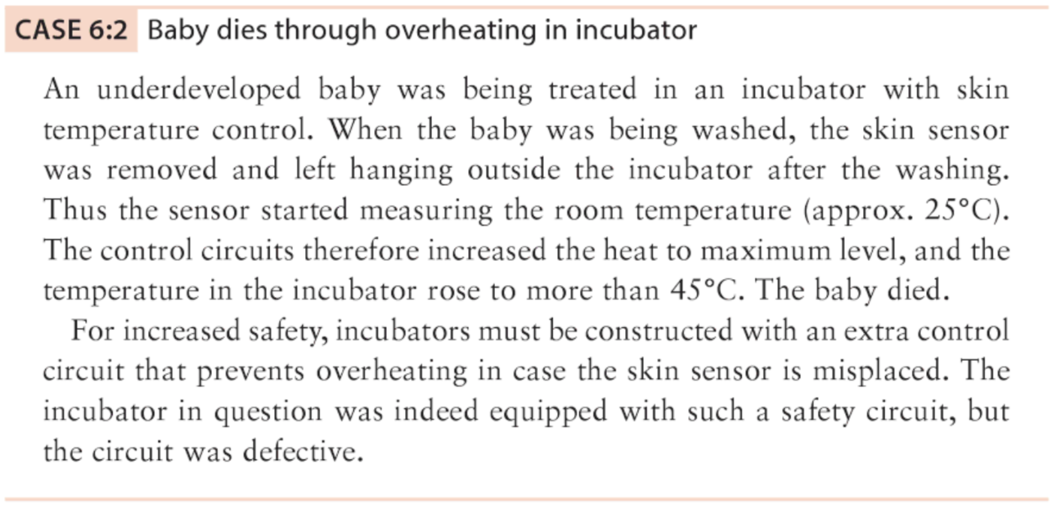
This requirements document is specifically for the control of temperature. The purpose of the Isolette computer controller is to maintain the air temperature of an Isolette within a desired range. It senses the current temperature of the Isolette and turns the heat source on and off to warm the air as needed. If the temperature falls too far below or rises too far above the desired temperature range, it activates an alarm to alert the nurse. The system allows the nurse to set the desired temperature range and to set the alarm temperature range outside the desired temperature range of which the alarm should be activated. This requirements documents follows the specification in [1] (Appendix A) except where noted.

Figure 1: Isolette

Many babies have dies due to faulty incubators. There is thus a standard that manufacturers must satisfy. Modern incubators are equipped with alarms for air temperature, skin temperature, oxygen concentration and humidity. The alarms are both visual such

[1]The image in Fig 1 is from: `www.nufer-medical.ch`.

as red warning lamps, and audio such as beep signals. Once measured measured values exceed permitted limits as well as when faults occur in sensors. For one such incident leading to death see “Medical Devices: Use and Safety” shown in Fig. 2.


CASE 6:2 Baby dies through overheating in incubator

An underdeveloped baby was being treated in an incubator with skin temperature control. When the baby was being washed, the skin sensor was removed and left hanging outside the incubator after the washing. Thus the sensor started measuring the room temperature (approx. 25°C). The control circuits therefore increased the heat to maximum level, and the temperature in the incubator rose to more than 45°C. The baby died.

For increased safety, incubators must be constructed with an extra control circuit that prevents overheating in case the skin sensor is misplaced. The incubator in question was indeed equipped with such a safety circuit, but the circuit was defective.

Figure 2: Incubator Safety Problems [2, p98]

2. Goals

The high-level goals (G) of the system are:

- G1—The Infant should be kept at a safe and comfortable temperature.
- G2—The Nurse should be warned if the Infant becomes too hot or too cold.
- G3—The cost of manufacturing the computer controller for the thermostat should be as low as possible.

3. Context Diagram

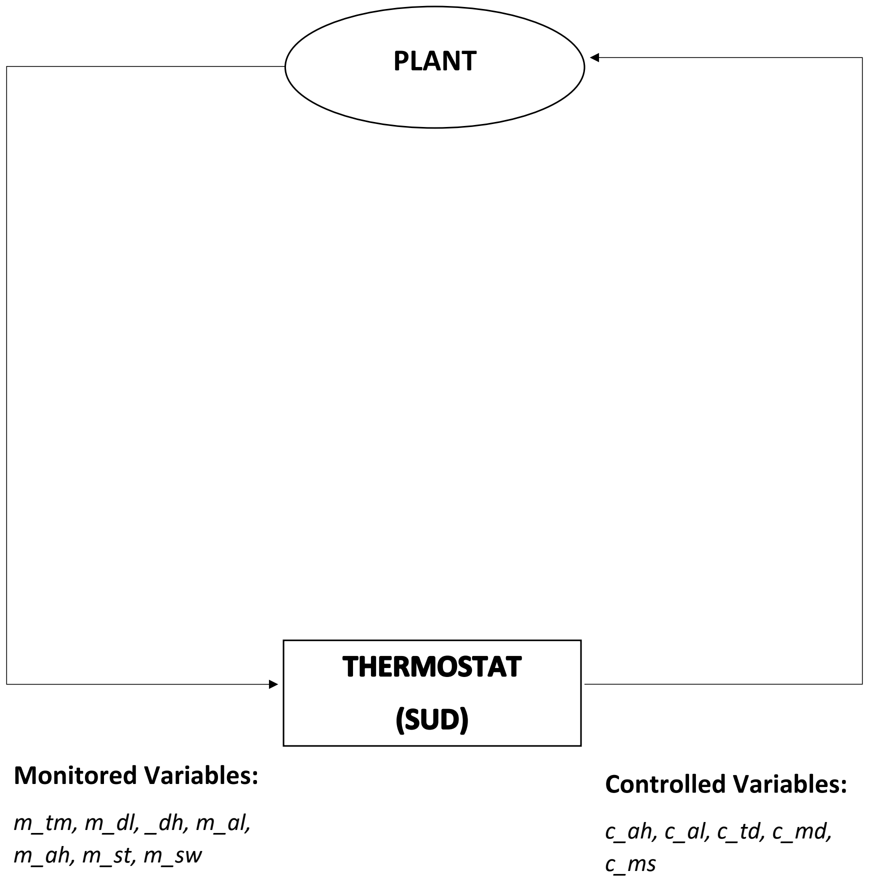


Figure 3: Context diagram for the SUD

4. Monitored Variables

The monitored variables are a subset of those described in [1].[2] There is a single status variable *m_st* that is *invalid* whenever any one of the operator inputs or temperature sensor are in a failed state. Otherwise types and ranges are as in [1].

