

System Modelling and Design

Square Root

Revision: 3.0, March 16, 2010

Ken Robinson

March 16, 2010

©Ken Robinson 2005

[mailto::k.robinson@unsw.edu.au](mailto:k.robinson@unsw.edu.au)

Contents

1	What is this lecture about?	1
2	The Problem	2
2.1	Contexts	2
2.2	Substitution	3
2.3	The SquareRoot machine	3
2.4	SquareRoot event	4
3	First refinement	4
3.1	Improving low and high	6
4	Second refinement	6
5	The third refinement	8
6	The fourth refinement	9
7	Alternative refinement sequence	11
8	Extracting code	13

1 What is this lecture about?

This lecture presents the design of a simple numerical algorithm, namely integer square root.

The focus of the development is on understanding how each step fits, so that the correctness of the development as a whole is clear from the steps.

Discharge of proof obligations gives greater confidence in the correctness in the final algorithm.

It is important to appreciate that our focus is not principally on numeric algorithms; square root gives us a simple starting point.

2 The Problem

The problem is to compute the integer square root of some natural number value num .

The definition of the integer square root of num is

the largest natural number that when squared does not exceed num ,

that is

$$sqrt(num) \times sqrt(num) \leq num \tag{1}$$

$$(sqrt(num) + 1) \times (sqrt(num) + 1) > num \tag{2}$$

2.1 Contexts

Theories: context

We also use a context to present a number of theorems concerning natural numbers. These will be useful for assisting the discharge of proof obligations.

CONTEXT Theories

AXIOMS

thm1: $(\forall n. n \in \mathbb{N} \Rightarrow (\exists m. m \in \mathbb{N} \wedge (n = 2 * m \vee n = 2 * m + 1)))$

thm2: $(\forall n. n \in \mathbb{N} \Rightarrow n < (n + 1) * (n + 1))$

thm3: $\forall m, n. m \in \mathbb{N} \wedge n \in \mathbb{N} \wedge m < n \Rightarrow (m + n)/2 < n$

thm4: $\forall m, n. m \in \mathbb{N} \wedge n \in \mathbb{N} \wedge m < n \Rightarrow (m + n)/2 \geq m$

END

SquareRoot_ctx: context

We use a context to define a constant num that is any natural number. num is essentially used as an argument of the event $sqrt$ in the **SquareRoot** machine.

This context *extends* the *Theories* context.

CONTEXT SquareRoot_ctx

EXTENDS Theories

CONSTANTS

num

AXIOMS

axm1: $num \in \mathbb{N}$

END

2.2 Substitution

Event-B has three forms of *substitution* for changing the value assigned to a variable:

1. *becomes equal*: $x := e$, x becomes equal to the value of the expression e ;
2. *becomes such that*: $x :| p$, x becomes a value that satisfies the predicate p . Within the predicate p x represents the value of the variable x before the substitution and x' represents the value of the variable x after the substitution.
3. *becomes in*: $x : \in s$, x becomes any value in the set s

$x := x + 1$ and $x :| x' = x + 1$ are equivalent.

2.3 The SquareRoot machine

We now develop a SquareRoot machine.

The machine

- sees the context;
- has a variable $sqrt$ that will be used to store the square root of the constant (parameter) num . Initially $sqrt$ is assigned any natural number ($sqrt : \in \mathbb{N}$).

MACHINE SquareRoot

SEES SquareRoot_ctx

VARIABLES $sqrt$

INVARIANTS

$inv1 : sqrt \in \mathbb{N}$

EVENTS

Initialisation

begin

$act1 : sqrt : \in \mathbb{N}$

end

Event $SquareRoot \hat{=}$

begin

$act1 : sqrt :| (sqrt' \in \mathbb{N}$
 $\wedge sqrt' * sqrt' \leq num$
 $\wedge num < (sqrt' + 1) * (sqrt' + 1))$

end

END

2.4 SquareRoot event

Note the use of $:$ to make a *declarative* assignment to $sqrt$ using the predicates determined earlier for the definition of *square root*.

This form of assignment requires that $sqrt$ is assigned a value consistent with the definition of square root given in (1) and (2) above.

3 First refinement

The current assignment to $sqrt$, while obviously correct, is abstract and we have to compute a concrete value.

The problem with the predicate

$$sqrt' * sqrt' \leq num \wedge num < (sqrt' + 1) * (sqrt' + 1)$$

is that it contains a conjunction of two properties for $sqrt'$ each easy to satisfy on their own, but together much more difficult.

We will split this conjunction into two predicates, each with a different variable as follows:

$$low * low \leq num$$

and

$$num < high * high$$

and contrive to bring low and $high$ closer together until

$$high = low + 1$$

at which point low will be the required square root.

