Verifying Concurrent Software

Wytse Oortwijn

Formal Methods and Tools, University of Twente

December 13, 2016

Concurrent programming is error-prone

Program Logics (Tony Hoare)

Hoare Triple

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

Hoare Triple

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

Formal meaning:

Hoare Triple

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

Formal meaning:

$\forall \sigma : \textsc{Store},$

Hoare Triple

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

Formal meaning:

$$\forall \sigma : \textsc{Store},\ \ \sigma \models P$$

Hoare Triple

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

Formal meaning:

$$\forall \sigma : \textsc{Store},\ \ \sigma \models P \land \langle S, \sigma \rangle \rightsquigarrow^* \langle \mathsf{done}, \sigma' \rangle$$

Hoare Triple

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

Formal meaning:

$$\forall \sigma : \textsc{Store},\ \ \sigma \models P \land \langle S, \sigma \rangle \rightsquigarrow^* \langle \mathsf{done}, \sigma' \rangle \rightarrow \sigma' \models Q$$

Hoare Logic

$$\frac{}{\{P\}\ \textsc{skip}\ \{P\}}$$

Hoare Logic

$$\frac{}{\{P\}\ \text{SKIP}\ \{P\}} \qquad \frac{\{P\}\ S_1\ \{R\} \qquad \{R\}\ S_2\ \{Q\}}{\{P\}\ S_1; S_2\ \{Q\}}$$

Hoare Logic

$$\frac{}{\{P\}\ \text{SKIP}\ \{P\}} \qquad \frac{\{P\}\ S_1\ \{R\} \qquad \{R\}\ S_2\ \{Q\}}{\{P\}\ S_1; S_2\ \{Q\}}$$

$$\frac{\{P \wedge B\}\ S_1\ \{Q\} \qquad \{P \wedge \neg B\}\ S_2\ \{Q\}}{\{P\}\ \text{IF}\ B\ \text{THEN}\ S_1\ \text{ELSE}\ S_2\ \{Q\}}$$

Hoare Logic

$$\frac{}{\{P\}\ \textsc{skip}\ \{P\}} \qquad \frac{\{P\}\ S_1\ \{R\} \qquad \{R\}\ S_2\ \{Q\}}{\{P\}\ S_1; S_2\ \{Q\}}$$

$$\frac{\{P \wedge B\}\ S_1\ \{Q\} \qquad \{P \wedge \neg B\}\ S_2\ \{Q\}}{\{P\}\ \textsc{if}\ B\ \textsc{then}\ S_1\ \textsc{else}\ S_2\ \{Q\}}$$

$$\frac{\{I \wedge B\}\ S\ \{I\}}{\{I\}\ \textsc{while}\ B\ \textsc{do}\ S\ \{I \wedge \neg B\}}$$

Program Specifications (Rustan Leino)

void *method*(· · ·) {

    · · ·

}

Program Specifications (Rustan Leino)

requires precondition
void *method*(· · ·) {
    · · ·
}

Program Specifications (Rustan Leino)