Name	Type	Range	Units	Physical Interpretation
m_tm	$\mathbb{R}$	$68 \mathinner{..} 105$	°F	actual temperature of Isolette air temperature from sensor
m_dl	$\mathbb{Z}$	$97 \mathinner{..} 99$	°F	desired lower temperature set by operator
m_dh	$\mathbb{Z}$	$98 \mathinner{..} 100$	°F	desired higher temperature set by operator
m_al	$\mathbb{Z}$	$93 \mathinner{..} 98$	°F	lower alarm temperature set by operator
m_ah	$\mathbb{Z}$	$99 \mathinner{..} 103$	°F	higher alarm temperature set by operator
m_st	Enumerated	{valid, invalid}		status of sensor and operator settings
m_sw	Enumerated	{on, off}		switch set by operator

Table 1: Monitored Variables

[2]With some change of nomenclature. Monitored variables have an “m” prefix.

5. Controlled Variables

The controlled variables are a subset of those described in [1].[3] In addition, there is a mode display c_md and a message display c_ms.[4]

Name	Type	Range	Units	Physical Interpretation
c_hc	Enumerated	{on, off}		heat control: command to turn heat source on or off
c_td	$\mathbb{Z}$	$\{0\} \cup \{68 \mathinner{..} 105\}$	°F	displayed temperature of Isolette (zero when Isolette is off)
c_al	Enumerated	{off, on}		sound alarm to call nurse
c_md	Enumerated	{off, init, normal, fail}		mode of Isolette operation (failed if $m\_st = invalid$)
c_ms	Enumerated	{ok, too_hot_alarm, too_cool_alarm, warming_up, cooling_down, system_error}		messages to display to nurse

Table 2: Controlled Variables

[3]With some change of nomenclature. Controlled variables have a "c" prefix.
[4]The mode "off" is added to that of Fig. A-4 in [1], and the mode transitions have been changed.

6. Mode Diagram

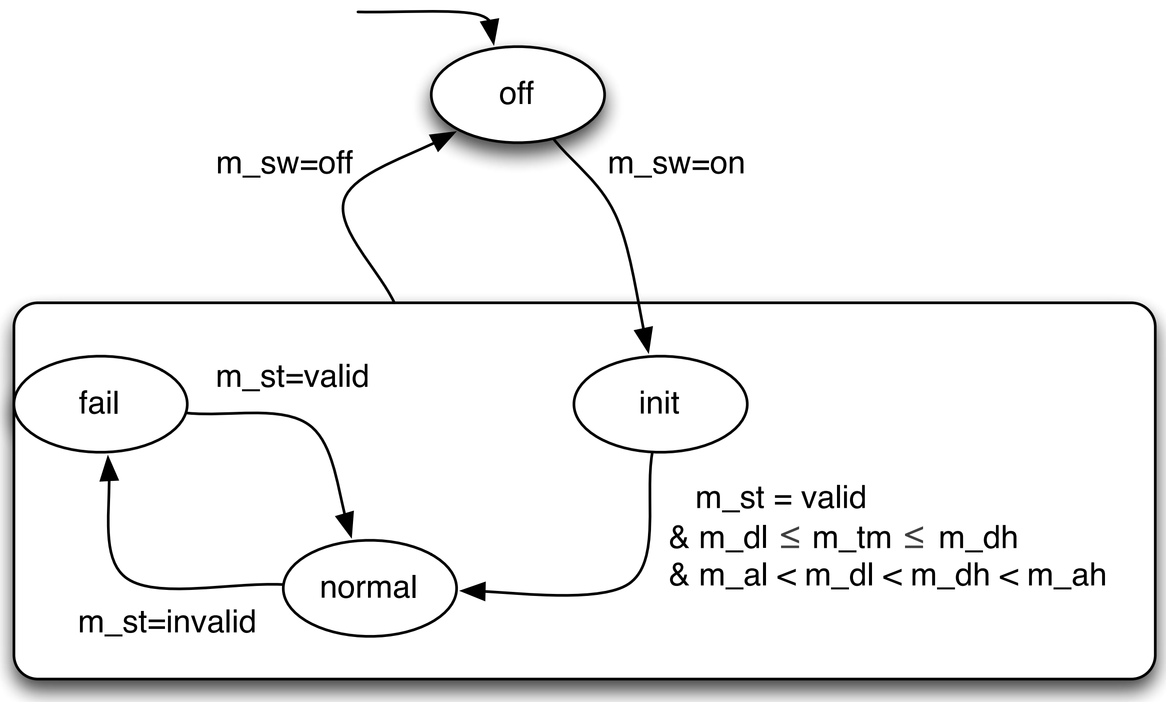


Figure 4: Mode diagram for the states

Rationale: A mode diagram denotes the possible states the system can be in. They are *abstract* variables whose output are visible to the client, in this case the nurse. In a particular state, the *guard* condition has to be satisfied in order for the mode to change, for example: in order to transition from the *normal* state to the *fail* state, the condition $m\_st = invalid$ has to hold true and if the condition does not hold, the states are unchanged.

