System Modelling and Design
An Introduction to Modelling

Revision: 1.2, May 19, 2004

Ken Robinson

1st March 2005

mailto::k.robinson@unsw.edu.au

Contents

1 Prelude 2
2 Modelling: Why? and What? 2
   2.1 Why? .......................................................... 2
   2.2 What? .......................................................... 2
3 Software Modelling 3
   3.1 Software ......................................................... 3
   3.2 Specifications ................................................ 3
   3.3 Example ........................................................ 3
   3.4 Specification ................................................... 4
   3.5 Animation ..................................................... 5
   3.6 Refinement .................................................... 6
   3.7 Interface ....................................................... 9
A Translations 10
   A.1 SquareRoot.c ............................................... 10
   A.2 SquareRoot.h ............................................... 10
   A.3 SqrUtility.c ............................................... 10
   A.4 SqrUtility.h ............................................... 11
B Proof Obligations 12
   B.1 SquareRootI.prf ........................................... 12
   B.2 SqrtUtilityR.prf ........................................... 14
   B.3 SqrtUtilityRR.prf .......................................... 16
1 Prelude

This course is going to be concerned with the rigorous development of systems. We will be using the B Method (B), a formal development method that spans development from specification through to implementation. B is based on set theory and logic, but more of that later. There will be some B in this lecture, but it will be given without introduction and hopefully will be reasonably easy to understand.

We are concerned here with the notion of modelling, but in particular with the need for a model to enable quantitative measurement. This isn’t always the case: Unified Modelling Language (UML) is a widespread modelling language for computing systems, but in UML we get pictures and no measurement.

2 Modelling: Why? and What?

2.1 Why do we want a model?

We are going to use a mathematics based notation to describe our models, but the intention is the same as in any application where a model is used. We want a model:

- to exhibit some important characteristics of some “thing” that we want to build or implement;
- to reveal what our “thing” will “look” like;
- to help us understand how our “thing” will behave.

2.2 What should a model do?

A model must provide us with the capability of doing measurements that will enable us to establish the behaviour of the “thing” being modelled.

If we were modelling a proposed Sydney Harbour Bridge we would want to know:

- what the bridge looks like;
- the length of the bridge;
- the width of the bridge;
- the maximum load of the bridge;
- etc;

In general a model is not an implementation:

- if we wanted a model of a proposed Sydney Harbour Bridge, we would not build the bridge.

Why not?
3 Software Modelling

3.1 Is Software different?

Some think that software systems are different because of they are soft.
But, for similar reasons to why we build models of conventional engineering structures, it would be useful to build software models that concentrate on the system behaviour, not on how it is implemented.

3.2 Specification, Design and Implementation

We can differentiate between 3 phases of development:

**Specification:** is concerned with describing, or modelling, *what* a system should do. There is no requirement that a specification is executable.

**Implementation:** is concerned with *how* the specification is to be realised. In a software implementation this must be executable.

**Design:** describes the process between specification and implementation. Until the final stage of design (implementation) intermediate stages may contain a mixture of specification and implementation. We will frequently call design refinement.

3.3 A Simple Numeric Example

We mostly won’t be dealing with numerical algorithms, but the following simple example is complex enough to illustrate our point.

The following programming fragments are expressed in the low-level executable subset of the B Method (B), the language we will be using to model systems. It doesn’t need much explanation, except to note that operations can return multiple results.

**ChooseInitApprox**

\[
low, high \leftarrow \text{ChooseInitApprox}(num) \equiv \\
\begin{align*}
\text{begin} & \quad low := 0 ; \quad high := (num + 1)/2 + 1 \text{ end ;}
\end{align*}
\]

**Mystery Loop**

\[
\text{while } low + 1 \neq high \\
\text{do } low, high \leftarrow \text{ChooseBetterApprox}(num, low, high) \text{ end }
\]

**ChooseBetterApprox**

\[
low, high \leftarrow \text{ChooseBetterApprox}(num, oldlow, oldhigh) \equiv \\
\begin{align*}
\text{var} & \quad mid \text{ in} \\
mid & \quad := (oldlow + oldhigh)/2 \\
\text{if} & \quad mid \leq num/mid \\
\text{then} & \quad low := mid ; \quad high := oldhigh
\end{align*}
\]
else  \( \text{low} := \text{oldlow} \); \( \text{high} := \text{mid} \)
end

The Problem

What does it do?
Given an initial value of \( \text{num} \) as any natural number (non-negative integer), what does the loop compute?