MACHINE SquareRootR1

REFINES SquareRoot

SEES SquareRoot_ctx

VARIABLES $sqrt$ low $high$

INVARIANTS

inv1 : $low \in \mathbb{N}$

inv2 : $high \in \mathbb{N}$

inv3 : $low + 1 \leq high$

inv4 : $low * low \leq num$

inv5 : $num < high * high$

EVENTS

Initialisation

begin

```

act1 :  $sqrt \in \mathbb{N}$ 
act2 :  $low \mid (low' \in \mathbb{N} \wedge low' * low' \leq num)$ 
act3 :  $high \mid (high' \in \mathbb{N} \wedge num < high' * high')$ 
end
Event SquareRoot  $\hat{=}$ 
refines SquareRoot
when
grd1 :  $low + 1 = high$ 
then
act1 :  $sqrt := low$ 
end
Event Improve  $\hat{=}$ 
Status convergent
any  $l \quad h$ 
when
grd1 :  $low + 1 \neq high$ 
grd2 :  $l \in \mathbb{N} \wedge low \leq l \wedge l * l \leq num$ 
grd3 :  $h \in \mathbb{N} \wedge h \leq high \wedge num < h * h$ 
grd4 :  $l + 1 \leq h$ 
grd5 :  $h - l < high - low$ 
then
act1 :  $low, high := l, h$ 
end
VARIANT  $high - low$ 
END

```

3.1 Improving low and high

As shown in the above, the refinement of *SquareRoot* fires only when $low + 1 = high$. We need a new event to bring that about.

We introduce a new event *Improve* that brings *low* and *high* closer together, while maintaining the invariant relation on those variables.

The specification of *Improve* is abstract, simply requiring that either *low* is increased, or *high* is decreased, or both, without showing how that may be achieved concretely.

In order to ensure that *SquareRoot* will eventually fire, the new event must be convergent, so we are required to give a variant expression, which must yield a natural number and is guaranteed to decrease on each execution of *Improve*.

Since either *low* is increased or *high* is increased but $low + 1 \leq high$ an adequate expression is $high - low$.

4 Second refinement

We now do a second refinement in which we refine the *Improve* event, mapping out how we can produce a better value of *low* or *high*.

The idea is to take a value *mid* that is strictly between *low* and *high* and test whether *mid* is a better replacement for *low* or *high* and replace those values accordingly.

We know that such a value exists because $low + 1 \leq high$ and also $low + 1 \neq high$, so $low + 1 < high$ implying that there is at least one value between *low* and *high*.

In this refinement *Improve* is refined two ways into *Improve1* and *Improve2* depending on whether *low* or *high* is improved, respectively.

Witnesses: The refinements of *Improve* have eliminated the parameters *l* and *h*, so we are required to show how those parameters are represented by the refinement. This is achieved by the *with* clause, which shows the values of those parameters.

MACHINE SquareRootR2

REFINES SquareRootR1

SEES SquareRoot.ctx

VARIABLES *sqrt low high*

EVENTS

Initialisation

begin

act1 : *sqrt* : \in \mathbb{N}

act2 : *low* : | ($low' \in \mathbb{N} \wedge low' * low' \leq num$)

act3 : *high* : | ($high' \in \mathbb{N} \wedge num < high' * high'$)

end

Event *SquareRoot* $\hat{=}$

```

refines SquareRoot

when

grd1 :  $low + 1 = high$ 

then

act1 :  $sqrt := low$ 

end

Event Improve1  $\hat{=}$ 

refines Improve

any  $m$ 

when

grd1 :  $low + 1 \neq high$ 

grd2 :  $m \in \mathbb{N}$ 

grd3 :  $low < m \wedge m < high$ 

grd4 :  $m * m \leq num$ 

with

l :  $l = m$ 

h :  $h = high$ 

then

act1 :  $low := m$ 

end

Event Improve2  $\hat{=}$ 

refines Improve

any  $m$ 

when

grd1 :  $low + 1 \neq high$ 

grd2 :  $m \in \mathbb{N}$ 

grd3 :  $low < m \wedge m < high$ 

grd4 :  $num < m * m$ 

with

l :  $l = low$ 

h :  $h = m$ 

```

then

act1 : $high := m$

end

END

5 The third refinement

The third refinement refines the abstract specifications to concrete expressions.

1. We compute the initial values of low and $high$ as 0 and $num + 1$ respectively. We could find better initial values, but since the performance is going to be logarithmic, it may not be worth the expense of more complicated computations.

2. In *SquareRootR3* we refine *Improve* two ways, each with a parameter m set to the value $(low + high)/2$, that is we introduce the midpoint value between low and $high$.

This is not the only option, we could have used $low + 1$ or $high - 1$; any value that is strictly between —and not equal to— low and $high$.

The difference is one of performance, not correctness. Taking the mid point will give logarithmic performance, while incrementing or decrementing will give linear performance.

