Static Verification of Message Passing Programs

Wytse Oortwijn, Stefan Blom, and Marieke Huisman

Formal Methods and Tools, University of Twente

March 22, 2016
Writing software correctly is hard

Motivational examples

```c
int value = 0;
void increase(int n) {
    value ← value + n;
}
```
Writing software correctly is hard

Motivational examples

$400 million Pentium bug
Writing software correctly is hard

Motivational examples

$400 million Pentium bug

THERAC-25 radiation therapy machine
Writing software correctly is hard

Motivational examples

$400 million Pentium bug

THERAC-25 radiation therapy machine

Floyd & Hoare: Hoare logic

```c
int value = 0;

void increase(int n):
    value ← value + n;
```
Writing software correctly is hard

Motivational examples

$400 million Pentium bug

THERAC-25 radiation therapy machine

Floyd & Hoare: Hoare logic

```c
int value = 0;

requires n > 0;

void increase(int n):
    value ← value + n;
```
Writing software correctly is hard

Motivational examples

$400 million Pentium bug

THERAC-25 radiation therapy machine

Floyd & Hoare: Hoare logic

```c
int value = 0;

requires n > 0;
ensures value = old(value) + n;
void increase(int n):
    value ← value + n;
```
Writing software correctly is hard

Motivational examples

- $400 million Pentium bug
- THERAC-25 radiation therapy machine

Floyd & Hoare: Hoare logic

```c
int value = 0;

requires n > 0;
ensures value = old(value) + n;
void increase(int n):
    value ← value + n;
```

Hoare triple reasoning

\{P\}S\{Q\}
Writing software correctly is hard

Motivational examples

$400 million Pentium bug

THERAC-25 radiation therapy machine

Floyd & Hoare: Hoare logic

```c
int value = 0;

requires n > 0;
ensures value = old(value) + n;
void increase(int n):
    value ← value + n;
```

Hoare triple reasoning

\[
\{P\} S \{Q\}
\]

\[
\{P\} S \{Q\} \quad \{Q\} T \{R\}
\]

\[
\{P\} S ; T \{R\}
\]

etc...
Introduction

Approach

Static verification

Hoare logic extensions

- **Reynolds**: Separation logic (2002)
- **Boyland**: Permission-based separation logic (2003)
- **Parkinson**: Separation logic for Java (2005)
Introduction

Approach

Static verification

Hoare logic extensions

- **Reynolds**: Separation logic (2002)
- **Boyland**: Permission-based separation logic (2003)
- **Parkinson**: Separation logic for Java (2005)

the VerCors toolset

Verification of concurrent software

Proving *data race freedom* and *functional program properties*
Static verification

Hoare logic extensions

- **Reynolds**: Separation logic (2002)
- **Boyland**: Permission-based separation logic (2003)
- **Parkinson**: Separation logic for Java (2005)

the VerCors toolset

Verification of concurrent software
Proving *data race freedom* and *functional program properties*

Research question
Can we leverage verification techniques for concurrent software to message passing programs?
Message Passing Interface

The MPI standard

Sending a message:

MPI

Send(j, m)

Receiving a message:

m := MPI

Recv(j)

Broadcasting a message:

MPI

Bcast(m)

etc...

Challenges

1. Message exchanges are often concurrent.
2. Number of processes are often unknown.
3. Number of possible behaviours are often unbounded.

Our solution

1. Use separation logic for local correctness.
2. Capture communication behaviour in abstract models, called futures.
3. Model checking the futures to show functional correctness.
The MPI standard

- **Sending a message:**
  
  ```c
  MPI_Send(j, m)
  ```

- **Receiving a message:**
  
  ```c
  m := MPI_Recv(j)
  ```

- **Broadcasting a message:**
  
  ```c
  MPI_Bcast(m)
  ```

Challenges

1. Message exchanges are often concurrent.
2. Number of processes are often unknown.
3. Number of possible behaviours are often unbounded.

