An Event-B Specification of

SquareRoot

This Event-B system is based on a model that appeared in the book: System Modelling & Design Using Event-B by Ken Robinson.

This project implements an integer square root algorithm. The algorithm performs a binary search of a value x such that $x^*x = input$, ie x will become the square root.

1	CONTEXT SquareRootDefinition
	1.1 SQRT
2	CONTEXT Theories
3	MACHINE SquareRoot
	3.1 input input_valid result_valid
	$3.2 \text{setInput}(v) \dots \dots \dots \dots \dots \dots \dots \dots \dots $
	3.3 SquareRoot
	3.4 getResult(out_result)
	3.5 high low \ldots
	3.6 setInput extends setInput
	3.7 SquareRoot refines SquareRoot
	3.8 Improve(h l)
	3.9
	3.10 ImproveLowerBound(m) refines Improve
	3.11 ImproveUpperBound(m) refines Improve
	3.13 ImproveLowerBound(m) refines ImproveLowerBound
	3.14 ImproveUpperBound(m) refines ImproveUpperBound
	3.15 mid
	3.16 setInput(v) refines setInput
	3.17 ImproveLowerBound refines ImproveLowerBound
	3.18 ImproveUpperBound refines ImproveUpperBound
4	REFINEMENT SquareRoot_R1_AddIncrementalImprovements 10
	4.1 high low
	4.2 setInput extends setInput
	4.3 SquareRoot refines SquareRoot
	4.4 Improve(h l)
۲	DEFINEMENT C D. A. D. W.A. I
5	REFINEMENT SquareRoot_R2_WithImproveLowerOrUpper 12 5.1 12
	5.2 ImproveLowerBound(m) refines Improve
	5.3 ImproveUpperBound(m) refines Improve
6	REFINEMENT SquareRoot_R3_AddDivisionToFindM 13
	6.1
	6.2 ImproveLowerBound(m) refines ImproveLowerBound
	6.3 ImproveUpperBound(m) refines ImproveUpperBound
7	REFINEMENT SquareRoot_R4_WithMiddleInVariable 14
•	7.1 mid
	7.2 setInput(v) refines setInput
	7.3 ImproveLowerBound refines ImproveLowerBound
	7.4 ImproveUpperBound refines ImproveUpperBound





This is the mathematical definition of the function SQRT.

EXTENDS Theories

CONSTANTS

1.1

 SQRT

AXIOMS

axm1: $SQRT \in \mathbb{N} \to \mathbb{N}$

 $\mathtt{axm2} \colon \forall m, n \cdot m \in \mathbb{N} \land n \in \mathbb{N} \Rightarrow (m = \mathrm{SQRT}(n) \Leftrightarrow m * m \leq n \land (m+1) * (m+1) > n)$

theorem thm1:

 $\forall n \cdot n \in \mathbb{N} \Rightarrow \operatorname{SQRT}(n) * \operatorname{SQRT}(n) \leq n \wedge (\operatorname{SQRT}(n) + 1) * (\operatorname{SQRT}(n) + 1) > n$

theorem thm2:

 $\forall n \cdot n \in \mathbb{N} \Rightarrow n = \text{SQRT}(n * n)$



Helpful theorems when proving the square root algorithm.

AXIOMS

```
\mathtt{axm1} \colon \forall n \cdot n \in \mathbb{N} \Rightarrow (\exists m \cdot m \in \mathbb{N} \land (n = 2 * m \lor n = 2 * m + 1))
```

Every natural number is either even or odd.

```
theorem thm2:
```

```
\forall n \cdot n \in \mathbb{N} \Rightarrow n < (n+1) * (n+1)
```

Every natural number is less than the square of its successor.

```
theorem thm3:
```

```
\forall m, n \cdot m \in \mathbb{N} \land n \in \mathbb{N} \land n > m \Rightarrow (m+n) \div 2 < n
```

The mean of any pair of unequal natural numbers is less than the larger of the pair.

```
theorem thm4:
```

```
\forall m, n \cdot m \in \mathbb{N} \land n \in \mathbb{N} \land n > m \Rightarrow (m+n) \div 2 \geq m
```

The mean of any pair of natural numbers is greater than or equal to the smaller of the pair.

