Reasoning About Programs

Mark Wheelhouse
PhD Student
Department of Computing
Imperial College London
So What is a PhD Anyway?

- Research
  - Extending Existing Work
  - Doing Something New
  - Experimentation
  - Results
  - Understanding
  - Revelations

- Teaching
  - Communicating
  - Becoming an Expert
  - Explaining to Experts
  - Research Papers
  - Explaining to Non-experts
  - Presentations
What am I Doing?

“Context Logic, Tree Update and Concurrency”
What am I Doing?

“Context Logic, Tree Update and Concurrency”

Huh?!
What am I Doing?

“Context Logic, Tree Update and Concurrency”

Huh?!

What's He Talking About?!
What am I Doing?

“Context Logic, Tree Update and Concurrency”

Huh?!

What’s He Talking About?!

Using Mathematics to prove properties about computer programs

Sometimes we want to be 100% sure the computer is doing what it should be
Why Prove Programs?
A History Lesson
Why Prove Programs?
A History Lesson

1980's - Therac 25
Why Prove Programs?
A History Lesson

1980’s - Therac 25

race
condition
Why Prove Programs?
A History Lesson

1980’s - Therac 25

1991 - MIM-104 Patriot
Why Prove Programs?
A History Lesson

1980’s - Therac 25
race condition

1991 - MIM-104 Patriot
rounding error
Why Prove Programs?  
A History Lesson

1980's - Therac 25

1991 - MIM-104 Patriot

1996 - Ariane 5

race condition

rounding error
Why Prove Programs?
A History Lesson

1980's - Therac 25
race condition

1991 - MIM-104 Patriot
rounding error

1996 - Ariane 5
data overflow!
Why Prove Programs?
A History Lesson

1980's - Therac 25
- race condition

1991 - MIM-104 Patriot
- rounding error

1996 - Ariane 5
- data overflow!

1997 - USS Yorktown
Why Prove Programs?
A History Lesson

1980's - Therac 25
race condition

1991 - MIM-104 Patriot
rounding error

1996 - Ariane 5
data overflow!

1997 - USS Yorktown
divide by zero error!
What Kind of Systems do We Want to be Sure of?
What Kind of Systems do We Want to be Sure of?

- Internet Banking
- Automatic Breaking
- Nuclear Power Stations
- Flight Control Systems
- Building Design Software
- IFF Targeting Systems
- Heart Monitors
- Spaceships
- Pacemakers
What Kind of Systems do We Want to be Sure of?

i.e. - Anything that is safety critical
- When the cost of system failure is huge

Internet Banking
Automatic Breaking
Nuclear Power Stations
Flight Control Systems
Building Design Software
IFF Targeting Systems
Heart Monitors
Spaceships
Pacemakers

i.e. - Anything that is safety critical
- When the cost of system failure is huge
Spotting Failure

- We want to spot when a system can fail.
- We want to prove that a system will not fail.
- We can use mathematics (namely logic) to do this.
Logic – The Basics

boolean variables - “have one of two values”

\[ P = \begin{array}{cc}
\text{true} & \text{false} \\
1 & 0 \\
\end{array} \]
Logic – The Basics

boolean variables - “have one of two values”

\[ p = \begin{array}{c|c|c}
\text{true} & \text{false} \\
1 & 0 
\end{array} \]

boolean operators - “ways of combining variables”

and: \( \wedge \)  

or: \( \vee \)  

implies: \( \Rightarrow \)
Logic – The Basics

boolean variables – “have one of two values”

\[ p = \begin{align*} &\text{true} \quad &\text{false} \\ 1 \quad &0 \end{align*} \]

boolean operators – “ways of combining variables”

and: \( \land \)

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p \land q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

or: \( \lor \)

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p \lor q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

implies: \( \Rightarrow \)

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p \Rightarrow q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Logic - Predicates

X,Y,Z are things (objects, people, concepts,...)

isRed(X) = true if X is red
isRound(X) = true if X is round
isRedBall(X) = true if X is a red ball
Logic - Predicates

X, Y, Z are things (objects, people, concepts, ...)