Our solution

1. Use separation logic for local correctness.
2. Capture communication behaviour in abstract models, called **futures**.
3. Model checking the futures to show functional correctness.
Message Passing Interface

The MPI standard

- **Sending a message:**
  \[ \text{MPI\_Send}(j, m) \]

- **Receiving a message:**
  \[ m := \text{MPI\_Recv}(j) \]
The MPI standard

- **Sending a message:**
  \[\text{MPI\_Send}(j, m)\]

- **Receiving a message:**
  \[m := \text{MPI\_Recv}(j)\]

- **Broadcasting a message:**
  \[\text{MPI\_Bcast}(m)\]

- **etc...**

Challenges

1. Message exchanges are often concurrent.
2. Number of processes are often unknown.
3. Number of possible behaviours are often unbounded.

Our solution

1. Use separation logic for local correctness.
2. Capture communication behaviour in abstract models, called futures.
3. Model checking the futures to show functional correctness.
Introduction

Approach

Message Passing Interface

The MPI standard

- Sending a message: MPI_Send(j, m)
- Receiving a message: m := MPI_Recv(j)
- Broadcasting a message: MPI_Bcast(m)
- etc...

Challenges

1. Message exchanges are often concurrent.

Wytse Oortwijn, FMT, University of Twente
The MPI standard
- **Sending a message:**
  \[ \text{MPI	extunderscore Send}(j, m) \]
- **Receiving a message:**
  \[ m := \text{MPI	extunderscore Recv}(j) \]
- **Broadcasting a message:**
  \[ \text{MPI	extunderscore Bcast}(m) \]
- etc...

Challenges
1. Message exchanges are often **concurrent**.
2. Number of processes are often **unknown**.

Wytse Oortwijn, FMT, University of Twente
Message Passing Interface

The MPI standard

- **Sending a message:**
  
  $\text{MPI\_Send}(j, m)$

- **Receiving a message:**
  
  $m := \text{MPI\_Recv}(j)$

- **Broadcasting a message:**
  
  $\text{MPI\_Bcast}(m)$

- etc...

Challenges

1. Message exchanges are often concurrent.
2. Number of processes are often unknown.
3. Number of possible behaviours are often unbounded.
Message Passing Interface

The MPI standard

- **Sending a message:**
  \[ \text{MPI\_Send}(j, m) \]

- **Receiving a message:**
  \[ m := \text{MPI\_Recv}(j) \]

- **Broadcasting a message:**
  \[ \text{MPI\_Bcast}(m) \]

- etc...

Challenges

1. Message exchanges are often **concurrent**.
2. Number of processes are often **unknown**.
3. Number of possible behaviours are often **unbounded**.

Our solution

1. Use separation logic for local correctness.
2. Capture communication behaviour in abstract models, called **futures**.
3. Model checking the futures to show functional correctness.
How to reason about distributed programs?

MPI program + separation logic
How to reason about distributed programs?

MPI program + separation logic

Abstract model of the communication behaviour. (process algebra)
How to reason about distributed programs?

MPI program + separation logic

Abstract model of the communication behaviour.
(process algebra)
How to reason about distributed programs?

MPI program + separation logic

verify

Abstract model of the communication behaviour. (process algebra)

Temporal properties
Safety properties, e.g. resource leakage, deadlocks, etc.
How to reason about distributed programs?

MPI program + separation logic

Abstract model of the communication behaviour. (process algebra)

temporal properties
  safety properties, e.g.
  resource leakage, deadlocks, etc.

verify
How to reason about distributed programs?