We first define the sequence: setInput,SquareRoot,getResult. The value calculated is entirely desribed using the mathematical description.

3

${\tt SEES} \ {\tt SquareRootDefinition}$

VARIABLES

```
3.1
 input
                  The value to calculate the square root for.
                  True when a number has been supplied
 input\_valid
                  This is the calculated result
 result
 result\_valid
                  True when result = sqrt(num)
INVARIANTS
 inv_1:
             input \in \mathbb{N}
 inv_2:
             input\_valid \in BOOL
            result \in \mathbb{N}
 inv_3:
             result\_valid \in \mathsf{BOOL}
 inv_4:
             input\_valid = \text{TRUE} \land result\_valid = \text{TRUE} \Rightarrow result = \text{SQRT}(input)
 inv_5:
EVENT INITIALISATION
THEN
             input := 0
 act_1:
 act_2:
             input\_valid := FALSE
 act_3:
             result := 0
 act_4:
             result\_valid := FALSE
END
                                                                                                                       3.2
EVENT setInput
ANY
 v
WHERE
 grd_1:
            v \in \mathbb{N}
            input\_valid = \mathrm{FALSE}
 grd_2:
             result\_valid = FALSE
 grd_3:
THEN
 act_1:
             input := v
 act_2:
             input\_valid := TRUE
END
                                                                                                                       3.3
EVENT SquareRoot
WHERE
             input \ valid = TRUE
 grd_1:
             result\_valid = FALSE
 grd_2:
THEN
 act_1:
             result := SQRT(input)
 An alternative is to specify a non-deterministic assignment:
 \operatorname{sqrt}:|(\operatorname{sqrt}' \in \mathbb{N} \wedge \operatorname{sqrt}' * \operatorname{sqrt}' \leq \operatorname{num} \wedge \operatorname{num} < (\operatorname{sqrt}' + 1) * (\operatorname{sqrt}' + 1))|
            result\_valid := TRUE
 act_2:
END
```

Add a lower and upper bound of the correct answer. Each improvement will narrow the bounds until, eventually, we have a single number as the answer.

REFINES SquareRoot SEES SquareRootDefinition

VARIABLES 3.5

low When improving we have a lower bound of the correct answer.

high And an upper bound of the correct answer.

INVARIANTS

```
low \in \mathbb{N}
inv1_1:
            high \in \mathbb{N}
inv1_2:
inv1_3:
            low + 1 \le high
                                The span is 1 or more.
            low * low \le input
inv1_4:
inv1_5:
            input < high * high
inv16:
            low < high
theorem
            thm1_1:
            low + 1 \neq high \Rightarrow low < (low + high) \div 2
theorem
            (low + high) \div 2 < high
theorem
            thm1_3:
            high - low > 0
theorem
            thm1_4:
            high - low \in \mathbb{N}
```

VARIANTS

high-low The variant guarantees that the span must decrease in each step. Eventually the span will be exactly one and low is the sought number .

```
EVENT INITIALISATION EXTENDS INITIALISATION THEN  \begin{array}{ll} \text{init1\_1:} & low := 0 \\ \text{init1\_2:} & high := 1 \end{array}  END
```

