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

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

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

int value $= 0$;

requires $n > 0$;
void `increase`(**int** n):
    value $\leftarrow$ value $+$ n;

Writing software correctly is hard

Motivational examples

$400 million Pentium bug

THERAC-25 radiation therapy machine

Floyd & Hoare: Hoare logic

int value = 0;

requires $n > 0$;
ensures $value = \textbf{old}(value) + n$;
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

int value = 0;

requires $n > 0$;
ensures $value = \textbf{old}(value) + n$;
void `increase`(**int** n):
    value $\leftarrow$ 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

int value = 0;

requires $n > 0$;
ensures $value = \textbf{old}(value) + n$;
void `increase`(**int** n):
    value ← value + n;

Hoare triple reasoning

$$\{P\}S\{Q\}$$

$$\frac{\{P\}S\{Q\} \quad \{Q\}T\{R\}}{\{P\}S; T\{R\}}$$

etc...

Static verification

Hoare logic extensions

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

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

Message Passing Interface

The MPI standard

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

Message Passing Interface

The MPI standard

- **Sending a message:**
  $\texttt{MPI\_Send}(j, m)$
- **Receiving a message:**
  $m := \texttt{MPI\_Recv}(j)$

Message Passing Interface

The MPI standard

- **Sending a message:**
  $\texttt{MPI\_Send}(j, m)$
- **Receiving a message:**
  $m := \texttt{MPI\_Recv}(j)$
- **Broadcasting a message:**
  $\texttt{MPI\_Bcast}(m)$
- **etc...**

Message Passing Interface

The MPI standard

- **Sending a message:** $\texttt{MPI\_Send}(j, m)$
- **Receiving a message:** $m := \texttt{MPI\_Recv}(j)$
- **Broadcasting a message:** $\texttt{MPI\_Bcast}(m)$
- **etc...**

Challenges

1. Message exchanges are often **concurrent**.

Message Passing Interface

The MPI standard

- **Sending a message:** $\texttt{MPI\_Send}(j, m)$
- **Receiving a message:** $m := \texttt{MPI\_Recv}(j)$
- **Broadcasting a message:** $\texttt{MPI\_Bcast}(m)$
- **etc...**

Challenges

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

Message Passing Interface

The MPI standard

- **Sending a message:** $\texttt{MPI\_Send}(j, m)$
- **Receiving a message:** $m := \texttt{MPI\_Recv}(j)$
- **Broadcasting a message:** $\texttt{MPI\_Bcast}(m)$
- **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:** $\texttt{MPI\_Send}(j, m)$
- **Receiving a message:** $m := \texttt{MPI\_Recv}(j)$
- **Broadcasting a message:** $\texttt{MPI\_Bcast}(m)$
- **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?

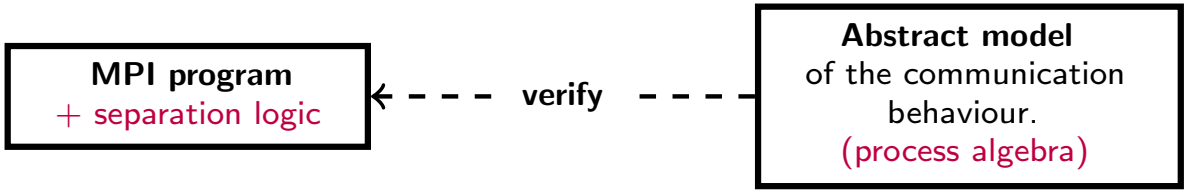
MPI program
+ separation logic

How to reason about distributed programs?

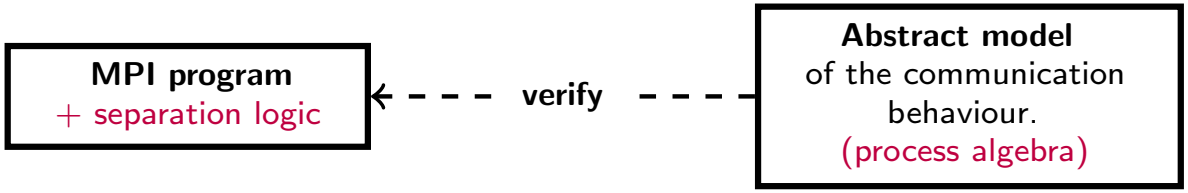
MPI program
+ separation logic

Abstract model
of the communication behaviour.
(process algebra)

How to reason about distributed programs?