7. R-Descriptions

The following are the R-descriptions for the System Under Description.

REQ1	The *controller* shall operate in one of four modes: *off*, *init*, *normal* and *fail*.	See statechart in Fig. 4.

Rationale: The Isolette can show four statuses to the Nurse.

- *off* mode indicates that the Isolette is switched off. If the Isolette gets switched on, move to the *init* state.
- *init* mode indicates that the Isolette has just been switched on and the temperature sensor is still not in the desired range. If the input from the nurse is valid and the temperatures are in their desired ranges then move to the *normal* state.
- *normal* mode indicates everything is under control (i.e all inputs are correct and the sensors are working fine). If something goes wrong, then move on to the *fail* state.
- *fail* mode indicates something went wrong and sound an alarm. The nurse can either then turn the Isolette off to go to the *off* state or fix the input temperature ranges to go back to the *normal* state.

REQ2	In the *normal* mode, the temperature controller shall maintain current temperature inside the Isolette within a set temperature range (the *desired* range).	The *desired* temperature range is *m_dl .. m_dh*. If the current temperature *m_tm* is outside this range, the controller shall turn the heater on or off via the controlled variable *m_hc* to maintain the desired state.

Rationale: The *desired temperature range* will be set by the nurse to the desired range based on the infant's weight and health. The controller shall maintain the current temperature within this range under normal operation.

The following relevant hazard was identified through the safety assessment process:

- **H1**: Prolonged exposure of Infant to unsafe heat or cold;
- *Classification*: catastrophic;
- *Probability*: $< 10^{-9}$ per hour of operation.

To ensure that probability of hazard H1 is 10^{-9} per hour of operation, the following derived safety requirement shall apply to the Isolette controller:

REQ3	In *normal* mode, the controller shall activate an alarm whenever • the current temperature falls outside the *alarm* temperature range (either through temperature fluctuation or a change in the alarm range by an operator), or • a failure is signalled in any of the input devices (temperature sensor and operator settings).	The alarm temperature range is $m\_al \mathinner{..} m\_ah$. Monitored variable m_st in Table 1 shows "invalid" when any of the input signals fail.

Rationale: The alarm needs to go off to alert the nurse during critical situations. Most importantly, in the *normal* mode since that is when the child is first put into the Isolette. It must be ensured that the Isolette is capable of handling the child and in any case when it cannot, an alarm must go off to notify the nurse.

REQ4	Once the alarm is activated, it becomes deactivated in one of two ways: • The nurse turns off the Isolette; • The alarm has lasted for 10 seconds, and after 10 seconds or more the alarm conditions are removed.	The Isolette can be switched off by setting m_sw to off. The alarm condition can be removed when the alarm c_hc has been activated for more than 10 seconds. Refer to Table 2 and 8.

Rationale: The Alarm Temperature Range will be set by the Nurse based on the Infants weight and health. Once the alarm is activated (i.e the temperature falls beyond the range), the Infant should be removed from the Isolette and the Isolette should be turned off. If the nurse fails to accomplish that within 10 seconds, the alarm gets deactivated with the infant inside.

REQ5	If mode is *normal* and the value of the Current Temperature is greater than or equal to the Lower Alarm Temperature +0.5 and less than or equal to the Upper Alarm Temperature -0.5, the alarm shall be set to *off*.	When the current temperature is between a delta value, 0.5 of the alarm temperature range, $m\_al + 0.5 <= m\_tm <= m\_ah - 0.5$ then c_al should be off. Refer to Table 2 and 8.

Rationale: The alarm should be turned off at the same moment that the Displayed Temperature shows a value greater than the Lower Alarm Temperature and less than the Upper Alarm Temperature.

REQ6	The Thermostat shall set the value of the Heat Control depending on the Current Temperature.	The heat control of the Isolette is the control variable *c_hc*. Refer to Table 2 and 5.

Rationale: The controlled variable *c_hc* is used by the thermostat to turn the Heat Control on and off to maintain the Current Temperature in the Isolette within the Desired Temperature Range, i.e within *m_dl* and *m_dh*. If the Current Temperature is below the Desired Low Temperature, the heat control should be set on. If the Current Temperature is above the Desired High Temperature, the heat control should be set off.

REQ7	In the *init* and *normal* modes, the displayed temperature shall be set to the value of the current temperature rounded to the nearest integer.	The displayed temperature of the Isolette is the control variable *c_td*. Refer to Table 2 and 6.

Rationale: The controlled variable *c_td* is used by the thermostat to show the Current Temperature of the Isolette. The Displayed Temperature is calculated using the mathematical floor function of the Current Temperature to avoid showing invalid temperatures to the nurse.

8. E-descriptions

The following are the environmental assumption made on the Isolette System:

ENV1	The current temperature received from the sensor is a a real number in the range 68.0 to 105.0°F.	The monitored variable *m_tm* in Table 1 specifies the range, type and unit for the Current Temperature received from the sensor.

Rationale: This is the specified range of operation of the Isolette. The lower end of this range is useful for monitoring an Isolette that is warming to the Desired Temperature Range. The upper end is set to be greater than the Higher Alarm Temperature.

ENV2	The desired and alarm temperatures received from the operator are all in increments of 1°F.	The monitored variables *m_tm*, *m_dl*, *m_dh*, *m_al* and *m_ah* set in an increment of 1°F. Refer to Table 1 to see details about the monitored inputs.

Rationale: Marketing studies have shown that customers prefer to set temperatures in 1 degree increments. A resolution 1F is sufficient to be consistent with the functionality and performance for the Isolette.

ENV3	The Lower Alarm Temperature will always be >= 93°F.	The monitored variables *m_al* denotes the Lower Alarm Temperature. Table 1 specifies the range, type and unit for the Lower Alarm Temperature set by the Nurse.