EVENT setInput

3.6

When num is set, we specify that a lower and upper bound is magically selected in some way, that enables the improvement step to work. We do not yet know, how this magical selection is performed. Here we merely state that low and high are selected so that the predicates become true.

```
EXTENDS setInput
```

```
THEN
```

```
act1_1: low : | low' \in \mathbb{N} \wedge low' * low' \leq v
```

```
3.7
EVENT SquareRoot
We detect the terminating case, when low+1=high, then low = \sqrt{num}
REFINES SquareRoot
WHERE
 grd1_1:
           input\_valid = TRUE
 grd1_2:
           result\_valid = FALSE
          low + 1 = high We have found the best value.
 grd1_3:
 thm1_1:
          low * low \le input
 thm1_2:
          input < high * high
THEN
 act1_1:
           result := low
 act1 2:
           result \ valid := TRUE
END
CONVERGENT EVENT Improve
                                                                                               3.8
The improve event magically selects an l and h, that are an improvement on the existing bounds.
We do not know how this is done, but we specify the result here.
ANY
 l
 h
WHERE
 grd1_1: l \in \mathbb{N}
 grd1_2: h \in \mathbb{N}
 grd1_3:
          low + 1 \neq high
                                We can still improve.
 grd1 4:
          low < l
                                The new lower bound is higher.
          l*l \leq input
                                But still not above the input.
 grd1_5:
          h \leq high
                                The new higher bound is lower.
 grd1_6:
 grd1_7:
          input < h * h
                                But still above the input.
 grd1_8:
           l+1 \leq h
                                The new span is 1 or more.
           h - l < high - low
                               The new bound is an improvement.
 grd1_9:
THEN
           low := l
 act1_1:
           high := h
 act1_2:
{\tt REFINEMENT~SquareRoot\_R2\_WithImproveLowerOrUpper}
                                                                         20 a
We now split the improve event into improveLowerBound and improveUpperBound.
REFINES SquareRoot_R1_AddIncrementalImprovements
SEES SquareRootDefinition
VARIABLES
                                                                                               3.9
                                                                                               3.10
EVENT ImproveLowerBound
REFINES Improve
ANY
 m
```

act1_2: $high: |high' \in \mathbb{N} \land v < high' * high'$

```
WHERE
 grd2_1: low + 1 \neq high
 grd2_2:
          m \in \mathbb{N}
 grd2_3:
          low < m \land m < high
          m*m \le input The new m is a better lower bound.
 grd2_4:
WITH
 1: l=m
                Therefore we pick the new m as the lower bound.
     h = high
                The high bound stays the same.
THEN
 act2_1: low := m
END
                                                                                           3.11
EVENT ImproveUpperBound
REFINES Improve
ANY
 m
WHERE
 grd2_1: low + 1 \neq high
          m \in \mathbb{N}
 grd2_2:
 grd2_3:
          low < m \land m < high
 grd2_4: m*m > input The new m is a better upper bound.
WITH
 1:
     l = low
               The low bound stays the same.
 h:
     h = m
               Therefore we picke the new m as the higher bound.
THEN
 act2_1: high := m
REFINEMENT SquareRoot_R3_AddDivisionToFindM
                                                                    6 a
We now pick a suitable middle value by dividing by 2.
REFINES SquareRoot_R2_WithImproveLowerOrUpper
SEES SquareRootDefinition
                                                                                           3.12
VARIABLES
EVENT ImproveLowerBound
                                                                                           3.13
REFINES ImproveLowerBound
ANY
 m
WHERE
 grd3_1:
         low + 1 \neq high
          m = (low + high) \div 2
 grd3_2:
 grd3_3:
          m * m \le input m is a better lower bound!
THEN
 act3_1:
          low := m
END
```

```
3.14
EVENT ImproveUpperBound
REFINES ImproveUpperBound
ANY
 m
WHERE
 grd3_1: low + 1 \neq high
 grd3_2:
          m = (low + high) \div 2
 grd3_3:
          m*m > input
THEN
           high := m m is a better upper bound!
 act3<sub>1</sub>:
END
{\tt REFINEMENT~SquareRoot\_R4\_WithMiddleInVariable}
                                                                   18a 2i
We now store the middle value in a variable.
{\tt REFINES}\ SquareRoot\_R3\_AddDivisionToFindM
SEES SquareRootDefinition
VARIABLES
                                                                                              3.15
 mid Track each middle value to find next bound.,
INVARIANTS
 inv1:
         mid = (low + high) \div 2
 inv2:
         mid \in \mathbb{N}
EVENT INITIALISATION
THEN
 init4_1:
            input := 0
 init4_2:
            input\_valid := FALSE
 init4_3:
            low := 0
 init4_4: mid := 0
 init4_5:
            high := 1
 init4_6:
            result := 0
 init4_7:
            result\_valid := FALSE
END
                                                                                              3.16
EVENT setInput
REFINES setInput
ANY
 v
WHERE
          v \in \mathbb{N}
 grd4_1:
 grd4_2:
          input\_valid = FALSE
 grd4_3:
          result\_valid = FALSE
THEN
 init4_0:
            input := v
 init4_1:
            low := 0
            high := v + 1
 init4_2:
 init4_3:
            mid := (v+1) \div 2
 init4_4:
            input\_valid := TRUE
```

```
3.17
EVENT ImproveLowerBound
{\tt REFINES} \ {\tt ImproveLowerBound}
WHERE
 grd4_1: low + 1 \neq high
 grd4_2: mid*mid \leq input
WITH
     m = mid mid is a better value for low
 m:
THEN
 act4_1: low := mid
 act4_2: mid := (mid + high) \div 2
END
                                                                                          3.18
EVENT ImproveUpperBound
REFINES ImproveUpperBound
WHERE
 grd4_1: low + 1 \neq high
 grd4_2: mid*mid>input
WITH
     m = mid mid is a better value for high
THEN
 act4_1: high := mid
 act4_2: mid := (low + mid) \div 2
END
```