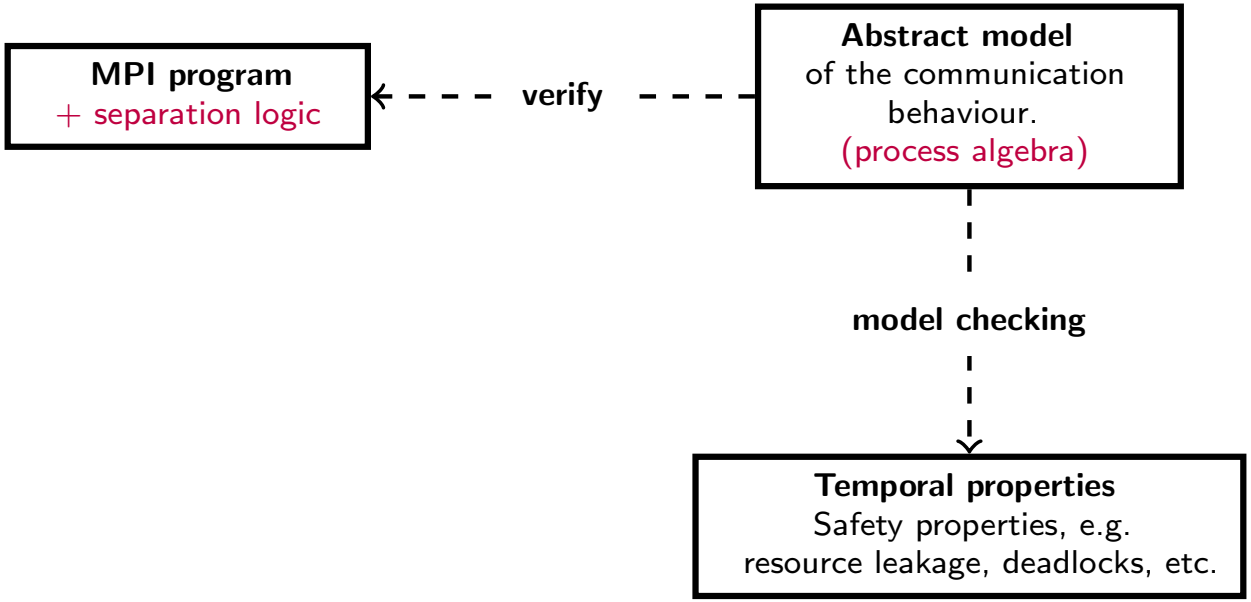


How to reason about distributed programs?

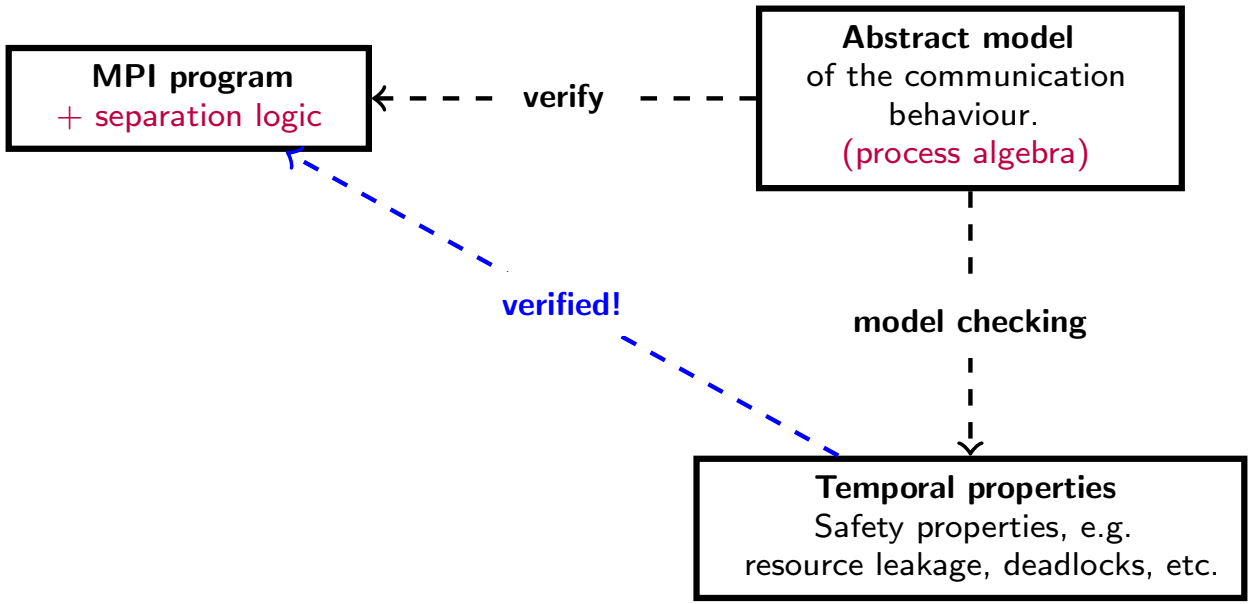


Temporal properties
Safety properties, e.g.
resource leakage, deadlocks, etc.

How to reason about distributed programs?