requires precondition
ensures postcondition
void *method*(· · ·) {
    · · ·
}

Specification Example: Sorting

```
void sort(int[] a) {
    ...
}
```

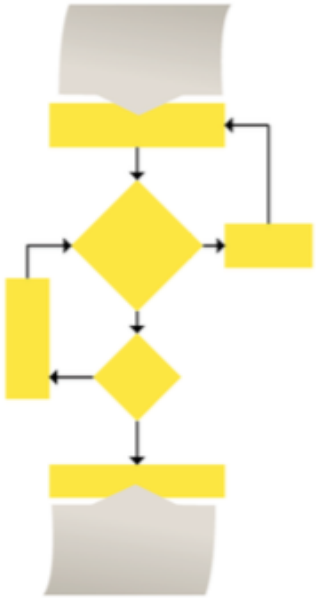
Specification Example: Sorting

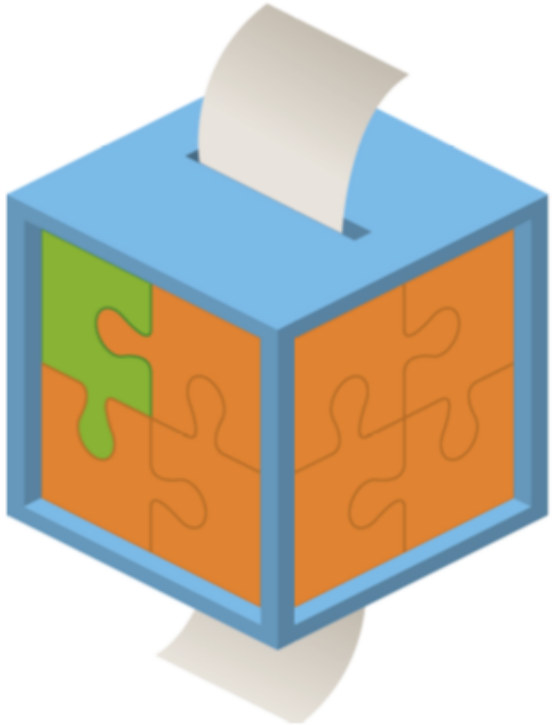
requires $a \neq \texttt{null}$
void *sort*(**int**[] *a*) {
    ...
}

Specification Example: Sorting

requires $a \neq \texttt{null}$
ensures $\forall i : \text{INT},\ (0 \leq i < a.length - 1) \rightarrow a[i] \leq a[i+1]$
void *sort*(**int**[] *a*) {
    ...
}

Static Verifiers





Separation Logic (John Reynolds)

Separation Logic: Pointers

```
class Counter {
    int count;

    public void incr(int x) {
        count = count + x;
    }
}
```

Separation Logic: Pointers

```
class Counter {
    int count;

    requires this.count → ?v
    public void incr(int x) {
        count = count + x;
    }
}
```

Separation Logic: Pointers

```
class Counter {
    int count;

    requires this.count → ?v
    ensures this.count → ?v + x
    public void incr(int x) {
        count = count + x;
    }
}
```

Separation Logic: Separating Conjunction

$$h, \sigma \models P_1 * P_2$$

Separation Logic: Separating Conjunction

$$h, \sigma \models P_1 * P_2$$

if and only if

Separation Logic: Separating Conjunction

$$h, \sigma \models P_1 * P_2$$

if and only if

$$h = h_1 \uplus h_2,$$

Separation Logic: Separating Conjunction

$$h, \sigma \models P_1 * P_2$$

if and only if

$$h = h_1 \uplus h_2, \text{ such that } h_1, \sigma \models P_1$$

Separation Logic: Separating Conjunction

$$h, \sigma \models P_1 * P_2$$

if and only if

$$h = h_1 \uplus h_2, \text{ such that } h_1, \sigma \models P_1 \text{ and } h_2, \sigma \models P_2$$

Concurrent Separation Logic (Peter O'Hearn)

$$\frac{}{\{P_1 * P_2\}\, S_1 \parallel S_2\, \{Q_1 * Q_2\}}$$

Concurrent Separation Logic (Peter O'Hearn)

$$\frac{\{P_1\}\, S_1\, \{Q_1\} \qquad \{P_2\}\, S_2\, \{Q_2\}}{\{P_1 * P_2\}\, S_1 \parallel S_2\, \{Q_1 * Q_2\}}$$

Separation Logic with Permissions (Boyland)

$$\frac{}{\{c.val \xrightarrow{1} ?v\}\, x = c.val + 1 \parallel y := c.val + 2\, \{c.val \xrightarrow{1} ?v\}}$$

Separation Logic with Permissions (Boyland)