How do you know?
How would you discover what it does?

Are you sure?
How would you verify that it does what you think it does?

The Answer

When the loop terminates the value of \( \text{low} \) is the largest natural number whose square does not exceed the value of \( \text{num} \), that is, the integer part of \( \sqrt{\text{num}} \).
Again, how would you verify that?
Indeed how do you read that from the algorithm?
Is it a good specification model?

3.4 Specification

Consider an operation with a single natural number argument that is to compute the numerical square of the value of the argument.
The specification could look like the \( \text{Sqr} \) operation in the \textbf{Square} machine.

MACHINE \textit{Square}

\textbf{OPERATIONS}
\begin{verbatim}
sqr \leftarrow \textbf{Sqr} (\text{num} ) \equiv \\
\text{PRE} \quad \text{num} \in \mathbb{N} \\
\text{THEN} \quad \text{sqr} := \text{num} \times \text{num} \\
\text{END}
\end{verbatim}

\textit{Square} is Deterministic and Computational

The specification \textit{Square} not only clearly explains what the operation does, it also shows how to compute it.

Suppose we wish to have an operation \texttt{ApproxSqrt} (\texttt{num}) that should return the largest natural number that does not exceed \( \sqrt{\text{num}} \), the mathematical square root of \( \text{num} \).

We could specify this as shown in the \texttt{ApproxSqrt} operation of the \textbf{SquareRoot} machine.
MACHINE SquareRoot

OPERATIONS
sqrt ← ApproxSqrt (num) ≜

PRE num ∈ ℕ THEN
  ANY approx
  WHERE approx ∈ ℕ ∧
  approx × approx ≤ num ∧
  num < (approx + 1) × (approx + 1)
  THEN sqrt := approx
END
END
END

ApproxSqrt is specified non-deterministically

- What the specification says is:
  
  | if | approx × approx ≤ num and (approx + 1) × (approx + 1) > num |
  |    | then approx = ApproxSqrt(num) |

- Non-determinism is frequently used in writing formal specification. It allows the specification to concentrate on the properties of the operation.

- In many, perhaps most, uses of non-determinism we are not specifying an operation that is itself non-deterministic; they are frequently deterministic. Certainly ApproxSqrt is deterministic; it is a function.

- We use non-determinism to specify what not how.

- The specification of ApproxSqrt is non-constructive; it gives no clue to how we should compute the result.

- Rather, the specification provides an acceptance test.

3.5 Animation

While the B Method (B) promises an implementation that is consistent with the specification, there can be no promise that the specification is consistent with the informal requirements.

In general, a formal specification cannot be compiled and executed, but animation enables the specifier to validate the behaviour through —possibly human-assisted— interpretation of the specification.

We will animate the operations in Square and SquareRoot.

To animate a machine, select the anim button in the Main environment.
3.6 Refinement

Refinement is the name given to a design step in which we take a specification closer to an implementation.

A refinement of an operation can be thought of as the specification of an operation that is at least as acceptable as the operation being refined.

The problem with the specification of \texttt{ApproxSqrt} is that we have to choose the value of a single variable that satisfies two conjuncts:

- \( \text{approx} \times \text{approx} \leq \text{num} \) and
- \( \text{num} < (\text{approx} + 1) \times (\text{approx} + 1) \)

It is simple to choose a value of \text{approx} to satisfy either conjunct, but not both.

Thus we refine the operation by using two variables, \text{low} and \text{high} with \( \text{low} \leq \text{approx} \), \( \text{low} < \text{high} \) and \( \text{approx} < \text{high} \), as shown in the implementation machine \texttt{SquareRootI}, where the loop moves \text{low} and \text{high} closer together under the above invariant. Termination clearly occurs when \( \text{low} + 1 = \text{high} \).