29 a



4

Add a lower and upper bound of the correct answer. Each improvement will narrow the bounds until, eventually, we have a single number as the answer.

REFINES SquareRoot

SEES SquareRootDefinition

VARIABLES

4.1

low When improving we have a lower bound of the correct answer.high And an upper bound of the correct answer.

INVARIANTS

```
inv1_1:
            low \in \mathbb{N}
inv1_2:
            high \in \mathbb{N}
inv1_3:
            low + 1 \le high
                               The span is 1 or more.
inv1_4:
            low * low \le input
            input < high * high
inv1_5:
            low < high
inv1_6:
theorem
            thm1_1:
            low + 1 \neq high \Rightarrow low < (low + high) \div 2
theorem
            thm1_2:
            (low + high) \div 2 < high
            thm1_3:
theorem
            high - low > 0
            thm1_4:
theorem
            high - low \in \mathbb{N}
```

VARIANTS

high - low The variant guarantees that the span must decrease in each step. Eventually the span will be exactly one and low is the sought number.

EVENT INITIALISATION

EXTENDS INITIALISATION

THEN

```
init1_1: low := 0
init1_2: high := 1
END
```

EVENT setInput

4.2

When num is set, we specify that a lower and upper bound is magically selected in some way, that enables the improvement step to work. We do not yet know, how this magical selection is performed. Here we merely state that low and high are selected so that the predicates become true.

EXTENDS setInput

THEN

```
\begin{array}{ll} \texttt{act1\_1:} & low: |\ low' \in \mathbb{N} \land low' * low' \leq v \\ \texttt{act1\_2:} & high: |\ high' \in \mathbb{N} \land v < high' * high' \\ \texttt{END} \end{array}
```

EVENT SquareRoot

4.3

We detect the terminating case, when low+1=high, then $low = \sqrt{num}$

REFINES SquareRoot

WHERE

```
input\_valid = TRUE
 grd1_1:
 grd1_2:
            result\_valid = FALSE
            low + 1 = high
                             We have found the best value.
 grd1_3:
            low*low \leq input
 thm1_1:
 thm1_2:
            input < high * high
THEN
 act1_1:
            \mathit{result} := \mathit{low}
 act1_2:
            result \ valid := TRUE
END
```

CONVERGENT EVENT Improve

4.4

The improve event magically selects an l and h, that are an improvement on the existing bounds. We do not know how this is done, but we specify the result here.

```
ANY
 l
 h
WHERE
 grd1_1: l \in \mathbb{N}
           h \in \mathbb{N}
 grd1_2:
 grd1_3:
           low + 1 \neq high
                                  We can still improve.
           low \leq l
 grd1_4:
                                  The new lower bound is higher.
 grd1_5:
           l*l \leq input
                                  But still not above the input.
 grd1_6:
           h \leq high
                                  The new higher bound is lower.
 grd1_7:
           input < h * h
                                  But still above the input.
 grd1_8:
           l+1 \leq h
                                  The new span is 1 or more.
 grd1_9:
           h - l < high - low
                                 The new bound is an improvement.
THEN
 act1_1:
            low := l
 act1_2:
            high := h
END
```

We now split the improve event into improveLowerBound and improveUpperBound.