MACHINE SquareRootR3

REFINES SquareRootR2

SEES SquareRoot.ctx

VARIABLES $sqrt$ low $high$

EVENTS

Initialisation

begin

act1 : $sqrt \in \mathbb{N}$

act2 : $low := 0$

act3 : $high := num + 1$

end

Event *SquareRoot* $\hat{=}$

refines *SquareRoot*

when

grd1 : $low + 1 = high$

then

```

act1 : sqrt := low
end

Event Improve1  $\hat{=}$ 
refines Improve1
any m
when
grd1 : low + 1  $\neq$  high
grd2 :  $m = (low + high)/2$ 
grd3 :  $m * m \leq num$ 
then
act1 : low := m
end

Event Improve2  $\hat{=}$ 
refines Improve2
any m
when
grd1 : low + 1  $\neq$  high
grd2 :  $m = (low + high)/2$ 
grd3 :  $num < m * m$ 
then
act1 : high := m
end

END

```

6 The fourth refinement

In the third refinement we still have an abstract element in the form of the parameter m to the events *Improve1* and *Improve2*.

The value of m has to be produced, nondeterministically, to satisfy the guards of *Improve1*.

In *SquareRootR4* we introduce another variable, mid , to replace the parameter m .

MACHINE SquareRootR4

REFINES SquareRootR3

SEES SquareRoot_ctx

VARIABLES *sqrt low high mid*

INVARIANTS

inv1 : $mid = (low + high)/2$

EVENTS

Initialisation

begin

act1 : $sqrt \in \mathbb{N}$

act2 : $low := 0$

act3 : $high := num + 1$

act4 : $mid := (num + 1)/2$

end

Event *SquareRoot* $\hat{=}$

refines *SquareRoot*

when

grd1 : $low + 1 = high$

then

act1 : $sqrt := low$

end

Event *Improve1* $\hat{=}$

refines *Improve1*

when

grd1 : $low + 1 \neq high$

grd2 : $mid * mid \leq num$

with

m : $m = mid$

then

act1 : $low := mid$

act2 : $mid := (mid + high)/2$

end

Event *Improve2* $\hat{=}$

```

refines Improve2

when

grd1 :  $low + 1 \neq high$ 

grd2 :  $num < mid * mid$ 

with

m :  $m = mid$ 

then

act1 :  $high := mid$ 

act2 :  $mid := (low + mid)/2$ 

end

END

```

7 Alternative refinement sequence

Instead of the refinement sequence *SquarerootR3A*, *SquarerootR4* we could have effectively gone directly to *SquareRootR4* from *SquareRootR2* and this is shown in *SquareRootR3B*

Historical note: the presentation sequence of a model development is frequently different to the chronological sequence. Refinement to *SquarerootR3B* was the original refinement step from *Square-rootR2*. The extra step via *SquarerootR3A* to *SquarerootR4* was added later.

MACHINE SquareRootR3B

REFINES SquareRootR2

SEES SquareRoot_ctx

VARIABLES *sqrt low high mid*

INVARIANTS

inv1 : $mid \in \mathbb{N}$

inv2 : $mid = (low + high)/2$

inv3 : $low \leq mid$

inv4 : $mid < high$

EVENTS

Initialisation

begin

act1 : $sqrt := \mathbb{N}$

```

act2 :  $low := 0$ 
act3 :  $high := num + 1$ 
act4 :  $mid := (num + 1)/2$ 
end
Event SquareRoot  $\hat{=}$ 
refines SquareRoot
when
grd1 :  $low + 1 = high$ 
then
act1 :  $sqrt := low$ 
end
Event Improve1  $\hat{=}$ 
refines Improve1
when
grd1 :  $low + 1 \neq high$ 
grd2 :  $mid * mid \leq num$ 
with
m :  $m = mid$ 
then
act1 :  $low := mid$ 
act2 :  $mid := (mid + high)/2$ 
end
Event Improve2  $\hat{=}$ 
refines Improve2
when
grd1 :  $low + 1 \neq high$ 
grd2 :  $num < mid * mid$ 
with
m :  $m = mid$ 
then
act1 :  $high := mid$ 
act2 :  $mid := (low + mid)/2$ 
end
END

```

8 Extracting code

If the final refinement is examined carefully, it is clear that the events can be replaced by a loop:

- the subsidiary events *Improve1* and *Improve2* execute until $low + 1 = high$
- at which point the loop terminates and the final square root is computed.

Thus we can manually refine the final refinement to the following pseudo code.

$low := 0;$

$high := num + 1;$

```
while  $low + 1 \neq high$  {  
     $mid := (low + high)/2$   
    if  $mid * mid \leq num$  {  
         $low := mid$   
    }  
    else  $high := mid$ 
```

```
}
```

$sqrt := low$