\[
\text{isRed}(X) = \text{true if } X \text{ is red}
\]

\[
\text{isRound}(X) = \text{true if } X \text{ is round}
\]

\[
\text{isRedBall}(X) = \text{true if } X \text{ is a red ball}
\]

\[
\text{isRedBall}(X) \Rightarrow \text{isRed}(X) \land \text{isRound}(X)
\]
another way of looking at this using a satisfaction relation,

\[ X \models \text{isRed} \iff X \text{ is red} \]
\[ X \models \text{isRound} \iff X \text{ is round} \]
\[ X \models \text{isRedBall} \iff X \text{ is a red ball} \]
Logic – Structures

another way of looking at this using a satisfaction relation,

\[ \models \]

\[ X \models \text{isRed} \iff X \text{ is red} \]
\[ X \models \text{isRound} \iff X \text{ is round} \]
\[ X \models \text{isRedBall} \iff X \text{ is a red ball} \]

\[ \text{isRedBall} \Rightarrow \text{isRed} \land \text{isRound} \]
Logic – Structures

\[
data \vdash \text{Tree-Formula}
\]

\[
tree \ T ::= 
\]
Logic - Structures

data $\vdash$ Tree-Formula

$\text{tree } T ::= 0$  empty tree
Logic - Structures

data \vdash \text{Tree-Formula}

tree \ T ::= \ 0 \quad \text{empty tree}

n[T] \quad \text{tree node}
Logic - Structures

data \vdash \text{Tree-Formula}

tree \quad T ::= \quad 0 \quad \text{empty tree}
\quad n[T] \quad \text{tree node}
\quad T \mid T \quad \text{ordered trees}
Logic - Structures

data \models Tree-Formula

tree T ::= 0 \quad \text{empty tree}

\quad n[T] \quad \text{tree node}

\quad T | T \quad \text{ordered trees}

\text{Eg:} \quad 1[ 2[0] | 3[0] ]
Tree Examples

a) 1

d) 1
   2
   3

b) 1 2

e) 1
   2
   3 4

f) 1
   2
   3
   4
   5
   6
   7
Tree Examples

a) 1 [✓] 1[0]

d) 1
   2 3

e) 1
   2 3
      4

f) 1
   2 3 4
      5 6 7
Tree Examples

a) \( 1 \)  \( 1[0] \)

d) 

b) \( 1 \)  \( 2 \)

c) 

d) 

e) 
f)
Tree Examples

a) \(1\) \[\checkmark\]  
\[1[0]\]

d) 

b) \(1\) \(2\) \[\checkmark\]  
\[1[0] \mid 2[0]\]

e) 

f) 

Tree Examples

a) 1 1[0] ✓

b) 1 2[0] ✓

1[0] | 2[0]

c) 1 2

1

3

X

d) 1

2 3 ✓

1 [ 2[0] | 3[0] ]

e) 1

2

3

4

3

e) 1

2

4

3

2

5

6

7

f) 1

2

3

4

5

6

7
Tree Examples

a) 1 [0] ✓

b) 1 [0] | 2 [0] ✓

c) 1 2
   3

   ✓


d) 1
   2
   3

   1 [2 [0] | 3 [0]]

   ✓

e) 1
   2
   3

   4

   3

   ✓

f) 1
   2
   3

   4

   5
   6
   7

   ✓
Tree Examples

a) \[ 1 \] ✓
   \[ 1[0] \]

b) \[ 1 \] \[ 2 \] ✓
   \[ 1[0] | 2[0] \]

c) \[ 1 \]
   \[ 2 \]
   \[ 3 \]
   \[ 1[0] \]

 d) \[ 1 \]
   \[ 2 \]
   \[ 3 \]
   \[ 1[2[0] | 3[0]] \]

 e) \[ 1 \]
   \[ 2 \]
   \[ 3 \]
   \[ 4 \]
   \[ 1[2[5[0]] | 3[0] | 4[6[0] | 7[0]]] \]

 f) \[ 1 \]
   \[ 2 \]
   \[ 3 \]
   \[ 4 \]
   \[ 5 \]
   \[ 6 \]
   \[ 7 \]
Data Update

We want to describe the behavior of a program,

Eg:

\[
\{ \text{colour}(X, ?) \}
\]

\[
\text{paint}(X, \text{Green})
\]

\[
\{ \text{colour}(X, \text{Green}) \}
\]
Data Update

We want to describe the behavior of a program, \( \{ P \} \) computer command \( \{ Q \} \)

pre-condition \hspace{1cm} \text{post-condition}

Eg:

\[
\{ \text{colour}(X, \, ?) \} \\
\text{paint}(X, \, \text{Green}) \\
\{ \text{colour}(X, \, \text{Green}) \} 
\]
Data Update

We want to describe the behavior of a program, \( \{ P \} \) computer command \( \{ Q \} \)

pre-condition \hspace{2cm} \text{post-condition}

pre-condition must hold before and post-condition must hold after, or we get a fault!