Rationale: Exposure to temperatures less than 93°F will result in hypothermia, which can lead to death within a few minutes for severely ill pre-term infants.

ENV4	The Higher Alarm Temperature will always be <= 103°F.	The monitored variables *m_ah* denotes the Higher Alarm Temperature. Table 1 specifies the range, type and unit for the Higher Alarm Temperature set by the Nurse.

Rationale: Exposure to temperatures greater than 103°F will result in hyperthermia, which can lead to cardiac arrhythmias and febrile seizures within a few minutes.

ENV5	The Lower Desired Temperature will always be >= 97°F.	The monitored variables *m_dh* denotes the Lower Desired Temperature. Table 1 specifies the range, type and unit for the Lower Desired Temperature set by the Nurse.

Rationale: Exposing the Infant to temperatures lower than 97°F may result in excessive heat loss and drop in heart rate secondary to metabolic acidosis.

9. Abstract variables needed for the Function Table

No abstract variables were used for the function tables.

10. Function Tables

Below are the function tables for each of the controlled variables:

10.1. Function Table for Mode Control: c_md

				$c\_md(i)$
$i = 0$				off
$i > 0$	$m\_sw(i) = off$			off
	$\neg(m\_sw(i) = off)$	$c\_md(i-1) = off$		$init$
		$c\_md(i-1) = init$	S_1	$normal$
			$\neg S_1$	$c\_md(i-1)$
		$c\_md(i-1) = normal$	$m\_st(i) = invalid$	$fail$
			$m\_st(i) = valid$	$c\_md(i-1)$
		$c\_md(i-1) = fail$	$m\_st(i) = invalid$	$c\_md(i-1)$
			$m\_st(i) = valid$	$normal$

Table 3: Function Table for Mode Control

S_1	$m\_st(i) = valid \wedge$ $m\_dl(i) \leq m\_tm(i) \leq m\_dh(i) \wedge$ $m\_al(i) < m\_dl(i) < m\_dh(i) < m\_ah(i)$	This is the condition to be met for system to go from *init* to *normal*

Table 4: Definition of S_1

10.2. Function Table for Heat Control: c_hc

				$c\_hc(i)$
$i = 0$				off
$i > 0$	$m\_sw(i) = off$			off
	$\neg(m\_sw(i) = off)$	$\neg(m\_dl(i) < m\_dh(i))$		$c\_hc(i-1)$
		$m\_dl(i) < m\_dh(i)$	$m\_tm(i) < m\_dl(i)$	on
			$m\_dl(i) <= m\_tm(i) \land m\_tm(i) <= m\_dh(i)$	$c\_hc(i-1)$
			$m\_tm(i) > m\_dh(i)$	off

Table 5: Function Table for Heat Control

10.3. Function Table for Tempearature Display Control: c_td

		$c\_td(i)$
$i = 0$		0
$i > 0$	$c\_md(i-1) = off \vee c\_md(i-1) = fail$	
	$c\_md(i-1) = init \vee c\_md(i-1) = normal$	$floor(m\_tm(i) + 0.5)$

Table 6: Function Table for Temperature Display Control

10.4. Function Table for Message Display Control: c_ms

	$c\_ms(i)$
$m\_st(i) = invalid$	$system\_error$
$m\_tm(i) > m\_ah(i)$	too_hot_alarm
$m\_tm(i) < m\_al(i)$	too_cool_alarm
$m\_al(i) < m\_tm(i) < m\_dl(i)$	$warming\_up$
$m\_dh(i) < m\_tm(i) < m\_ah(i)$	$cooling\_down$
$ELSE$	ok

Table 7: Function Table for Message Display Control

10.5. Function Table for Alarm Control: c_al

							$c\_al(i)$
$i = 0$							off
$i > 0$	S_2						off
	$\neg S_2$	S_3					$c\_al(i-1)$
		$\neg S_3$	S_4				on
			$\neg S_4$	$c\_al(i-1) = off$			$c\_al(i-1)$
				$c\_al(i-1) = on$	S_5		off
					$\neg S_5$	$m\_sw(i) = off$	off
						$m\_sw(i) = on$	on

Table 8: Function Table for Alarm Control

S_2	$c\_md(i-1) = off \vee c\_md(i-1) = init$	Mode is off or $init$. Else it is in $normal$ or $fail$ mode.
S_3	$(m\_al(i) <= m\_tm(i) \wedge m\_tm(i) < m\_al(i) + 0.5) \vee$ $(m\_ah(i) - 0.5 <= m\_tm(i) \wedge m\_tm(i) <= m\_ah(i))$	This is the Hysteresis Region.
S_4	$m\_tm(i) > m\_ah(i) \vee m\_tm(i) < m\_al(i) \vee$ $m\_st(i) = invalid$	Conditions for when the alarm is on.
S_5	$held\_for(alarm, 10)(i-1)$	Checks if alarm was on for at least 10 seconds.

Table 9: Definitions for S_2, S_3, S_4 and S_5

11. Validation

The PVS Proof Validation in section E outlines the proof summary of completeness, disjointness and validation of the requirements using PVS. There are a total of 26 formulas which has been generated and proved.

- **mode_ft**: There are 9 TCC's generated and proved for the *mode_ft* function table. The proofs validate the disjointness and completeness of the function table to verify the requirements.
- **heat_control_ft**: There are 3 TCC's generated and proved for the *heat_control_ft* function table. The proofs validate the disjointness and completeness of the function table to verify the requirements.
- **temperature_display_ft**: There are 3 TCC's generated and proved for the *temperature_display_ft* function table. The proofs validate the disjointness and completeness of the function table to verify the requirements.
- **alarm_ft**: There are 5 TCC's generated and proved for the *alarm_ft* function table. The proofs validate the disjointness and completeness of the function table to verify the requirements.
- **spec_inv_holds**: This provides the proof of the specification implying the invariant. Specification is the conjunction of all function tables which should imply the invariants are true at all time.
- **usecase1 - usecase5**: This provides the proof for all of the five different use cases.