MPI program + separation logic

Abstract model of the communication behaviour. (process algebra)

Temporal properties
Safety properties, e.g. resource leakage, deadlocks, etc.

verify

verified!
mCRL2: Process algebra

**Example vending machine**

\[
\text{process} \quad \text{Machine()} \equiv \text{coin} \cdot \\
(\text{water} \cdot \text{Machine()} + \text{coin} \cdot \text{cola} \cdot \text{Machine()})
\]
mCRL2: Process algebra

Example vending machine

\[
\text{process } \text{Machine}() \equiv \text{coin} \cdot \\
(\text{water} \cdot \text{Machine}() + \text{coin} \cdot \text{ cola} \cdot \text{Machine}())
\]
Abstracting MPI primitives

MPI primitives $\mapsto$ corresponding actions

- `MPI_Send(int dest, msg m)`
Abstracting MPI primitives

MPI primitives $\rightsquigarrow$ corresponding actions

- MPI_Send(int dest, msg m)
- action send : int \times msg
Abstracting MPI primitives

MPI primitives $\sim$ corresponding actions

- MPI_Send(int dest, msg m)
- msg m := MPI_Recv(int src)
- action send : int $\times$ msg
- action recv : int $\times$ msg
Abstracting MPI primitives

- **MPI_Send**(int dest, msg m)
- **msg m := MPI_Recv**(int src)
- **MPI_Bcast**(msg m)
- **MPI_Barrier**()

- action **send** : int × msg
- action **recv** : int × msg
- action **bcast** : msg
- action **barrier**
Abstracting MPI primitives

<table>
<thead>
<tr>
<th>MPI primitives</th>
<th>Corresponding actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send(int dest, msg m)</td>
<td>action send : int × msg</td>
</tr>
<tr>
<td>msg m := MPI_Recv(int src)</td>
<td>action recv : int × msg</td>
</tr>
<tr>
<td>MPI_Bcast(msg m)</td>
<td>action bcast : msg</td>
</tr>
<tr>
<td>MPI_Barrier()</td>
<td>action barrier</td>
</tr>
</tbody>
</table>

Finding a correspondence: Hoare-triple reasoning
Abstracting MPI primitives

**MPI primitives ⇻ corresponding actions**

- `MPI_Send(int dest, msg m)`
- `msg m := MPI_Recv(int src)`
- `MPI_Bcast(msg m)`
- `MPI_Barrier()`
- **Action** `send : int × msg`
- **Action** `recv : int × msg`
- **Action** `bcast : msg`
- **Action** `barrier`

**Finding a correspondence: Hoare-triple reasoning**

\[
\{send(i, m) \odot F\} \quad MPI\_Send(i, m) \{F\}
\]
### Abstracting MPI primitives

<table>
<thead>
<tr>
<th>MPI primitives</th>
<th>Corresponding actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send(int dest, msg m)</td>
<td>action send : int \times msg</td>
</tr>
<tr>
<td>msg m := MPI_Recv(int src)</td>
<td>action recv : int \times msg</td>
</tr>
<tr>
<td>MPI_Bcast(msg m)</td>
<td>action bcast : msg</td>
</tr>
<tr>
<td>MPI_BARRIER()</td>
<td>action barrier</td>
</tr>
</tbody>
</table>

### Finding a correspondence: Hoare-triple reasoning

\[
\{\text{send}(i, m) \cdot F\} \text{MPI_Send}(i, m)\{F\} \\
\{\text{recv}(i, v) \cdot F\} v := \text{MPI_Recv}(i)\{F\} \\
\{\text{bcast}(m) \cdot F\} \text{MPI_Bcast}(m)\{F\} \\
\{\text{barrier()} \cdot F\} \text{MPI_BARRIER()}\{F\}
\]
Example program abstraction

Example MPI program

```c
void main(int k):
    int v ← MPI_Recv(⋆)
    MPI_Send(0, v + k)
```

Predicted future process

\[ P(int k) ≡ recv(⋆, i) \cdot send(0, i + k) \]
Example program abstraction

**Example MPI program**

```c
void main(int k):
    int v ← MPI_Recv(⋆)
    MPI_Send(0, v + k)
```

**Predicted future**

```
process P(int k) ≡
recv(⋆, i) · send(0, i + k)
```
Example program abstraction

Example MPI program

\textbf{requires} Future(P(k) \cdot \epsilon)

\textbf{void} main(int k):
\textbf{int} v \leftarrow \text{MPIRecv}(\ast)
\text{MPISend}(0, v + k)