Notice that this machine uses the services of (\texttt{IMPORTS}) of a new machine \texttt{SqrtUtility}.

\begin{verbatim}
IMPLEMENTATION SquareI
REFINES Square

OPERATIONS
  \texttt{sqr} --- \texttt{Sqr (num )} \equiv
  BEGIN
      \texttt{sqr} := \texttt{num} \times \texttt{num}
  END

IMPLEMENTATION SquareRootI
REFINES SquareRoot
IMPORTS SqrtUtility

OPERATIONS
  \texttt{sqrt} --- \texttt{ApproxSqrt (num )} \equiv
  VAR  \texttt{low, high} IN
      \texttt{low, high} \leftarrow \texttt{ChooseInitApprox (num )} ;
  WHILE \texttt{low} + 1 \neq \texttt{high}
    \DO \texttt{low, high} \leftarrow \texttt{ChooseBetterApprox (num , low, high )}
  INVARIANT
      \texttt{low} \in \mathbb{N} \land \texttt{high} \in \mathbb{N} \land
      \texttt{low} + 1 \leq \texttt{high} \land
      \texttt{low} \times \texttt{low} \leq \texttt{num} \land
      \texttt{num} < \texttt{high} \times \texttt{high}
  VARIANT
      \texttt{high} \rightarrow \texttt{low}
\end{verbatim}
END;

\[
\sqrt{\text{num}} := \text{low}
\]

END

END

MACHINE SqrtUtility

OPERATIONS

\[
\text{low}, \text{high} \leftarrow \text{ChooseInitApprox} (\text{num}) \equiv
\]

PRE \ \text{num} \in \mathbb{N} \ \text{THEN}

ANY \ \text{xx}, \text{yy}

WHERE \ \text{xx} \in \mathbb{N} \land \text{yy} \in \mathbb{N} \land

\text{xx} + 1 \leq \text{yy} \land
\text{xx} \times \text{xx} \leq \text{num} \land
\text{num} < \text{yy} \times \text{yy}

THEN \ \text{low}, \text{high} := \text{xx}, \text{yy}

END

END

\[
\text{low}, \text{high} \leftarrow \text{ChooseBetterApprox} (\text{num}, \text{oldlow}, \text{oldhigh}) \equiv
\]

PRE \ \text{num} \in \mathbb{N} \land \text{oldlow} \in \mathbb{N} \land \text{oldhigh} \in \mathbb{N} \land

\text{oldlow} \times \text{oldlow} \leq \text{num} \land \text{num} < \text{oldhigh} \times \text{oldhigh} \land
\text{oldlow} + 1 < \text{oldhigh}

THEN \ \text{ANY \ xx, yy}

WHERE \ \text{xx} \in \mathbb{N} \land \text{yy} \in \mathbb{N} \land

\text{xx} + 1 \leq \text{yy} \land
\text{xx} \times \text{xx} \leq \text{num} \land
\text{num} < \text{yy} \times \text{yy} \land
\text{yy} \leq \text{xx} < \text{oldhigh} - \text{oldlow}

THEN \ \text{low}, \text{high} := \text{xx}, \text{yy}

END

END

Having introduced the machine SqrtUtility we now have to refine it to an implementation.

This is done in three refinement steps as shown in machines SqrtUtilityR, SqrtUtilityRR and SqrtUtilityRRI.

REFINEMENT SqrtUtilityR

REFINES SqrtUtility

OPERATIONS

\[
\text{low}, \text{high} \leftarrow \text{ChooseInitApprox} (\text{num}) \equiv
\]

\[
\text{low}, \text{high} := 0, (\text{num} + 1) / 2 + 1;
\]

7
ChooseBetterApprox(\text{num}, \text{oldlow}, \text{oldhigh})$
\begin{align*}
\text{END} \quad \text{WHERE} \quad \text{mid} \in \mathbb{N} \land \text{oldlow} < \text{mid} \land \text{mid} < \text{oldhigh} \quad \text{THEN} \\
\text{SELECT} \quad \text{mid} \times \text{mid} \leq \text{num} \quad \text{THEN} \\
\text{low}, \text{high} := \text{mid}, \text{oldhigh} \\
\text{WHEN} \quad \text{num} < \text{mid} \times \text{mid} \quad \text{THEN} \\
\text{low}, \text{high} := \text{oldlow}, \text{mid} \\
\text{END} \quad \text{END} \\
\text{END}
\end{align*}
Having produced implementation machines, SquareI and SqrtUtilityRRI, we can produce code by selecting the trl button in the Translators environment of the B-Toolkit.