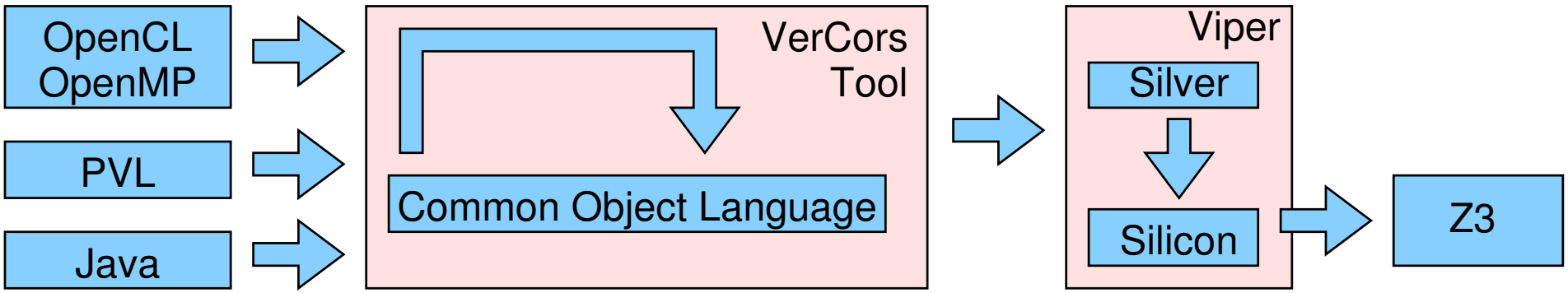
$$\frac{\{c.val \xrightarrow{\frac{1}{2}} ?v\}\, x = c.val + 1 \,\{c.val \xrightarrow{\frac{1}{2}} ?v\}}{\{c.val \xrightarrow{1} ?v\}\, x = c.val + 1 \parallel y := c.val + 2 \,\{c.val \xrightarrow{1} ?v\}}$$

Separation Logic with Permissions (Boyland)

$$\frac{\{c.val \xrightarrow{\frac{1}{2}} ?v\}\, x = c.val + 1 \,\{c.val \xrightarrow{\frac{1}{2}} ?v\} \qquad \{c.val \xrightarrow{\frac{1}{2}} ?v\}\, y = c.val + 2 \,\{c.val \xrightarrow{\frac{1}{2}} ?v\}}{\{c.val \xrightarrow{1} ?v\}\, x = c.val + 1 \parallel y := c.val + 2 \,\{c.val \xrightarrow{1} ?v\}}$$

The VerCors Toolset

The VerCors Toolset: Overview



Example: Verifying Loop Parallelisations

$$\frac{}{\textbf{for}(\textbf{int } i = 0; i < N; i{+}{+})\{S_i\}}$$

Example: Verifying Loop Parallelisations

$$\frac{}{\{*_{j=0}^{N-1} P_j\}\ \textbf{for}(\textbf{int}\ i = 0; i < N; i{+}{+})\{S_i\}\ \{*_{j=0}^{N-1} Q_j\}}$$

Example: Verifying Loop Parallelisations

$$\frac{\{P_0\}\ S_0\ \{Q_0\} \qquad \cdots \qquad \{P_{N-1}\}\ S_{N-1}\ \{Q_{N-1}\}}{\{*_{j=0}^{N-1} P_j\}\ \textbf{for}(\textbf{int}\ i = 0; i < N; i{+}{+})\{S_i\}\ \{*_{j=0}^{N-1} Q_j\}}$$

Example: Verifying GPGPU Kernels

kernel void add(**global int** A, **global int** B) {
    $A[tid] = 3 * B[tid]$;
}