Predicted future

\textbf{process} P(\text{int} \ k) \equiv
\text{recv}(\ast, i) \cdot \text{send}(0, i + k)
Example program abstraction

**Example MPI program**

```plaintext
requires Future(P(k) \cdot \epsilon)
ensures Future(\epsilon)
void main(int k):
    int v ← MPI_Recv(⋆)
    MPI_Send(0, v + k)
```

**Predicted future**

```
process P(int k) ≡
recv(⋆, i) \cdot send(0, i + k)
```
Example program abstraction

Example MPI program

requires Future(P(k) \cdot \epsilon)
ensures Future(\epsilon)

void main(int k):
    int v \leftarrow MPI.Recv(\star)
    MPI.Send(0, v + k)

Predicted future

process P(int k) \equiv
recv(\star, i) \cdot send(0, i + k)

Does the program correctly execute its predicted future?
Example program abstraction

Example MPI program

```c
requires Future(P(k) · ε)  
ensures Future(ε)  
void main(int k):
    [P(k) · ε]  
    int v ← MPI.Recv(⋆)
    MPI.Send(0, v + k)
```

Predicted future

```text
process P(int k) ≡
recv(⋆, i) · send(0, i + k)
```

Does the program correctly execute its predicted future?
Example program abstraction

**Example MPI program**

```c
requires Future(P(k) · ε)
ensures Future(ε)
void main(int k):
  [P(k) · ε]
  [recv(⋆, i) · send(0, i + k) · ε]
  int v ← MPI_Recv(⋆)
  MPI_Send(0, v + k)
```

**Predicted future**

```
process P(int k) ≡
recv(⋆, i) · send(0, i + k)
```

Does the program correctly execute its predicted future?
Example program abstraction

Example MPI program

```c
requires Future(P(k) \cdot \epsilon)
ensures Future(\epsilon)
void main(int k):
    [P(k) \cdot \epsilon]
    [recv(\star, i) \cdot send(0, i + k) \cdot \epsilon]
    int v \leftarrow MPI.Recv(\star)
    [send(0, v + k) \cdot \epsilon]
    MPI.Send(0, v + k)
```

Predicted future

```c
process P(int k) \equiv
    recv(\star, i) \cdot send(0, i + k)
```

Does the program correctly execute its predicted future?
Example program abstraction

Example MPI program

```plaintext
requires Future(P(k) \cdot \epsilon)
ensures Future(\epsilon)
void main(int k):
    [P(k) \cdot \epsilon]
    [recv(\star, i) \cdot send(0, i + k) \cdot \epsilon]
    int v ← MPIRecv(\star)
    [send(0, v + k) \cdot \epsilon]
    MPI_Send(0, v + k)
    [\epsilon]
```

Predicted future

```plaintext
process P(int k) ≡
recv(\star, i) \cdot send(0, i + k)
```

Does the program correctly execute its predicted future?
Example program abstraction

**Example MPI program**

```plaintext
requires Future(P(k) · ε)
ensures Future(ε)
void main(int k):
  [P(k) · ε]
  [recv(⋆, i) · send(0, i + k) · ε]
  int ν ← MPI_Recv(⋆)
  [send(0, ν + k) · ε]
  MPI_Send(0, ν + k)
  [ε]
```

**Predicted future**

```plaintext
process P(int k) ≡
recv(⋆, i) · send(0, i + k)
```

- Does the program correctly execute its predicted future? 
  yes!
Tool support

```c
requires ... 
requires Future(P(k) \cdot F) 
ensures Future(F) 

void main(int k):
    int v ← MPI_Recv(*)
    MPI_Send(0, v + k)
```
Tool support

```c
requires \ldots
requires Future(P(k) \cdot F)
ensures Future(F)
void main(int k):
    int v ← MPI_Recv(*)
    MPI_Send(0, v + k)
```

Verdict
Does the program correctly executes the future process?