Eg:

\[ \begin{align*}
\{ \text{colour}(X, \, ?) \} \\
\text{paint}(X, \, \text{Green}) \\
\{ \text{colour}(X, \, \text{Green}) \}
\end{align*} \]
Local Data Update

\[
\begin{align*}
\{ & \ y=3, \ x=n, \ z=2 \ \} \\
\text{addOne}(x) & \\
\{ & \ y=3, \ x=n+1, \ z=2 \ \}
\end{align*}
\]
Local Data Update

\[
\begin{align*}
\{ y &= 3, \; x = n, \; z = 2 \} \\
\text{addOne}(x) \\
\{ y &= 3, \; x = n+1, \; z = 2 \} \\
\downarrow \\
\{ x &= n \} \\
\text{addOne}(x) \\
\{ x &= n+1 \}
\end{align*}
\]
Local Tree Update

\{ \ m[ \ t_1 \ | \ n[t_2] \ | \ t_3 \ ] \ \} \\
\ p \ := \ goUpTree(n) \\
\{ \ m[ \ t_1 \ | \ n[t_2] \ | \ t_3 \ ] \ \land \ (p=m) \ \} \\

\{ \ n[t] \ \} \\
deleteTree(n) \\
\{ \ 0 \ \} \\

\{ \ n[t] \ \} \\
addNodeAfter(n,x) \\
\{ \ n[t] \ | \ x[0] \ \}
Example - Family Trees

A more complex data structure,
Example - Family Trees

A more complex data structure,

database $D ::= $
Example – Family Trees

A more complex data structure,

database $D ::= O_D$ empty database
Example - Family Trees

A more complex data structure,

database $D ::= 0_D$ empty database
familyName$[F]$ one family
Example – Family Trees

A more complex data structure,

\[
\text{database} \ D ::= \ O_D \quad \text{empty database} \\
\text{familyName}[F] \quad \text{one family} \\
D + D \quad \text{database join}
\]
Example – Family Trees

A more complex data structure,

database \( D \) ::= \( 0_D \) empty database

\( \text{familyName}[^F] \) one family

\( D + D \) database join

family tree \( F \) ::=
Example - Family Trees

A more complex data structure,

\[
\text{database } D ::= 0_D \quad \text{empty database}
\]

\[
\text{familyName}[F] \quad \text{one family}
\]

\[
D + D \quad \text{database join}
\]

\[
\text{family tree } F ::= 0_F \quad \text{empty tree}
\]
Example - Family Trees

A more complex data structure,

\[
\text{database } D ::= 0_D \quad \text{empty database}
\]

\[
\text{familyName}[F] \quad \text{one family}
\]

\[
D + D \quad \text{database join}
\]

\[
\text{family tree } F ::= 0_F \quad \text{empty tree}
\]

\[
\text{name:marTo} [F] \quad \text{tree node}
\]

\[
\text{Gen:Age} [F]
\]
Example - Family Trees

A more complex data structure,

\[
\text{database } D ::= \quad O_D \quad \text{empty database}
\]

\[
\text{familyName}[F] \quad \text{one family}
\]

\[
D + D \quad \text{database join}
\]

\[
\text{family tree } F ::= \quad O_F \quad \text{empty tree}
\]

\[
\text{name:marTo} [F] \quad \text{tree node}
\]

\[
\text{Gen:Age} \quad \text{ordered trees}
\]

\[
F \mid F
\]
Example - Family Tree

Nodes

name: marTo [F] tree node
Gen: Age
Example - Family Tree Nodes

name: marTo
Gen: Age

[F] tree node

name  marTo
Gen  Age

II
Example - Family Tree Nodes

name:marTo [F] tree node
Gen:Age

person's name
Example - Family Tree Nodes

name:marTo  
Gen:Age  [F]  tree node

- person's name
- person's gender (M or F)
Example - Family Tree Nodes

- Name:
- Gender:
- Age:
- Married to: [F]

Person's name
Person's gender (M or F)
 Married to (- if unmarried)
Example – Family Tree Nodes

ame:marTo [F] tree node

Gen:Age

married to (- if unmarried)
age in years

person’s name

person’s gender (M or F)
Example - Family Tree

Nodes

- **name**: person's name
- **marTo**: married to (- if unmarried)
- **Gen**: gender (M or F)
- **Age**: age in years

Tree node:

- **name**: person's name
- **marTo**: married to (- if unmarried)
- **Gen**: gender (M or F)
- **Age**: age in years

Links to children
Example - The Simpsons

Simpson

Abe  Mona
M    74 / D

Homer  Marge
M    38 / 38

Bart  Lisa  Maggie
M    F    F
10   8    2
Example - Commands
Example - Commands

\{
| name | ? |
| ?    | ? |
\} reName(name, nom) \{
| nom  | ? |
| ?    | ? |
\}


Example - Commands

reName(name, nom)
Example - Commands

reName(name, nom)

reage(name, x)

marry(m, w)
Example - Commands

1. `reName(name, nom)`
2. `reage(name, x)`
3. `marry(m, w)`
4. `birth(mother, G, child)`
Now you are going to perform concurrent local reasoning!
Example - Commands

- `reName(name, nom)`
- `reage(name, x)`
- `marry(m,w)`
- `birth(mother, G, child)`