 $\label{lem:refines} \begin{tabular}{ll} REFINES & SquareRoot_R1_AddIncrementalImprovements \\ SEES & SquareRootDefinition \\ \end{tabular}$

```
VARIABLES
                                                                                           5.1
EVENT ImproveLowerBound
                                                                                           5.2
REFINES Improve
ANY
 m
WHERE
 grd2_1: low + 1 \neq high
 grd2_2:
          m \in \mathbb{N}
 grd2_3:
          low < m \land m < high
 grd2_4: m*m \le input The new m is a better lower bound.
WITH
                Therefore we pick the new m as the lower bound.
 1: l=m
     h = high
                The high bound stays the same.
THEN
 act2_1: low := m
END
EVENT ImproveUpperBound
                                                                                           5.3
REFINES Improve
ANY
 m
WHERE
 grd2_1: low + 1 \neq high
 grd2_2: m \in \mathbb{N}
 grd2_3: low < m \land m < high
 grd2_4: m*m > input The new m is a better upper bound.
WITH
 1: l = low
               The low bound stays the same.
               Therefore we picke the new m as the higher bound.
     h = m
THEN
 act2_1: high := m
END
```

We now pick a suitable middle value by dividing by 2.

 $\label{lem:refines} \begin{tabular}{ll} REFINES & SquareRoot_R2_WithImproveLowerOrUpper \\ SEES & SquareRootDefinition \\ \end{tabular}$

act3_1: high := m m is a better upper bound!

```
VARIABLES
                                                                                       6.1
                                                                                       6.2
EVENT ImproveLowerBound
REFINES ImproveLowerBound
ANY
 m
WHERE
 grd3_1: low + 1 \neq high
         m = (low + high) \div 2
 grd3_2:
 grd3_3:
         m*m \leq input m is a better lower bound!
THEN
 act3_1: low := m
END
                                                                                       6.3
EVENT ImproveUpperBound
REFINES ImproveUpperBound
ANY
 m
WHERE
 grd3_1: low + 1 \neq high
 grd3_2:
         m = (low + high) \div 2
 grd3_3:
          m*m > input
THEN
```



We now store the middle value in a variable.

REFINES SquareRoot_R3_AddDivisionToFindM SEES SquareRootDefinition

```
VARIABLES
                                                                                     7.1
```

mid Track each middle value to find next bound.,

```
INVARIANTS
```

```
inv1: mid = (low + high) \div 2
inv2:
         mid \in \mathbb{N}
```

```
EVENT INITIALISATION
```

```
THEN
```

```
init4_1:
           input := 0
           input \ valid := FALSE
 init4_2:
 init4_3: low := 0
 init4_4: mid := 0
 init4_5: high := 1
 init4_6: result := 0
 init4_7: result\_valid := FALSE
END
```

EVENT setInput

REFINES setInput

ANY

v

WHERE

```
grd4_1: v \in \mathbb{N}
 grd4_2: input\_valid = FALSE
 grd4_3: result\_valid = FALSE
THEN
 init4_0: input := v
 init4_1: low := 0
 init4_2: high := v + 1
```

END

init4_3: init4_4:

EVENT ImproveLowerBound

REFINES ImproveLowerBound

WHERE

```
grd4_1: low + 1 \neq high
 grd4_2: mid * mid \leq input
WITH
      m = mid mid is a better value for low
 m:
THEN
```

 $mid := (v+1) \div 2$

 $input \ valid := TRUE$

```
act4_1: low := mid
```

```
act4_2: mid := (mid + high) \div 2
```

7.2

14

7.3

END

7.4

getResult, 5

high, 5, 10

Improve, 6, 7, 11, 12 ImproveLowerBound, 6, 7, 9, 12–14 ImproveUpperBound, 7–9, 12, 13, 15 INITIALISATION, 4, 5, 8, 10, 14 input, 4 input_valid, 4

low, 5, 10

mid, 8, 14

 $\begin{array}{c} \text{result, 4} \\ \text{result_valid, 4} \end{array}$

setInput, 4, 5, 8, 10, 14 SquareRoot, 4–6, 10 SquareRoot_R1_AddIncrementalImprovements, 5, 6, 10, 12 SquareRoot_R2_WithImproveLowerOrUpper, 6, 7, 12, 13 SquareRoot_R3_AddDivisionToFindM, 7, 8, 13,

 $SquareRoot_R4_WithMiddleInVariable,~8,~14\\ SquareRootDefinition,~2,~4–8,~10,~12–14$

Theories, 2, 3