Hoare logic reasoning

\begin{align*}
\text{P}(k_1) \parallel \text{P}(k_2) \parallel \ldots \parallel \\
\text{Network}
\end{align*}

Temporal properties

Final verdict
Either pass or fail
Tool support

```
void main(int k):
    int v ← MPI_Recv(*)
    MPI_Send(0, v + k)
```

process P(int k) ≡
recv(*, i) · send(0, i + k)

1. requires · · · requires Future(P(k) · F)
   ensures Future(F)

2. Verdict
   Does the program correctly executes the future process?

Wytse Oortwijn, FMT, University of Twente
Tool support

\[
\text{process } P(\text{int } k) \equiv \\
\text{recv}(\ast, i) \cdot \text{send}(0, i + k)
\]

Verdict
Does the program correctly executes the future process?

Hoare logic reasoning
Tool support

process $P(\text{int } k) \equiv$

$\text{recv}(\ast, i) \cdot \text{send}(0, i + k)$

Verdict

Does the program correctly executes the future process?

Hoare logic reasoning

requires $\ldots$

requires $\text{Future}(P(k) \cdot F)$

ensures $\text{Future}(F)$

void main(int k):

\[
\begin{align*}
\text{int } v & \leftarrow \text{MPI.Recv}(\ast) \\
\text{MPI.Send}(0, v + k)
\end{align*}
\]

$P(k_1) \parallel P(k_2) \parallel \cdots \parallel \text{Network}$
Tool support

\[
\text{process } P(\text{int } k) \equiv \\
\text{recv}(\star, i) \cdot \text{send}(0, i + k)
\]

1. \text{void main(int } k\text{):} \\
   \hspace{1cm} \text{int } v \leftarrow \text{MPI.Recv}(\star) \\
   \hspace{1cm} \text{MPI.Send}(0, v + k)

2. \text{requires } \cdots \\
   \text{requires } \text{Future}(P(k) \cdot F) \\
   \text{ensures } \text{Future}(F) \\

3. \text{Verdict} \\
   \text{Does the program correctly executes the future process?} \\
   \text{Hoare logic reasoning}

4. \text{P}(k_1) \parallel \text{P}(k_2) \parallel \cdots \parallel \text{Network}

\text{MCRL2} \\
\text{analysing system behaviour}
Tool support

process P(int k) ≡
recv(*, i) · send(0, i + k)

void main(int k):
    int v ← MPI.Recv(*)
    MPI.Send(0, v + k)

requires · · ·
requires Future(P(k) · F)
ensures Future(F)

P(k_1) || P(k_2) || · · · || Network

Verdict
Does the program correctly executes the future process?
Hoare logic reasoning

Temporal properties

Wytse Oortwijn, FMT, University of Twente
Static Verification of Message Passing Programs 9 / 9
Introduction

Tool support

process $P(int \ k) \equiv$
recv$(\ast, \ i) \cdot send(0, \ i + k)$

requires ... requires Future($P(k) \cdot F$)
ensures Future($F$)
void main(int $F$):
  int $v \leftarrow$ MPI_Receiv$(\ast)$
  MPI_Send$(0, \ v + k)$

$P(k_1) \parallel P(k_2) \parallel \ldots \parallel$ Network

Verdict
Does the program correctly executes the future process?

Hoare logic reasoning

Temporal properties

MCRCL$2$
analysing system behaviour

Final verdict
Either pass or fail

Wytse Oortwijn, FMT, University of Twente

Static Verification of Message Passing Programs

9 / 9
Tool support

```c
void main(int k):
    int v ← MPI_Recv(*)
    MPI_Send(0, v + k)
```

**Process**

\[ P(int \ k) \equiv \]
\[ \text{recv}(*, i) \cdot \text{send}(0, i + k) \]

**Requires**
- \( P(k_1) \parallel P(k_2) \parallel \cdots \parallel \text{Network} \)

**Future**

\[ \text{Future}(P(k) \cdot F) \]

**Ensures**

\[ \text{Future}(F) \]

**Verdict**

Does the program correctly executes the future process?

**Hoare logic reasoning**

**Final verdict**

Either pass or fail