How to reason about distributed programs?



mCRL2: Process algebra

Example vending machine

$$\textbf{process}\ \text{Machine}() \equiv \textbf{coin} \cdot$$

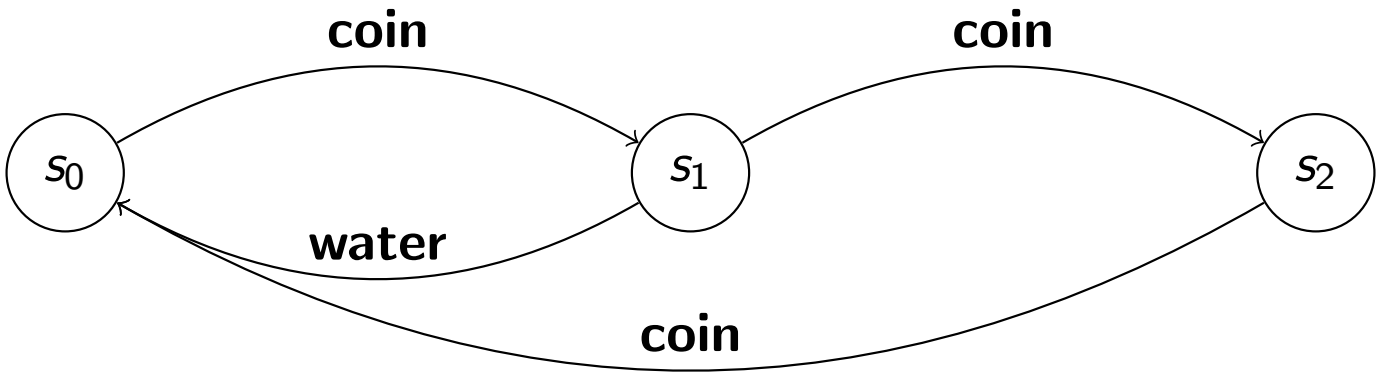
$$(\textbf{water} \cdot \text{Machine}() + \textbf{coin} \cdot \textbf{cola} \cdot \text{Machine}())$$

mCRL2: Process algebra

Example vending machine

$$\textbf{process}\ \text{Machine}() \equiv \textbf{coin} \cdot$$

$$(\textbf{water} \cdot \text{Machine}() + \textbf{coin} \cdot \textbf{cola} \cdot \text{Machine}())$$



Abstracting MPI primitives

MPI primitives $\rightsquigarrow$ 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 $\rightsquigarrow$ 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 primitives $\rightsquigarrow$ corresponding actions

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

- action **send** : **int** $\times$ **msg**
- action **recv** : **int** $\times$ **msg**
- action **bcast** : **msg**
- action **barrier**

Abstracting MPI primitives

MPI primitives $\rightsquigarrow$ corresponding actions

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

- action **send** : **int** $\times$ **msg**
- action **recv** : **int** $\times$ **msg**
- action **bcast** : **msg**
- action **barrier**

Finding a correspondence: Hoare-triple reasoning

Abstracting MPI primitives

MPI primitives $\rightsquigarrow$ corresponding actions

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

- action **send** : **int** $\times$ **msg**
- action **recv** : **int** $\times$ **msg**
- action **bcast** : **msg**
- action **barrier**

Finding a correspondence: Hoare-triple reasoning

$$\overline{\{\textbf{send}(i, m) \cdot \mathrm{F}\}\texttt{MPI\_Send}(i, m)\{\mathrm{F}\}}$$

Abstracting MPI primitives

MPI primitives $\rightsquigarrow$ corresponding actions

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

- action **send** : **int** $\times$ **msg**
- action **recv** : **int** $\times$ **msg**
- action **bcast** : **msg**
- action **barrier**

Finding a correspondence: Hoare-triple reasoning

$$\frac{}{\{\textbf{send}(i, m) \cdot F\}\texttt{MPI\_Send}(i, m)\{F\}} \qquad \frac{}{\{\textbf{recv}(i, v) \cdot F\}v := \texttt{MPI\_Recv}(i)\{F\}}$$

$$\frac{}{\{\textbf{bcast}(m) \cdot F\}\texttt{MPI\_Bcast}(m)\{F\}} \qquad \frac{}{\{\textbf{barrier}() \cdot F\}\texttt{MPI\_Barrier}()\{F\}}$$

Example program abstraction

Example MPI program

void `main`(**int** k):
    int $v \leftarrow$ `MPI_Recv`($\star$)
    `MPI_Send`(0, $v + k$)

Example program abstraction

Example MPI program

void `main`(**int** k):
    int $v \leftarrow$ `MPI_Recv`($\star$)
    `MPI_Send`(0, $v + k$)

Predicted future

$$\textbf{process } \mathrm{P}(\textbf{int } k) \equiv$$

$$\textbf{recv}(\star, i) \cdot \textbf{send}(0, i + k)$$

Example program abstraction

Example MPI program

requires Future(P(k) · ϵ)
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) · **send**(0, $i + k$)

