

# 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  $x*x = \text{input}$ , ie  $x$  will become the square root.

---

<b>1</b>	<b>CONTEXT SquareRootDefinition</b>	<b>2</b>
1.1	<b>SQRT</b> . . . . .	2
<b>2</b>	<b>CONTEXT Theories</b>	<b>3</b>
<b>3</b>	<b>MACHINE SquareRoot</b>	<b>4</b>
3.1	<i>input input_valid result result_valid</i> . . . . .	4
3.2	<b>setInput</b> ( $v$ ) . . . . .	4
3.3	<b>SquareRoot</b> . . . . .	4
3.4	<b>getResult</b> ( $out\_result$ ) . . . . .	5
<b>4</b>	<b>REFINEMENT SquareRoot_R1_AddIncrementalImprovements</b>	<b>6</b>
4.1	<b>u</b> <i>high low</i> . . . . .	6
4.2	<b>setInput</b> <b>extends</b> <b>setInput</b> . . . . .	6
4.3	<b>SquareRoot</b> <b>refines</b> <b>SquareRoot</b> . . . . .	7
4.4	<b>Improve</b> ( $h\ l$ ) . . . . .	7
<b>5</b>	<b>REFINEMENT SquareRoot_R2_WithImproveLowerOrUpper</b>	<b>8</b>
5.1	<b>u</b> . . . . .	8
5.2	<b>ImproveLowerBound</b> ( $m$ ) <b>refines</b> <b>Improve</b> . . . . .	8
5.3	<b>ImproveUpperBound</b> ( $m$ ) <b>refines</b> <b>Improve</b> . . . . .	8
<b>6</b>	<b>REFINEMENT SquareRoot_R3_AddDivisionToFindM</b>	<b>9</b>
6.1	. . . . .	9
6.2	<b>ImproveLowerBound</b> ( $m$ ) <b>refines</b> <b>ImproveLowerBound</b> . . . . .	9
6.3	<b>ImproveUpperBound</b> ( $m$ ) <b>refines</b> <b>ImproveUpperBound</b> . . . . .	9
<b>7</b>	<b>REFINEMENT SquareRoot_R4_WithMiddleInVariable</b>	<b>10</b>
7.1	<i>mid</i> . . . . .	10
7.2	<b>setInput</b> ( $v$ ) <b>refines</b> <b>setInput</b> . . . . .	10
7.3	<b>ImproveLowerBound</b> <b>refines</b> <b>ImproveLowerBound</b> . . . . .	10
7.4	<b>ImproveUpperBound</b> <b>refines</b> <b>ImproveUpperBound</b> . . . . .	11

---

This is the mathematical definition of the function Sqrt.

EXTENDS Theories

CONSTANTS

1.1

Sqrt

AXIOMS

axm1:  $\text{Sqrt} \in \mathbb{N} \rightarrow \mathbb{N}$

axm2:  $\forall m, n. m \in \mathbb{N} \wedge n \in \mathbb{N} \Rightarrow (m = \text{Sqrt}(n) \Leftrightarrow m * m \leq n \wedge (m + 1) * (m + 1) > n)$

THEOREM

thm1:  $\forall n. n \in \mathbb{N} \Rightarrow \text{Sqrt}(n) * \text{Sqrt}(n) \leq n \wedge (\text{Sqrt}(n) + 1) * (\text{Sqrt}(n) + 1) > n$

thm2:  $\forall n. n \in \mathbb{N} \Rightarrow n = \text{Sqrt}(n * n)$

END

---

Helpful theorems when proving the square root algorithm.

## AXIOMS

**axm1:**  $\forall n \cdot n \in \mathbb{N} \Rightarrow (\exists m \cdot m \in \mathbb{N} \wedge (n = 2 * m \vee n = 2 * m + 1))$  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.

**thm3:**  $\forall m, n \cdot m \in \mathbb{N} \wedge n \in \mathbb{N} \wedge n > m \Rightarrow (m + n) \div 2 < n$  The mean of any pair of unequal natural numbers is less than the larger of the pair.

**thm4:**  $\forall m, n \cdot m \in \mathbb{N} \wedge n \in \mathbb{N} \wedge n > m \Rightarrow (m + n) \div 2 \geq m$   
The mean of any pair of natural numbers is greater than or equal to the smaller of the pair.