The C code is shown in the .

3.7 Generating an interface

To run the code we can introduce an interface to Square and then generate the interface in the Generators environment of the B-Toolkit.

The interface can be run by selecting the exe (execute) button in the Translators environment.

The interface looks similar to that of the Animator, but this is not an interpretative execution; you are now running actual code.
A Translations

A.1 SquareRoot.c

#include "SquareRoot.h"

#include "SqrtUtility.h"

void
INI_SquareRoot()
{
    INI_SqrtUtility();
}

void
ApproxSqrt(_sqrt,_num)
int *_sqrt,_num;
{
    int low,high;
    ChooseInitApprox(&low,&high,_num);
    while ( low+1 != high ) {
        ChooseBetterApprox(&low,&high,_num,low,high);
    }
    *_sqrt = low;
}

A.2 SquareRoot.h

void INI_SquareRoot();
void ApproxSqrt();

A.3 SqrUtility.c

#include "SqrtUtility.h"

void
INI_SqrtUtility()
{
    ;
}

void
ChooseInitApprox(_low, _high,_num)
int *_low,*_high,_num;
{
    *_low = 0;
    *_high = (_num+1)/2+1;
}

void
ChooseBetterApprox(_low, _high,_num , _oldlow , _oldhigh)
int *_low,*_high,_num,_oldlow,_oldhigh;
```c
int mid;
mid = (_oldlow+_oldhigh)/2;
if ( mid <= _num/mid ) {
    *low = mid;
    *high = _oldhigh;
} else {
    *low = _oldlow;
    *high = mid;
}
}

A.4 SqrUtility.h

void INI_SqrtUtility();
void ChooseInitApprox();
void ChooseBetterApprox();
```
B Proof Obligations

B.1 SquareRootI.prf

Proof Obligations for SquareRootI.imp

\[ \text{ApproxSqrt.1} \]
\[
cst \ (\text{SquareRootI}_1) \land \ctx \ (\text{SquareRootI}_1) \land 
inv \ (\text{SquareRootI}_1) \land \asn \ (\text{SquareRootI}_1) \land 
pre \ (\text{ApproxSqrt}) \\
\Rightarrow \\
low \in \mathbb{N} \land 
high \in \mathbb{N} \land 
\text{low} + 1 \leq \text{high} \land 
\text{low} \times \text{low} \leq \text{num} \land 
\text{num} < \text{high} \times \text{high} \\
\Rightarrow \\
\text{high} - \text{low} \in \mathbb{N} 
\]

\[ \text{ApproxSqrt.2} \]
\[
cst \ (\text{SquareRootI}_1) \land \ctx \ (\text{SquareRootI}_1) \land 
inv \ (\text{SquareRootI}_1) \land \asn \ (\text{SquareRootI}_1) \land 
pre \ (\text{ApproxSqrt}) \\
\Rightarrow \\
low \in \mathbb{N} \land 
high \in \mathbb{N} \land 
\text{low} + 1 \leq \text{high} \land 
\text{low} \times \text{low} \leq \text{num} \land 
\text{num} < \text{high} \times \text{high} \\
\neg \ (\text{low} + 1 = \text{high}) \\
\Rightarrow \\
\text{low} + 1 < \text{high} 
\]

\[ \text{ApproxSqrt.3} \]
\[
cst \ (\text{SquareRootI}_1) \land \ctx \ (\text{SquareRootI}_1) \land 
inv \ (\text{SquareRootI}_1) \land \asn \ (\text{SquareRootI}_1) \land 
pre \ (\text{ApproxSqrt}) \\
\Rightarrow \\
low \in \mathbb{N} \land 
high \in \mathbb{N} \land 
\text{low} + 1 \leq \text{high} \land 
\text{low} \times \text{low} \leq \text{num} \land 
\text{num} < \text{high} \times \text{high} \\
\text{low} + 1 = \text{high} \\
\Rightarrow 
\]
\[ \text{num} < (\text{low} + 1) \times (\text{low} + 1) \]
B.2  SqrtUtilityR.prf

Proof Obligations for SqrtUtilityR.ref

ChooseInitApprox.1
cst ( SqrtUtilityR_1 ) \land ctx ( SqrtUtilityR_1 ) \land
inv ( SqrtUtilityR_1 ) \land asn ( SqrtUtilityR_1 ) \land
pre ( ChooseInitApprox )
\Rightarrow
0 \in \mathbb{N}