Example program abstraction

Example MPI program

requires Future(P(k) $\cdot$ ϵ)
ensures Future(ϵ)
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, $i + k$)

Example program abstraction

Example MPI program

requires Future($\mathrm{P}(k) \cdot \epsilon$)
ensures Future(ϵ)
void `main`(**int** k):
    int $v \leftarrow$ `MPI_Recv`($\star$)
    `MPI_Send`(0, $v + k$)

Predicted future

$$\textbf{process } \mathrm{P}(\textbf{int } k) \equiv$$

$$\textbf{recv}(\star, i) \cdot \textbf{send}(0, i + k)$$

- Does the program correctly execute its predicted future?

Example program abstraction

Example MPI program

requires Future(P(k) $\cdot$ ϵ)
ensures Future(ϵ)
void `main`(**int** k):
    [P(k) $\cdot$ ϵ]
    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

requires Future($\mathrm{P}(k) \cdot \epsilon$)
ensures Future(ϵ)
void `main`(**int** k):
    $[\mathrm{P}(k) \cdot \epsilon]$
    $[\mathbf{recv}(\star, i) \cdot \mathbf{send}(0, i + k) \cdot \epsilon]$
    int $v \leftarrow$ `MPI_Recv`($\star$)
    `MPI_Send`(0, $v + k$)

Predicted future

$$\mathbf{process}\ \mathrm{P}(\mathbf{int}\ k) \equiv$$

$$\mathbf{recv}(\star, i) \cdot \mathbf{send}(0, i + k)$$

- Does the program correctly execute its predicted future?

Example program abstraction

Example MPI program

requires Future($\mathrm{P}(k) \cdot \epsilon$)
ensures Future(ϵ)
void `main`(**int** k):
    $[\mathrm{P}(k) \cdot \epsilon]$
    $[\mathbf{recv}(\star, i) \cdot \mathbf{send}(0, i + k) \cdot \epsilon]$
    int $v \leftarrow$ `MPI_Recv`($\star$)
    $[\mathbf{send}(0, v + k) \cdot \epsilon]$
    `MPI_Send`(0, $v + k$)

Predicted future

$$\mathbf{process}\ \mathrm{P}(\mathbf{int}\ k) \equiv$$

$$\mathbf{recv}(\star, i) \cdot \mathbf{send}(0, i + k)$$

- Does the program correctly execute its predicted future?

Example program abstraction

Example MPI program

requires Future($\mathrm{P}(k) \cdot \epsilon$)
ensures Future(ϵ)
void `main`(**int** k):
    $[\mathrm{P}(k) \cdot \epsilon]$
    $[\mathbf{recv}(\star, i) \cdot \mathbf{send}(0, i + k) \cdot \epsilon]$
    int $v \leftarrow$ `MPI_Recv`($\star$)
    $[\mathbf{send}(0, v + k) \cdot \epsilon]$
    `MPI_Send`(0, $v + k$)
    $[\epsilon]$

Predicted future

$$\mathbf{process}\ \mathrm{P}(\mathbf{int}\ k) \equiv$$

$$\mathbf{recv}(\star, i) \cdot \mathbf{send}(0, i + k)$$

- Does the program correctly execute its predicted future?

Example program abstraction

Example MPI program

requires Future($\mathrm{P}(k) \cdot \epsilon$)
ensures Future(ϵ)
void `main`(**int** k):
    $[\mathrm{P}(k) \cdot \epsilon]$
    $[\mathbf{recv}(\star, i) \cdot \mathbf{send}(0, i + k) \cdot \epsilon]$
    int $v \leftarrow$ `MPI_Recv`($\star$)
    $[\mathbf{send}(0, v + k) \cdot \epsilon]$
    `MPI_Send`(0, $v + k$)
    $[\epsilon]$