END

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

### SEES SquareRootDefinition

#### VARIABLES

3.1

*input*      The value to calculate the square root for.  
*input\_valid*   True when a number has been supplied  
*result*      This is the calculated result  
*result\_valid*   True when  $result = \text{sqrt}(num)$

#### INVARIANTS

*inv\_1:*    *input*  $\in \mathbb{N}$   
*inv\_2:*    *input\_valid*  $\in \text{BOOL}$   
*inv\_3:*    *result*  $\in \mathbb{N}$   
*inv\_4:*    *result\_valid*  $\in \text{BOOL}$   
*inv\_5:*    *input\_valid* = **TRUE**  $\wedge$  *result\_valid* = **TRUE**  $\Rightarrow$  *result* = **SQRT**(*input*)

#### EVENT INITIALISATION

##### THEN

*act\_1:*    *input* := 0  
*act\_2:*    *input\_valid* := **FALSE**  
*act\_3:*    *result* := 0  
*act\_4:*    *result\_valid* := **FALSE**

##### END

#### EVENT setInput

3.2

##### ANY

*v*

##### WHERE

*grd\_1:*    *v*  $\in \mathbb{N}$   
*grd\_2:*    *input\_valid* = **FALSE**  
*grd\_3:*    *result\_valid* = **FALSE**

##### THEN

*act\_1:*    *input* := *v*  
*act\_2:*    *input\_valid* := **TRUE**

##### END

#### EVENT SquareRoot

3.3

##### WHERE

*grd\_1:*    *input\_valid* = **TRUE**  
*grd\_2:*    *result\_valid* = **FALSE**

##### THEN

*act\_1:*    *result* := **SQRT**(*input*)  
             An alternative is to specify a non-deterministic assignment:  
              $\text{sqrt} :| (\text{sqrt}' \in \mathbb{N} \wedge \text{sqrt}' * \text{sqrt}' \leq \text{num} \wedge \text{num} < (\text{sqrt}' + 1) * (\text{sqrt}' + 1))$   
*act\_2:*    *result\_valid* := **TRUE**