12. Use Cases

The following use case is defined informally and then formally using logic in PVS:

a) **Informal**: Assume the Isolette is switched on and an infant has already been placed in the incubator. The Isolette is in the normal mode and certainly the current temperature drops below the lower alarm temperature without any other fault in the sensors. Then the alarm should become **activated** and the message displayed to the the nurse should indicate the the Isolette is too cool.

b) **Formal**: Below is the PVS conjecture for the above use case:

```
usecase4: CONJECTURE
      c_md(2) = normal
      AND m_sw(3) = on
      AND m_tm(3) = 90
      AND m_al(3) = 93
      AND m_ah(3) = 103
      AND NOT (m_st(3) = invalid)
      AND alarm_ft(3)
      AND message_ft(3)
      AND mode_ft(3)
      IMPLIES
      c_al(3) = on
      AND c_ms(3) = too_cool_alarm
      AND c_md(3) = c_md(2)
```

Refer to section D for additional use cases.

13. Acceptance Tests

The acceptance test below describes the use case in the previous section:

Description: The Acceptance Test shall validate that at any given time t such that $t > 0$ and if the Current Temperature inside the Isolette has dropped below the Lower Alarm Temperature Range then the Alarm should be activated and an adequate message of the situation should be displayed to the nurse.

Pre-condition:

```
        t > 0
        AND c_md(t-1) = normal
        AND m_sw(t) = on
        AND m_tm(t) = 90
        AND m_al(t) = 93
        AND m_ah(t) = 103
```

Action:

```
        alarm_ft(t)
        AND message_ft(t)
        AND mode_ft(t)
```

Post-condition:

```
        c_al(t) = on
        AND c_ms(t) = too_cool_alarm
        AND c_md(t) = c_md(t)
```

References

[1] US FAA. Requirements Engineering Management Handbook. Technical Report DOT/FAA/AR-08/32, U.S. Department of Transportation Federal Aviation Administration, June 2009.

[2] Bertil Jacobson and Alan Murray. *Medical Devices: Use and Safety*. Elsevier, 2007.

A. PVS Time Theory for held-for operator

Below is the PVS Theory using the *held-for* operator:

```
Time[delta: posreal]: THEORY
 BEGIN

  DTIME: TYPE = nat

  init(i: DTIME): bool = i = 0

  RTIME: TYPE = {t: nnreal | (EXISTS (i: DTIME): t = i * delta)}

  TIME: TYPE = nnreal

  POS_DTIME: TYPE = posnat

  r2d(t: RTIME): DTIME = t / delta

  d2r(i: DTIME): RTIME = i * delta

  DURATION: TYPE = nnreal

  held_for(p: pred[DTIME], d: DURATION)(i: DTIME): bool =
      (FORALL (j: int):
         i - (d / delta) <= j AND j <= i IMPLIES 0 <= j AND p(j))
 END Time
```