Predicted future

$$\mathbf{process}\ \mathrm{P}(\mathbf{int}\ k) \equiv$$

$$\mathbf{recv}(\star, i) \cdot \mathbf{send}(0, i + k)$$

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

Tool support

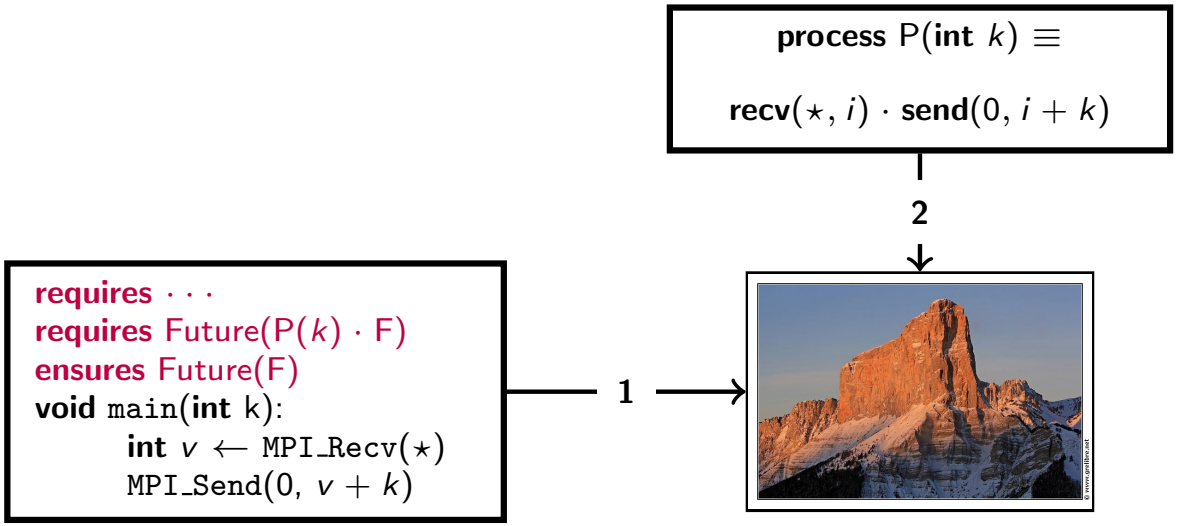
requires $\cdots$
requires Future(P(k) $\cdot$ F)
ensures Future(F)
void `main`(**int** k):
    int $v \leftarrow$ `MPI_Recv`($\star$)
    `MPI_Send`(0, $v + k$)

Tool support

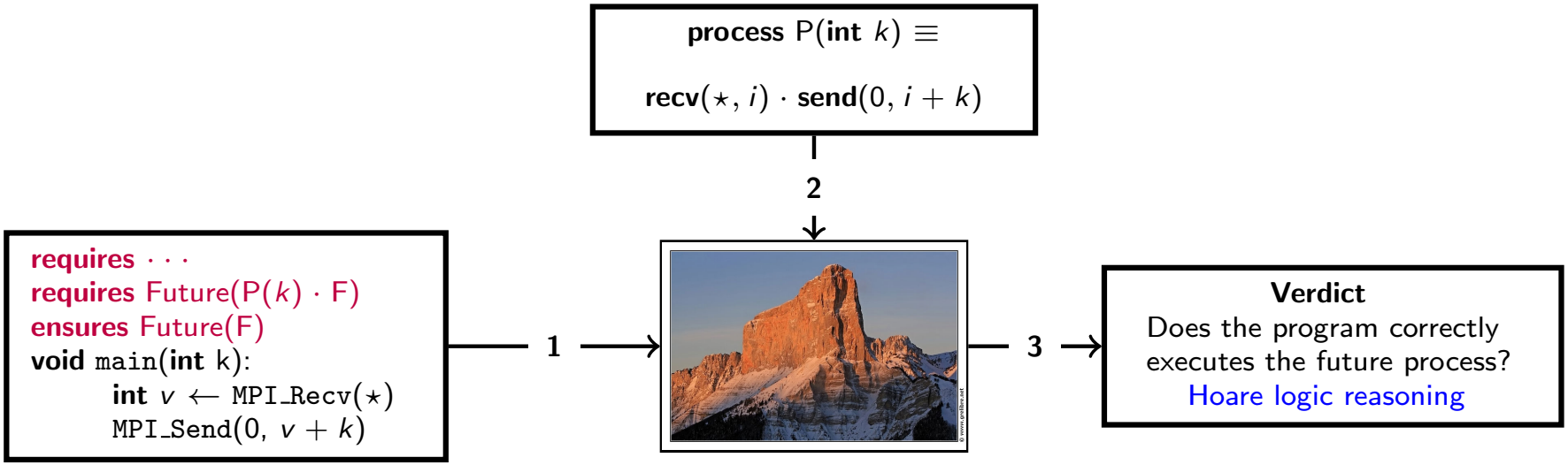
requires $\cdots$
requires Future(P(k) $\cdot$ F)
ensures Future(F)
void `main`(**int** k):
    int $v \leftarrow$ `MPI_Recv`($\star$)
    `MPI_Send`(0, $v + k$)