Example: Verifying GPGPU Kernels

requires $A[tid] \xrightarrow{1} - * B[tid] \xrightarrow{\frac{1}{2}} -$

kernel void add(**global int** A, **global int** B) {

$\quad A[tid] = 3 * B[tid];$

}

Example: Verifying GPGPU Kernels

requires $A[tid] \xrightarrow{1} - * B[tid] \xrightarrow{\frac{1}{2}} -$

ensures $A[tid] \xrightarrow{1} 3B[tid] * B[tid] \xrightarrow{\frac{1}{2}} -$

kernel void *add*(**global int** A, **global int** B) {

$A[tid] = 3 * B[tid];$

}

Example: Model-based Verification

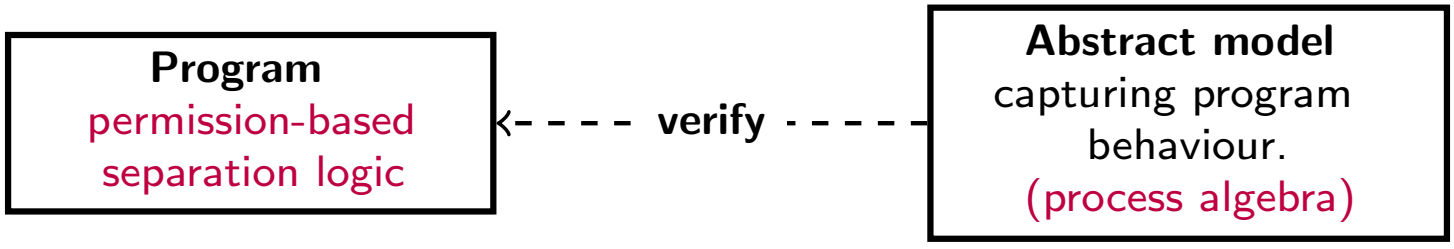
Program
permission-based
separation logic

Example: Model-based Verification

Program
permission-based
separation logic

Abstract model
capturing program
behaviour.
(process algebra)