##### END

```
EVENT getResult
```

3.4

```
ANY
```

```
    out_result
```

```
WHERE
```

```
    grd.1:  result_valid = TRUE
```

```
    grd.2:  out_result = result
```

```
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.

u Improve/vrt/VAR

REFINES `SquareRoot`

SEES `SquareRootDefinition`

#### VARIABLES

*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:`  $low + 1 \leq high$   
`inv1_4:`  $low * low \leq input$   
`inv1_5:`  $input < high * high$   
`inv1_6:`  $low < high$   
theorem `thm1_1:`  
 $low + 1 \neq high \Rightarrow low < (low + high) \div 2$   
theorem `thm1_2:`  
 $(low + high) \div 2 < high$   
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 zero and a number has been found.

#### EVENT `INITIALISATION`

#### EXTENDS `INITIALISATION`

#### THEN

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

#### END

#### EVENT `setInput`

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 performed.

#### EXTENDS `setInput`

#### THEN

`act1_1:`  $low :| low' \in \mathbb{N} \wedge low' * low' \leq v$   
`act1_2:`  $high :| high' \in \mathbb{N} \wedge v < high' * high'$

#### END

**EVENT SquareRoot**

4.3

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

**REFINES SquareRoot****WHERE**

`grd1_1:`  $input\_valid = \text{TRUE}$   
`grd1_2:`  $result\_valid = \text{FALSE}$   
`grd1_3:`  $low + 1 = high$   
`thm1_1:`  $low * low \leq input$   
`thm1_2:`  $input < high * high$

**THEN**

`act1_1:`  $result := low$   
`act1_2:`  $result\_valid := \text{TRUE}$

**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:`  $low + 1 \neq high$   
`grd1_2:`  $l \in \mathbb{N} \wedge low \leq l \wedge l * l \leq input$   
`grd1_3:`  $h \in \mathbb{N} \wedge h \leq high \wedge input < h * h$   
`grd1_4:`  $l + 1 \leq h$   
`grd1_5:`  $h - 1 < high - low$

**THEN**

`act1_1:`  $low := l$   
`act1_2:`  $high := h$

**END**

We now split the improve event into `improveLowerBound` and `improveUpperBound`.

- u `ImproveLowerBound/grd1_5/GRD`
- u `ImproveUpperBound/grd1_5/GRD`

REFINES `SquareRoot_R1_AddIncrementalImprovements`  
 SEES `SquareRootDefinition`

VARIABLES

5.1

EVENT `ImproveLowerBound`  
 REFINES `Improve`  
 ANY

5.2

$m$

WHERE

`grd2_1:`  $low + 1 \neq high$   
`grd2_2:`  $m \in \mathbb{N}$   
`grd2_3:`  $low < m \wedge m < high$   
`grd2_4:`  $m * m \leq input$   $m$  is a better lower bound!

WITH

`l:`  $l = m$   
`h:`  $h = high$

THEN

`act2_1:`  $low := m$

END

EVENT `ImproveUpperBound`  
 REFINES `Improve`  
 ANY

5.3

$m$

WHERE

`grd2_1:`  $low + 1 \neq high$   
`grd2_2:`  $m \in \mathbb{N}$   
`grd2_3:`  $low < m \wedge m < high$   
`grd2_4:`  $m * m > input$   $m$  is a better upper bound!

WITH

`l:`  $l = low$   
`h:`  $h = m$

THEN

`act2_1:`  $high := m$

END



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

REFINES `SquareRoot_R2_WithImproveLowerOrUpper`  
SEES `SquareRootDefinition`

VARIABLES

6.1

EVENT `ImproveLowerBound`  
REFINES `ImproveLowerBound`  
ANY

6.2

$m$

WHERE

`grd3_1:`  $low + 1 \neq high$   
`grd3_2:`  $m = (low + high) \div 2$   
`grd3_3:`  $m * m \leq input$   $m$  is a better lower bound!

THEN

`act3_1:`  $low := m$

END

EVENT `ImproveUpperBound`  
REFINES `ImproveUpperBound`  
ANY

6.3

$m$

WHERE

`grd3_1:`  $low + 1 \neq high$   
`grd3_2:`  $m = (low + high) \div 2$   
`grd3_3:`  $m * m > input$

THEN

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

END

We now store the middle value in a variable.

REFINES `SquareRoot_R3_AddDivisionToFindM`  
 SEES `SquareRootDefinition`

VARIABLES

*mid* Track each middle value to find next bound.,

INVARIANTS

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

7.1

EVENT `INITIALISATION`

THEN

*init4\_1*:  $input := 0$   
*init4\_2*:  $input\_valid := \text{FALSE}$   
*init4\_3*:  $low := 0$   
*init4\_4*:  $mid := 0$   
*init4\_5*:  $high := 1$   
*init4\_6*:  $result := 0$   
*init4\_7*:  $result\_valid := \text{FALSE}$

END

EVENT `setInput`

REFINES `setInput`

ANY

*v*

WHERE

*grd4\_1*:  $v \in \mathbb{N}$   
*grd4\_2*:  $input\_valid = \text{FALSE}$   
*grd4\_3*:  $result\_valid = \text{FALSE}$

THEN

*init4\_0*:  $input := v$   
*init4\_1*:  $low := 0$   
*init4\_2*:  $high := v + 1$   
*init4\_3*:  $mid := (v + 1) \div 2$   
*init4\_4*:  $input\_valid := \text{TRUE}$

END

7.2

EVENT `ImproveLowerBound`

REFINES `ImproveLowerBound`

WHERE

*grd4\_1*:  $low + 1 \neq high$   
*grd4\_2*:  $mid * mid \leq input$

WITH

*m*:  $m = mid$   $mid$  is a better value for  $low$

THEN

7.3

```

act4_1:   $low := mid$ 
act4_2:   $mid := (mid + high) \div 2$ 
END

```

```

EVENT ImproveUpperBound
REFINES ImproveUpperBound
WHERE

```

7.4

```

grd4_1:   $low + 1 \neq high$ 
grd4_2:   $mid * mid > input$ 
WITH
m:   $m = mid$    $mid$  is a better value for high
THEN
act4_1:   $high := mid$ 
act4_2:   $mid := (low + mid) \div 2$ 
END

```

getResult, 5

high, 6

Improve, 7, 8

ImproveLowerBound, 8–10

ImproveUpperBound, 8, 9, 11

INITIALISATION, 4, 6, 10

input, 4

input\_valid, 4

low, 6

mid, 10

result, 4

result\_valid, 4

setInput, 4, 6, 10

SquareRoot, 4, 6, 7

SquareRoot\_R1\_AddIncrementalImprovements,  
6, 8

SquareRoot\_R2\_WithImproveLowerOrUpper, 8,  
9

SquareRoot\_R3\_AddDivisionToFindM, 9, 10

SquareRoot\_R4\_WithMiddleInVariable, 10

SquareRootDefinition, 2, 4, 6, 8–10

Theories, 2, 3