ChooseInitApprox.2
cst ( SqrtUtilityR_1 ) \land ctx ( SqrtUtilityR_1 ) \land
inv ( SqrtUtilityR_1 ) \land asn ( SqrtUtilityR_1 ) \land
pre ( ChooseInitApprox )
\Rightarrow
(\text{num} + 1) / 2 + 1 \in \mathbb{N}

ChooseInitApprox.3
cst ( SqrtUtilityR_1 ) \land ctx ( SqrtUtilityR_1 ) \land
inv ( SqrtUtilityR_1 ) \land asn ( SqrtUtilityR_1 ) \land
pre ( ChooseInitApprox )
\Rightarrow
1 \leq (\text{num} + 1) / 2 + 1

ChooseInitApprox.4
cst ( SqrtUtilityR_1 ) \land ctx ( SqrtUtilityR_1 ) \land
inv ( SqrtUtilityR_1 ) \land asn ( SqrtUtilityR_1 ) \land
pre ( ChooseInitApprox )
\Rightarrow
0 \leq \text{num}

ChooseInitApprox.5
cst ( SqrtUtilityR_1 ) \land ctx ( SqrtUtilityR_1 ) \land
inv ( SqrtUtilityR_1 ) \land asn ( SqrtUtilityR_1 ) \land
pre ( ChooseInitApprox )
\Rightarrow
\text{num} < ((\text{num} + 1) / 2 + 1) \times ((\text{num} + 1) / 2 + 1)

ChooseBetterApprox.1
cst ( SqrtUtilityR_1 ) \land ctx ( SqrtUtilityR_1 ) \land
inv ( SqrtUtilityR_1 ) \land asn ( SqrtUtilityR_1 ) \land
pre ( ChooseBetterApprox )
⇒
mid ∈ \mathbb{N} \land
oldlow < mid \land
mid < oldhigh \land
mid \times mid \leq num
⇒
mid + 1 \leq oldhigh

ChooseBetterApprox.2
\begin{align*}
cst ( SqrtUtilityR_1 ) \land cxt ( SqrtUtilityR_1 ) \land 
inv ( SqrtUtilityR_1 ) \land asn ( SqrtUtilityR_1 ) \land 
pre ( ChooseBetterApprox )
⇒
mid ∈ \mathbb{N} \land
oldlow < mid \land
mid < oldhigh \land
mid \times mid \leq num
⇒
oldhigh - mid < oldhigh - oldlow
\end{align*}

ChooseBetterApprox.3
\begin{align*}
cst ( SqrtUtilityR_1 ) \land cxt ( SqrtUtilityR_1 ) \land 
inv ( SqrtUtilityR_1 ) \land asn ( SqrtUtilityR_1 ) \land 
pre ( ChooseBetterApprox )
⇒
mid ∈ \mathbb{N} \land
oldlow < mid \land
mid < oldhigh \land
num < mid \times mid
⇒
oldlow + 1 \leq mid
\end{align*}

ChooseBetterApprox.4
\begin{align*}
cst ( SqrtUtilityR_1 ) \land cxt ( SqrtUtilityR_1 ) \land 
inv ( SqrtUtilityR_1 ) \land asn ( SqrtUtilityR_1 ) \land 
pre ( ChooseBetterApprox )
⇒
mid ∈ \mathbb{N} \land
oldlow < mid \land
mid < oldhigh \land
num < mid \times mid
⇒
mid - oldlow < oldhigh - oldlow
\end{align*}
ChooseBetterApprox.1
\[
cst (\text{SqrtUtilityRR}_2) \land \text{ctx (SqrtUtilityRR}_2) \land
\]
\[
\text{inv (SqrtUtilityRR}_2) \land \text{asn (SqrtUtilityRR}_2) \land
\]
\[
\text{pre (ChooseBetterApprox)}
\Rightarrow
\]
\[
\text{num} < (\text{oldlow} + \text{oldhigh}) / 2 \times ((\text{oldlow} + \text{oldhigh}) / 2)
\Rightarrow
\]
\[
\exists \text{mid}_0. (\hspace{1cm}
\]
\[
\text{mid}_0 \in \mathbb{N} \land
\]
\[
\text{oldlow} < \text{mid}_0 \land
\]
\[
\text{mid}_0 < \text{oldhigh} \land
\]
\[
(\text{mid}_0 \times \text{mid}_0 \leq \text{num} \land \text{oldlow} \land \text{oldlow} = \text{mid}_0)
\land
\]
\[
(\text{num} < \text{mid}_0 \times \text{mid}_0 \land (\text{oldlow} + \text{oldhigh}) / 2 = \text{mid}_0)
\)
\]