B. PVS Isolette Theory

Below is the PVS Theory for the Isolette Specification:

```
% Isolette System Specification

Isolette[delta: posreal]: THEORY
 BEGIN
  % importing Time specification
  IMPORTING Time[delta]

  % variable
  i: VAR DTIME

  % types
  CURRENT_TEMPERATURE: TYPE = {n: real | n >= 68.0 AND n <= 105.0}
        CONTAINING 68.0

  DESIRED_LOW_TEMP: TYPE = {n: integer | n >= 97 AND n <= 99}
                CONTAINING 97

  DESIRED_HIGH_TEMP: TYPE = {n: integer | n >= 98 AND n <= 100}
                CONTAINING 98

  ALARM_LOW_TEMP: TYPE = {n: integer | n >= 93 AND n <= 98}
                CONTAINING 94

  ALARM_HIGH_TEMP: TYPE = {n: integer | n >= 99 AND n <= 103}
                CONTAINING 100

  STATUS: TYPE = {valid, invalid}

  SWITCH: TYPE = {on, off}

  HEAT_CONTROL: TYPE = {on, off}

  DISPLAYED_TEMPERATURE: TYPE =
        {n: integer | n = 0 OR n >= 68 AND n <= 105} CONTAINING 68
```

```
MESSAGES: TYPE =
{ok, cooling_down, warming_up, too_cool_alarm, too_hot_alarm,
 system_error}

ALARM: TYPE = {off, on}

MODES: TYPE = {off, init, normal, fail}

% monitored variables
m_tm: [DTIME -> CURRENT_TEMPERATURE]

m_dl: [DTIME -> DESIRED_LOW_TEMP]

m_dh: [DTIME -> DESIRED_HIGH_TEMP]

m_al: [DTIME -> ALARM_LOW_TEMP]

m_ah: [DTIME -> ALARM_HIGH_TEMP]

m_st: [DTIME -> STATUS]

m_sw: [DTIME -> SWITCH]

% controlled variables
c_hc: [DTIME -> HEAT_CONTROL]

c_td: [DTIME -> DISPLAYED_TEMPERATURE]

c_ms: [DTIME -> MESSAGES]

c_al: [DTIME -> ALARM]

c_md: [DTIME -> MODES]

alarm(i: DTIME): bool = c_al(i) = on
```

```
% FUNCTION TABLES %

% mode function table: c_md
mode_ft(i): bool =
    COND i = 0 -> c_md(i) = off,
         i > 0 ->
           COND m_sw(i) = off -> c_md(i) = off,
                NOT (m_sw(i) = off) ->
                  COND c_md(i - 1) = off -> c_md(i) = init,
                       c_md(i - 1) = init ->
                         COND m_st(i) = valid
                                    AND m_dl(i) <= m_tm(i)
                                    AND m_tm(i) <= m_dh(i)
                                    AND m_al(i) < m_dl(i)
                                    AND m_dl(i) < m_dh(i)
                                    AND m_dh(i) < m_ah(i)
                                        -> c_md(i) = normal,
                              NOT (m_st(i) = valid
                                    AND m_dl(i) <= m_tm(i)
                                    AND m_tm(i) <= m_dh(i)
                                    AND m_al(i) < m_dl(i)
                                    AND m_dl(i) < m_dh(i)
                                    AND m_dh(i) < m_ah(i))
                                        -> c_md(i) = c_md(i - 1)
                         ENDCOND,
                       c_md(i - 1) = normal ->
                         COND m_st(i) = invalid -> c_md(i) = fail,
                              NOT (m_st(i) = invalid) ->
                                c_md(i) = c_md(i - 1)
                         ENDCOND,
                       c_md(i - 1) = fail ->
                         COND NOT (m_st(i) = invalid) ->
                                c_md(i) = normal,
                              m_st(i) = invalid ->
                              c_md(i) = c_md(i - 1)
                         ENDCOND
                  ENDCOND
           ENDCOND
    ENDCOND
```

```
% heat control function table: c_hc
heat_control_ft(i): bool =
    COND i = 0 -> c_hc(i) = off,
         i > 0 ->
           COND m_sw(i) = off -> c_hc(i) = off,
                NOT (m_sw(i) = off) ->
                  COND m_dl(i) < m_dh(i) ->
                         COND m_tm(i) < m_dl(i) -> c_hc(i) = on,
                              m_dl(i) <= m_tm(i)
                              AND m_tm(i) <= m_dh(i)
                                      -> c_hc(i) = c_hc(i - 1),
                              m_tm(i) > m_dh(i)
                                      -> c_hc(i) = off
                         ENDCOND,
                       NOT (m_dl(i) < m_dh(i))
                              -> c_hc(i) = c_hc(i - 1)
                  ENDCOND
           ENDCOND
    ENDCOND

% temperature display function table: c_td
temperature_display_ft(i): bool =
    COND i = 0 -> c_td(i) = 0,
         i > 0 ->
           COND
              c_md(i - 1) = off OR c_md(i - 1) = fail
                      -> c_td(i) = 0,
              c_md(i - 1) = init OR c_md(i - 1) = normal
                      -> c_td(i) = floor(m_tm(i) + 0.5)
           ENDCOND
    ENDCOND
```

```
% alarm control function table: c_al
alarm_ft(i): bool =
    COND i = 0 -> c_al(i) = off,
         i > 0 ->
         COND c_md(i - 1) = off OR c_md(i - 1) = init ->
                c_al(i) = off,
              c_md(i - 1) = normal OR c_md(i - 1) = fail ->
              COND ((m_al(i) <= m_tm(i)
                    AND m_tm(i) < m_al(i) + 0.5)
                    OR
                    (m_ah(i) - 0.5 <= m_tm(i)
                    AND m_tm(i) <= m_ah(i))) ->
                    c_al(i) = c_al(i - 1),
                    NOT ((m_al(i) <= m_tm(i)
                         AND m_tm(i) < m_al(i) + 0.5)
                         OR
                         (m_ah(i) - 0.5 <= m_tm(i)
                         AND m_tm(i) <= m_ah(i))) ->
                         COND (m_tm(i) > m_ah(i)
                              OR m_tm(i) < m_al(i)
                              OR m_st(i) = invalid) ->
                              c_al(i) = on,
                              NOT (m_tm(i) > m_ah(i)
                                   OR
                                   m_tm(i) < m_al(i)
                                   OR
                                   m_st(i) = invalid) ->
                                   COND c_al(i - 1) = off ->
                                        c_al(i) = c_al(i - 1),
                                        NOT (c_al(i - 1) = off) ->
                                        COND
                                        held_for(alarm, 10)(i - 1)
                                        -> c_al(i) = off,
                                        NOT
                                        (held_for(alarm, 10)
                                        (i - 1))
                                        ->
                                        COND m_sw(i) = off ->
                                          c_al(i) = off,
                                          NOT (m_sw(i) = off) ->
                                          c_al(i) = on
```

```
                                                        ENDCOND
                                                     ENDCOND
                                                ENDCOND
                                         ENDCOND
                                  ENDCOND
                           ENDCOND
    ENDCOND

% displayed message functional table: c_ms
message_ft(i): bool =
    IF m_st(i) = invalid THEN c_ms(i) = system_error
    ELSIF m_tm(i) > m_ah(i) THEN c_ms(i) = too_hot_alarm
    ELSIF m_tm(i) < m_al(i) THEN c_ms(i) = too_cool_alarm
    ELSIF m_al(i) < m_tm(i) < m_dl(i) THEN c_ms(i) = warming_up
    ELSIF m_dh(i) < m_tm(i) < m_ah(i) THEN c_ms(i) = cooling_down
    ELSE c_ms(i) = ok
    ENDIF

% specification of all function tables
specification(i): bool =
         mode_ft(i) AND heat_control_ft(i)
         AND temperature_display_ft(i)
         AND alarm_ft(i) AND message_ft(i)

% INVARIANTS %

% inv1: All modes are in the set of { off, init, normal, fail }
invariant1(i): bool =
    c_md(i) = off OR c_md(i) = init OR c_md(i) = normal
    OR c_md(i) = fail

% inv2: All messages are in the set of { system_error,
% too_hot_alarm, too_cool_alarm, warming_up,
% cooling_down, ok }
invariant2(i): bool =
        c_ms(i) = system_error OR c_ms(i) = too_hot_alarm
        OR c_ms(i) = too_cool_alarm OR c_ms(i) = warming_up
        OR c_ms(i) = cooling_down OR c_ms(i) = ok

% inv3: The heat control cannot be on and off at the same time
invariant3(i): bool = NOT (c_hc(i) = on AND c_hc(i) = off)
```

```
% inv4: The alarm cannot be on and off at the same time
invariant4(i): bool = NOT (c_al(i) = on AND c_al(i) = off)

% all the invariants
invariant(i): bool =
              invariant1(i) AND invariant2(i) AND invariant3(i)
              AND invariant4(i)

% specficiation should satisfy the invariant
spec_inv_holds: CONJECTURE
              FORALL (i: DTIME):
              specification(i)
              IMPLIES
              invariant(i)

% USECASES %

% uc1: If the switch is on and the status is invalid
% then the message displayed should say "system_error"
usecase1: CONJECTURE
      m_sw(2) = on AND m_st(2) = invalid AND message_ft(2)
      IMPLIES
      c_ms(2) = system_error

% uc2: If the switch is on at time 1 and the current temperature
% is outside the temperature range at time 1 THEN
% the mode at time 0 is off and mode at time 1 is init
usecase2: CONJECTURE
       m_sw(1) = on AND m_tm(1) = 74 AND m_al(1) = 97
       AND m_ah(1) = 103 AND m_dl(1) = 98
       AND m_dh(1) = 100 AND m_st(1) = valid AND mode_ft(0)
       AND mode_ft(1)
       IMPLIES
       c_md(0) = off AND c_md(1) = init
```

```
  % uc3: If the switch is on and current temperature is
  % below the desired range the heat control should be on
  usecase3: CONJECTURE
         m_sw(2) = on AND m_tm(2) = 96 AND m_al(2) = 96
         AND m_ah(2) = 103 AND m_dl(2) = 98 AND m_dh(2) = 100
         AND heat_control_ft(2)
         IMPLIES
         c_hc(2) = on

  % uc4: If the mode is normal and the current temperature is below
  % the alarm temperature then the alarm should be on and the
  % displayed message should say "too_cool_alarm"
  usecase4: CONJECTURE
        c_md(2) = normal
        AND m_sw(3) = on
        AND m_tm(3) = 90
        AND m_al(3) = 93
        AND m_ah(3) = 103
        AND NOT (m_st(3) = invalid)
        AND alarm_ft(3)
        AND message_ft(3)
        AND mode_ft(3)
        IMPLIES
        c_al(3) = on
        AND c_ms(3) = too_cool_alarm
        AND c_md(3) = c_md(2)

  % uc5: If the mode is normal and the current temperature is 98,
  % the displayed temperature shown to the nurse should be 98
  usecase5: CONJECTURE
        c_md(2) = normal AND m_tm(3) = 98
        AND temperature_display_ft(3)
        IMPLIES
        c_td(3) = 98

END Isolette
```