1

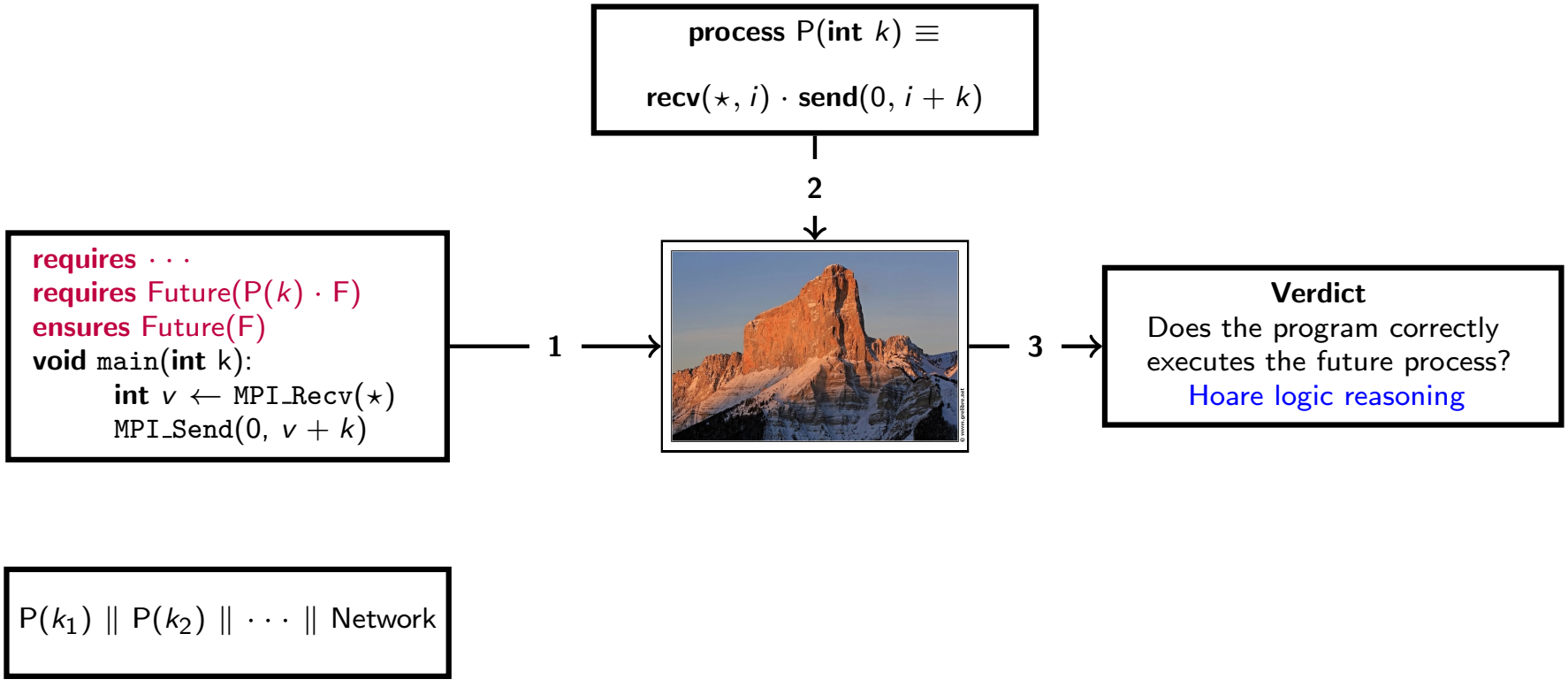
Tool support



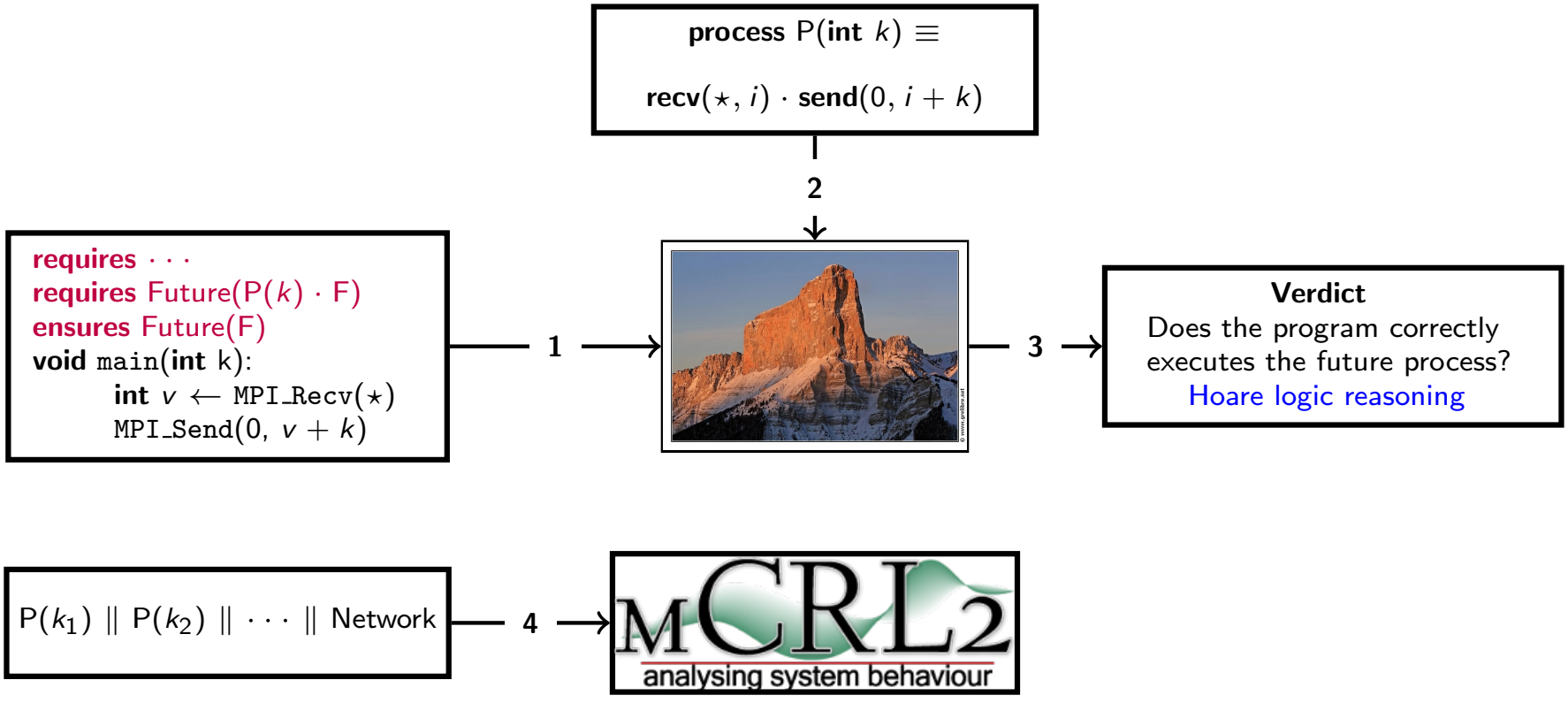