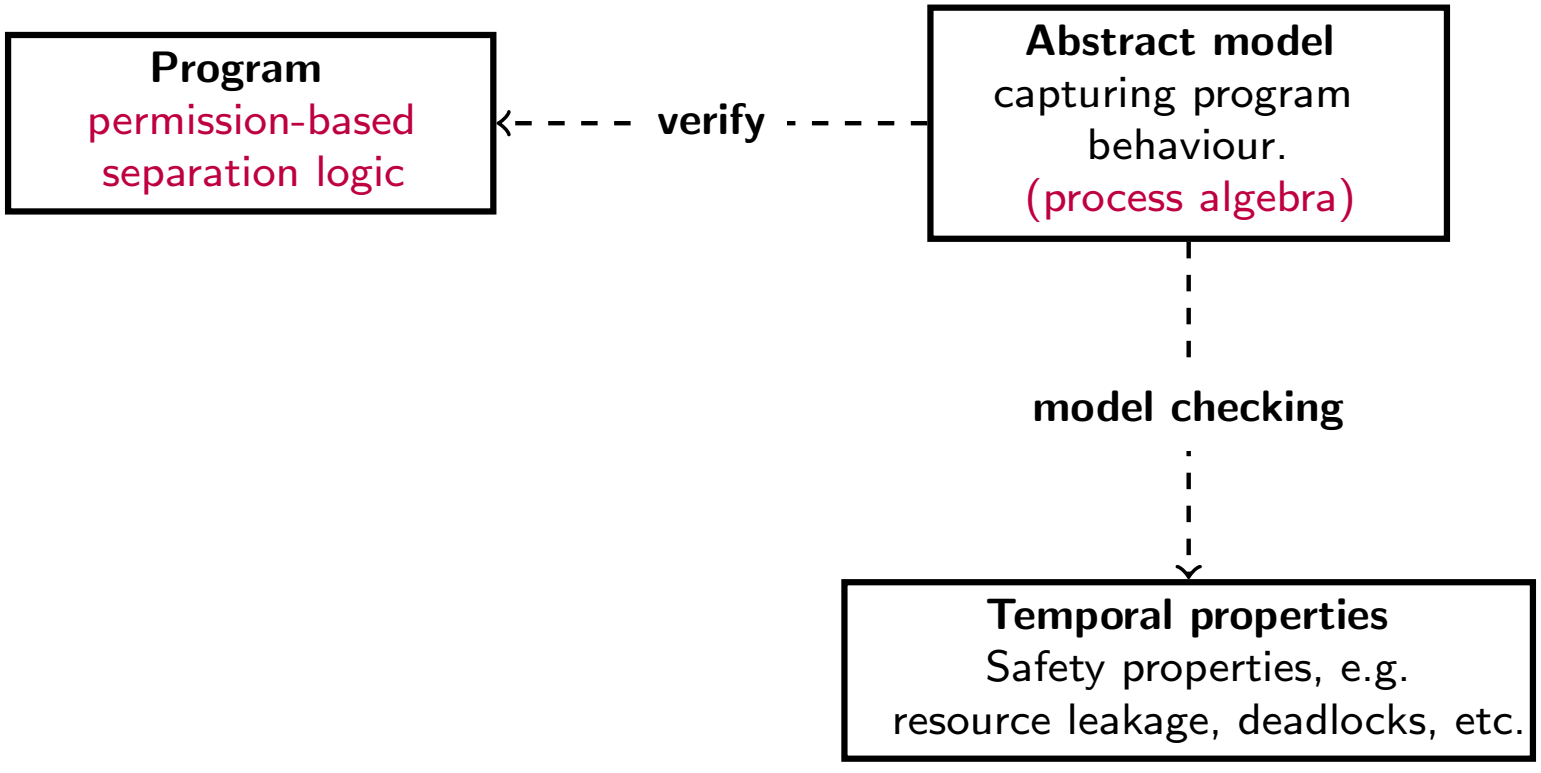
Example: Model-based Verification



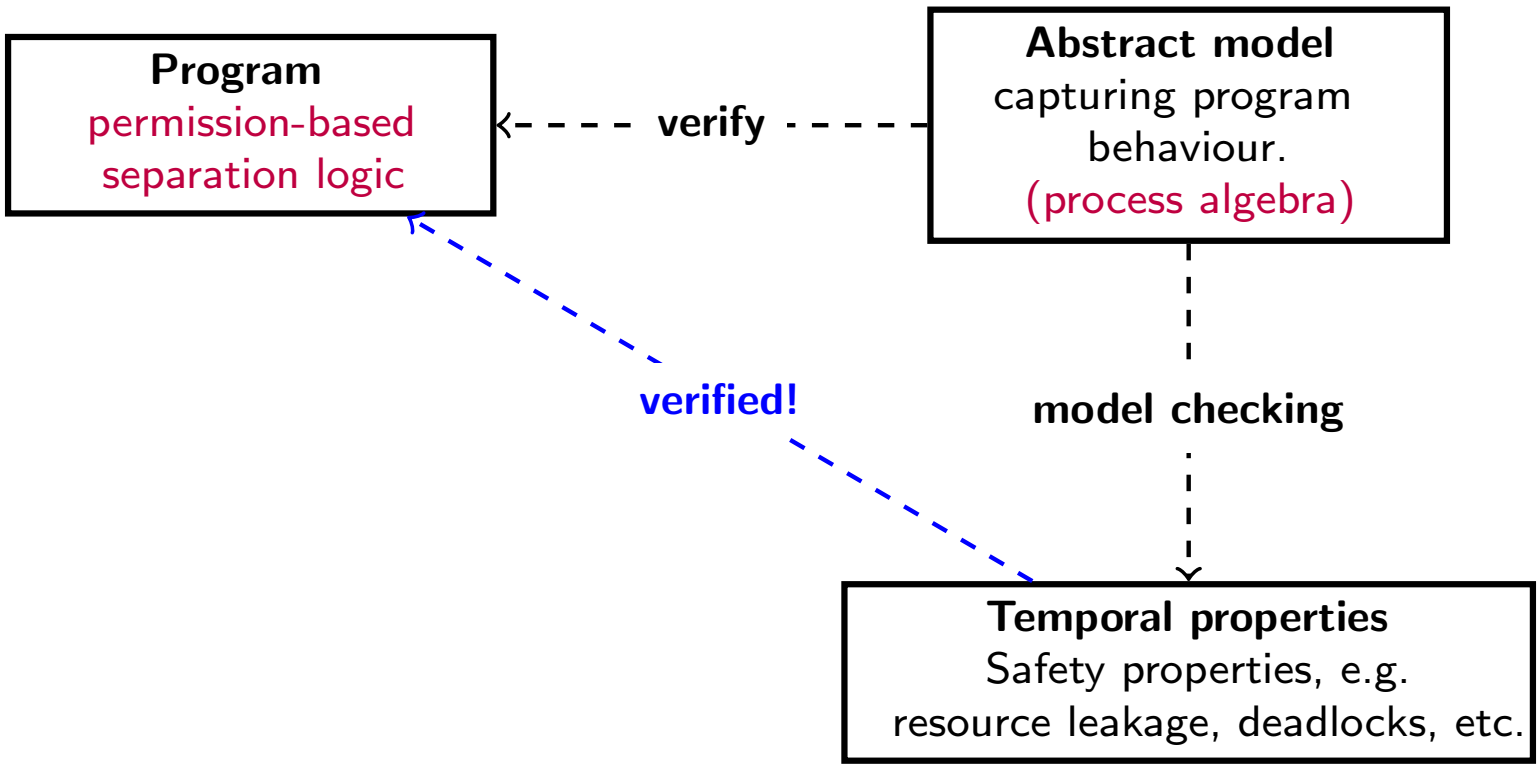
Example: Model-based Verification

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

Example: Model-based Verification



Example: Model-based Verification