ChooseBetterApprox.2
\[
cst (\text{SqrtUtilityRR}_2) \land \text{ctx (SqrtUtilityRR}_2) \land
\]
\[
\text{inv (SqrtUtilityRR}_2) \land \text{asn (SqrtUtilityRR}_2) \land
\]
\[
\text{pre (ChooseBetterApprox)}
\Rightarrow
\]
\[
(\text{oldlow} + \text{oldhigh}) / 2 \times ((\text{oldlow} + \text{oldhigh}) / 2) \leq \text{num}
\Rightarrow
\]
\[
\exists \text{mid}_0. (\hspace{1cm}
\]
\[
\text{mid}_0 \in \mathbb{N} \land
\]
\[
\text{oldlow} < \text{mid}_0 \land
\]
\[
\text{mid}_0 < \text{oldhigh} \land
\]
\[
(\text{mid}_0 \times \text{mid}_0 \leq \text{num} \land (\text{oldlow} + \text{oldhigh}) / 2 = \text{mid}_0)
\land
\]
\[
(\text{num} < \text{mid}_0 \times \text{mid}_0 \land \text{oldhigh} = \text{mid}_0 \land (\text{oldlow} + \text{oldhigh}) / 2 = \text{oldlow})
\)
\]
B.4 SqrtUtilityRRI

Proof Obligations for SqrtUtilityRRI.imp

---

ChooseBetterApprox.1
\[
\begin{align*}
&\text{cst} (\text{SqrtUtilityRRI}_3) \land \text{ctx} (\text{SqrtUtilityRRI}_3) \land \\
&\text{inv} (\text{SqrtUtilityRRI}_3) \land \text{asn} (\text{SqrtUtilityRRI}_3) \land \\
&\text{pre} (\text{ChooseBetterApprox}) \\
\Rightarrow \\
&\text{num} / ((\text{oldlow} + \text{oldhigh}) / 2) < (\text{oldlow} + \text{oldhigh}) / 2 \\
\Rightarrow \\
&(\text{oldlow} + \text{oldhigh}) / 2 \times ((\text{oldlow} + \text{oldhigh}) / 2) \leq \text{num} \land \\
&(\text{oldlow} + \text{oldhigh}) / 2 = \text{oldhigh} \land \\
&\text{oldlow} = (\text{oldlow} + \text{oldhigh}) / 2 \\
&\lor \\
&\text{num} < (\text{oldlow} + \text{oldhigh}) / 2 \times ((\text{oldlow} + \text{oldhigh}) / 2)
\end{align*}
\]

---

ChooseBetterApprox.2
\[
\begin{align*}
&\text{cst} (\text{SqrtUtilityRRI}_3) \land \text{ctx} (\text{SqrtUtilityRRI}_3) \land \\
&\text{inv} (\text{SqrtUtilityRRI}_3) \land \text{asn} (\text{SqrtUtilityRRI}_3) \land \\
&\text{pre} (\text{ChooseBetterApprox}) \\
\Rightarrow \\
&(\text{oldlow} + \text{oldhigh}) / 2 \leq \text{num} / ((\text{oldlow} + \text{oldhigh}) / 2) \\
\Rightarrow \\
&(\text{oldlow} + \text{oldhigh}) / 2 \times ((\text{oldlow} + \text{oldhigh}) / 2) \leq \text{num} \\
&\lor \\
&\text{num} < (\text{oldlow} + \text{oldhigh}) / 2 \times ((\text{oldlow} + \text{oldhigh}) / 2) \land \\
&\text{oldhigh} = (\text{oldlow} + \text{oldhigh}) / 2 \land \\
&(\text{oldlow} + \text{oldhigh}) / 2 = \text{oldlow}
\end{align*}
\]