Tool support



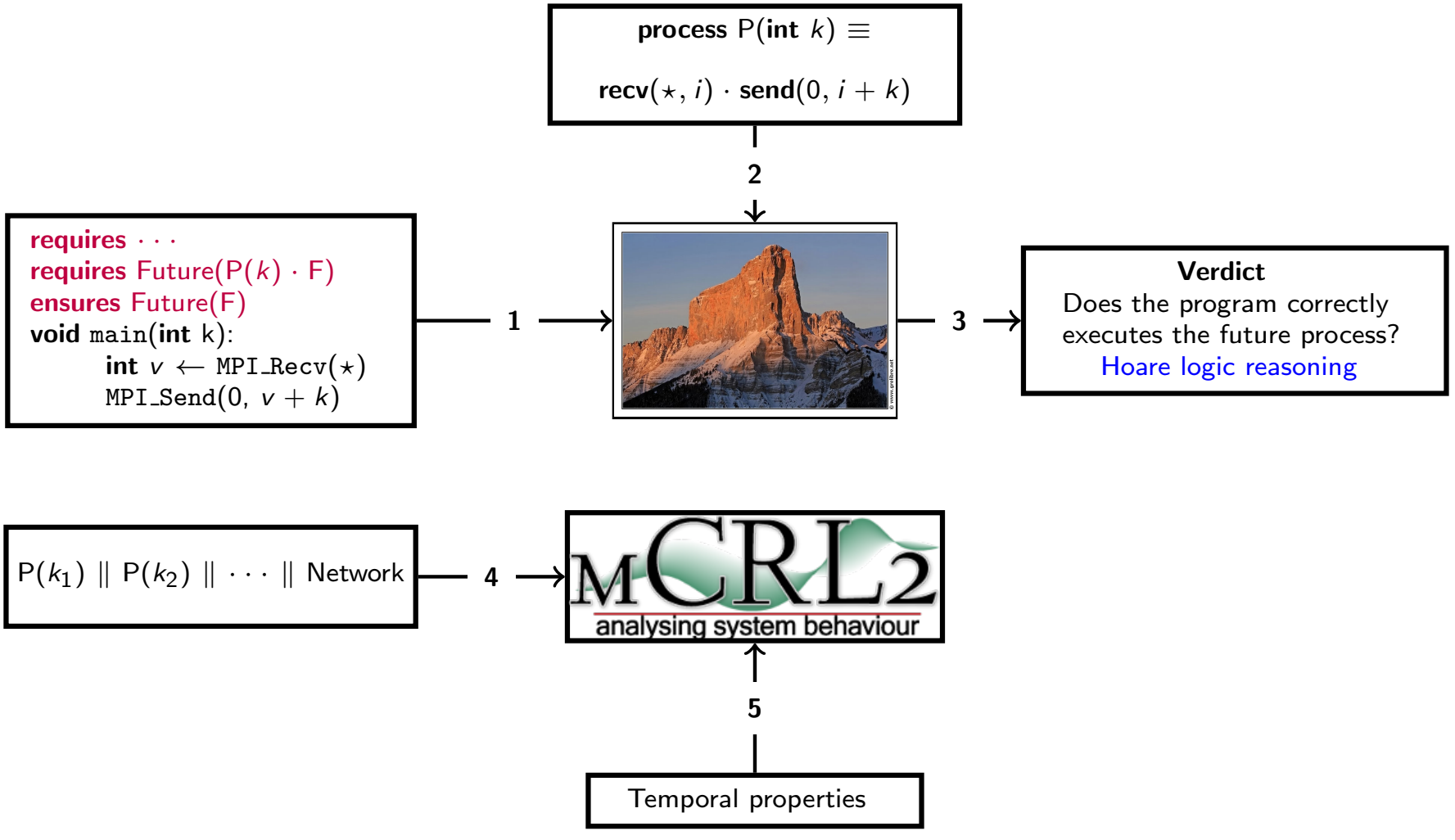
Tool support