C. Invariants

Below are the invariants in PVS logic used for the specification:

```
% specification of all function tables
specification(i): bool =
              mode_ft(i) AND heat_control_ft(i)
              AND temperature_display_ft(i)
              AND alarm_ft(i) AND message_ft(i)

% inv1: All modes are in the set of { off, init, normal, fail }
invariant1(i): bool =
              c_md(i) = off OR c_md(i) = init
              OR c_md(i) = normal OR c_md(i) = fail

% inv2: All messages are in the set of { system_error,
% too_hot_alarm, too_cool_alarm, warming_up, cooling_down, ok }
invariant2(i): bool =
              c_ms(i) = system_error OR c_ms(i) = too_hot_alarm
              OR c_ms(i) = too_cool_alarm OR c_ms(i) = warming_up
              OR c_ms(i) = cooling_down OR c_ms(i) = ok

% inv3: The heat control cannot be on and off at the same time
invariant3(i): bool = NOT (c_hc(i) = on AND c_hc(i) = off)

% inv4: The alarm cannot be on and off at the same time
invariant4(i): bool = NOT (c_al(i) = on AND c_al(i) = off)

% all the invariants
invariant(i): bool =
              invariant1(i) AND invariant2(i) AND invariant3(i)
              AND invariant4(i)

% specficiation should satisfy the invariant
spec_inv_holds: CONJECTURE
              FORALL (i: DTIME):
              specification(i)
              IMPLIES
              invariant(i)
```

