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.

CONTEXT SquareRootDefinition 4 a 1 i 1

This is the mathematical definition of the function SQRT.

EXTENDS Theories

CONSTANTS 1.1

SQRT

AXIOMS

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

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

theorem thm1:

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

theorem thm2:

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

END

Helpful theorems when proving the square root algorithm.

AXIOMS

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

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.

END

MACHINE SquareRoot 9a [3]

We first define the sequence: setInput,SquareRoot,getResult. The value calculated is entirely desribed 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 = sqrt(num)

INVARIANTS

inv_1: $input \in \mathbb{N}$
inv_2: $input\_valid \in \mathrm{BOOL}$
inv_3: $result \in \mathbb{N}$
inv_4: $result\_valid \in \mathrm{BOOL}$
inv_5: $input\_valid = \mathrm{TRUE} \land result\_valid = \mathrm{TRUE} \Rightarrow result = \mathrm{SQRT}(input)$

EVENT INITIALISATION
THEN

act_1: $input := 0$
act_2: $input\_valid := \mathrm{FALSE}$
act_3: $result := 0$
act_4: $result\_valid := \mathrm{FALSE}$

END

EVENT setInput [3.2]
ANY

v

WHERE

grd_1: $v \in \mathbb{N}$
grd_2: $input\_valid = \mathrm{FALSE}$
grd_3: $result\_valid = \mathrm{FALSE}$

THEN

act_1: $input := v$
act_2: $input\_valid := \mathrm{TRUE}$

END

EVENT SquareRoot [3.3]
WHERE

grd_1: $input\_valid = \mathrm{TRUE}$
grd_2: $result\_valid = \mathrm{FALSE}$

THEN

act_1: $result := \mathrm{SQRT}(input)$

An alternative is to specify a non-deterministic assignment:

sqrt :| (sqrt' $\in \mathbb{N} \land$ sqrt' $*$ sqrt' $\leq$ num $\land$ num $<$ (sqrt' + 1) $*$ (sqrt' + 1))

act_2: $result\_valid := \mathrm{TRUE}$

END

EVENT getResult $\boxed{3.4}$

ANY

out_result

WHERE

grd_1: $result\_valid = \mathrm{TRUE}$

grd_2: $out\_result = result$

END

REFINEMENT SquareRoot_R1_AddIncrementalImprovements 29 a 2 i [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: $low + 1 \leq high$ The span is 1 or more.
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 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

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

END

EVENT SquareRoot [4.3]
We detect the terminating case, when low+1=high, then $low = \sqrt{num}$

REFINES SquareRoot

WHERE

grd1_1:	$input\_valid = \mathrm{TRUE}$	
grd1_2:	$result\_valid = \mathrm{FALSE}$	
grd1_3:	$low + 1 = high$	We have found the best value.
thm1_1:	$low * low \leq input$	
thm1_2:	$input < high * high$	

THEN

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

END

CONVERGENT EVENT Improve $\boxed{4.4}$

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:	$l \in \mathbb{N}$	
grd1_2:	$h \in \mathbb{N}$	
grd1_3:	$low + 1 \neq high$	We can still improve.
grd1_4:	$low \leq l$	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

REFINEMENT SquareRoot_R2_WithImproveLowerOrUpper 20 a [5]

We now split the improve event into improveLowerBound and improveUpperBound.

REFINES SquareRoot_R1_AddIncrementalImprovements
SEES SquareRootDefinition

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: $low < m \wedge m < high$
grd2_4: $m * m \leq input$ The new m is a better lower bound.

WITH

l: $l = m$ Therefore we pick the new m as the lower bound.
h: $h = high$ 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: $low < m \wedge m < high$
grd2_4: $m * m > input$ The new m is a better upper bound.

WITH

l: $l = low$ The low bound stays the same.
h: $h = m$ Therefore we picke the new m as the higher bound.

THEN

act2_1: $high := m$

END

REFINEMENT SquareRoot_R3_AddDivisionToFindM 6 a [6]

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

REFINES SquareRoot_R2_WithImproveLowerOrUpper
SEES SquareRootDefinition

VARIABLES [6.1]

EVENT ImproveLowerBound [6.2]
REFINES ImproveLowerBound
ANY

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 [6.3]
REFINES ImproveUpperBound
ANY

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

REFINEMENT SquareRoot_R4_WithMiddleInVariable 18 a 2 i [7]

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: $input := 0$
init4_2: $input\_valid := \mathrm{FALSE}$
init4_3: $low := 0$
init4_4: $mid := 0$
init4_5: $high := 1$
init4_6: $result := 0$
init4_7: $result\_valid := \mathrm{FALSE}$

END

EVENT setInput [7.2]
REFINES setInput
ANY

v

WHERE

grd4_1: $v \in \mathbb{N}$
grd4_2: $input\_valid = \mathrm{FALSE}$
grd4_3: $result\_valid = \mathrm{FALSE}$

THEN

init4_0: $input := v$
init4_1: $low := 0$
init4_2: $high := v + 1$
init4_3: $mid := (v + 1) \div 2$
init4_4: $input\_valid := \mathrm{TRUE}$

END

EVENT ImproveLowerBound [7.3]
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

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

`END`

`EVENT` ImproveUpperBound $\boxed{7.4}$
`REFINES` ImproveUpperBound
`WHERE`

`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`