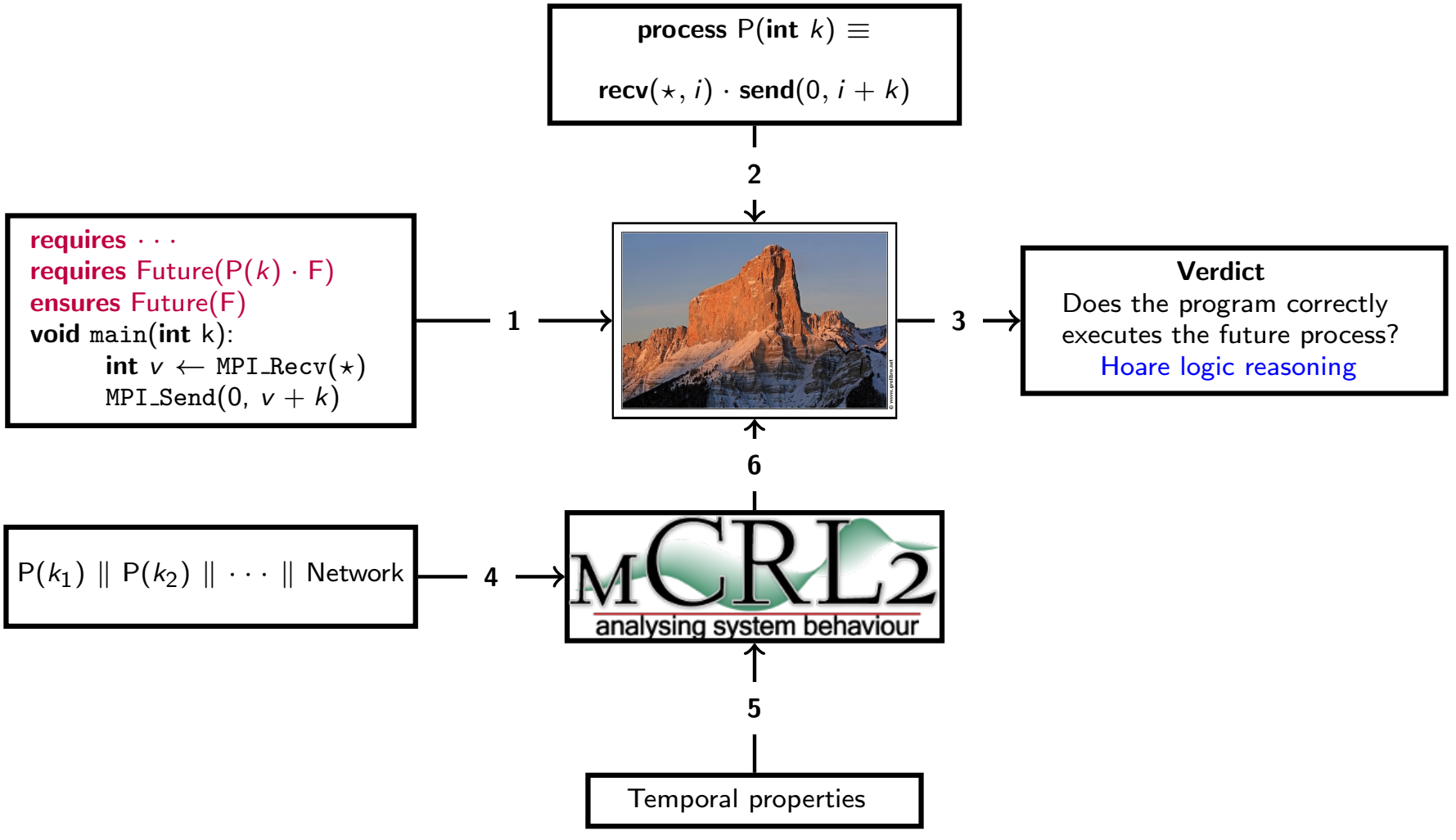
Tool support



Tool support



Tool support



Tool support