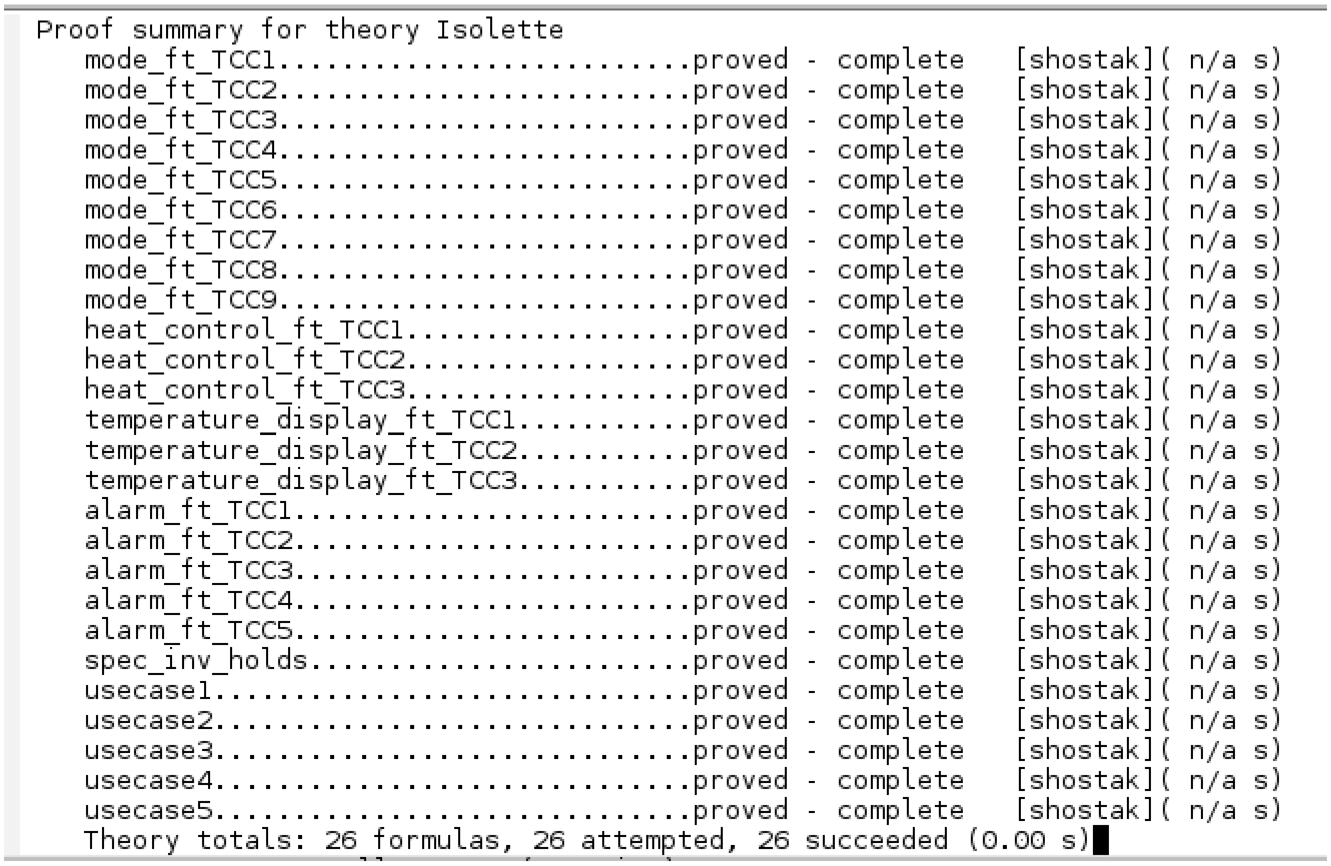
D. Additional Use Cases

Below are the additional use cases in PVS logic used to validate the specification:

```
% uc1: If the switch is on and the status is invalid
% then the message displayed should say "system_error"
usecase1: CONJECTURE
      m_sw(2) = on AND m_st(2) = invalid AND message_ft(2)
      IMPLIES
      c_ms(2) = system_error

% uc2: If the switch is on at time 1 and the current temperature
% is outside the temperature range at time 1 THEN
% the mode at time 0 is off and mode at time 1 is init
usecase2: CONJECTURE
       m_sw(1) = on AND m_tm(1) = 74 AND m_al(1) = 97
       AND m_ah(1) = 103 AND m_dl(1) = 98
       AND m_dh(1) = 100 AND m_st(1) = valid AND mode_ft(0)
       AND mode_ft(1)
       IMPLIES
       c_md(0) = off AND c_md(1) = init

% uc3: If the switch is on and current temperature is
% below the desired range the heat control should be on
usecase3: CONJECTURE
       m_sw(2) = on AND m_tm(2) = 96 AND m_al(2) = 96
       AND m_ah(2) = 103 AND m_dl(2) = 98 AND m_dh(2) = 100
       AND heat_control_ft(2)
       IMPLIES
       c_hc(2) = on

% uc5: If the mode is normal and the current temperature is 98,
% the displayed temperature shown to the nurse should be 98
usecase5: CONJECTURE
      c_md(2) = normal AND m_tm(3) = 98
      AND temperature_display_ft(3)
      IMPLIES
      c_td(3) = 98
```

E. PVS Proof Validation

Below is the proof of completeness and disjointness and validation of the requirements using PVS:



```
Proof summary for theory Isolette
   mode_ft_TCC1.........................proved - complete   [shostak]( n/a s)
   mode_ft_TCC2.........................proved - complete   [shostak]( n/a s)
   mode_ft_TCC3.........................proved - complete   [shostak]( n/a s)
   mode_ft_TCC4.........................proved - complete   [shostak]( n/a s)
   mode_ft_TCC5.........................proved - complete   [shostak]( n/a s)
   mode_ft_TCC6.........................proved - complete   [shostak]( n/a s)
   mode_ft_TCC7.........................proved - complete   [shostak]( n/a s)
   mode_ft_TCC8.........................proved - complete   [shostak]( n/a s)
   mode_ft_TCC9.........................proved - complete   [shostak]( n/a s)
   heat_control_ft_TCC1.................proved - complete   [shostak]( n/a s)
   heat_control_ft_TCC2.................proved - complete   [shostak]( n/a s)
   heat_control_ft_TCC3.................proved - complete   [shostak]( n/a s)
   temperature_display_ft_TCC1..........proved - complete   [shostak]( n/a s)
   temperature_display_ft_TCC2..........proved - complete   [shostak]( n/a s)
   temperature_display_ft_TCC3..........proved - complete   [shostak]( n/a s)
   alarm_ft_TCC1........................proved - complete   [shostak]( n/a s)
   alarm_ft_TCC2........................proved - complete   [shostak]( n/a s)
   alarm_ft_TCC3........................proved - complete   [shostak]( n/a s)
   alarm_ft_TCC4........................proved - complete   [shostak]( n/a s)
   alarm_ft_TCC5........................proved - complete   [shostak]( n/a s)
   spec_inv_holds.......................proved - complete   [shostak]( n/a s)
   usecase1.............................proved - complete   [shostak]( n/a s)
   usecase2.............................proved - complete   [shostak]( n/a s)
   usecase3.............................proved - complete   [shostak]( n/a s)
   usecase4.............................proved - complete   [shostak]( n/a s)
   usecase5.............................proved - complete   [shostak]( n/a s)
   Theory totals: 26 formulas, 26 attempted, 26 succeeded (0.00 s)
```

Figure 5: PVS Proof Validation