Model-based verification: future directions

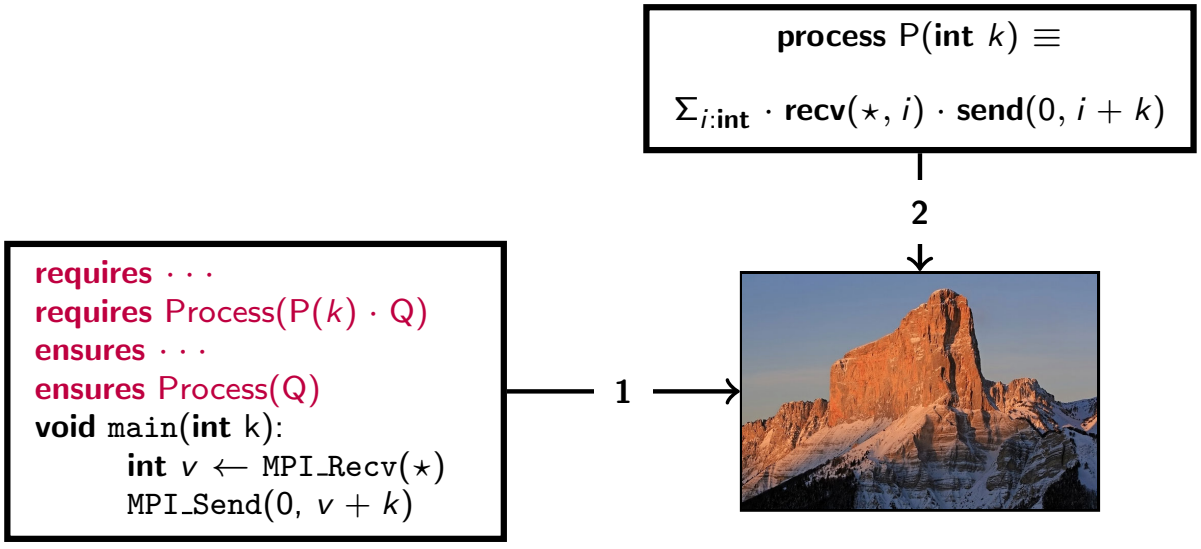
```
requires · · ·
requires Process(P(k) · Q)
ensures · · ·
ensures Process(Q)
void main(int k):
        int v ← MPI_Recv(⋆)
        MPI_Send(0, v + k)
```

Model-based verification: future directions

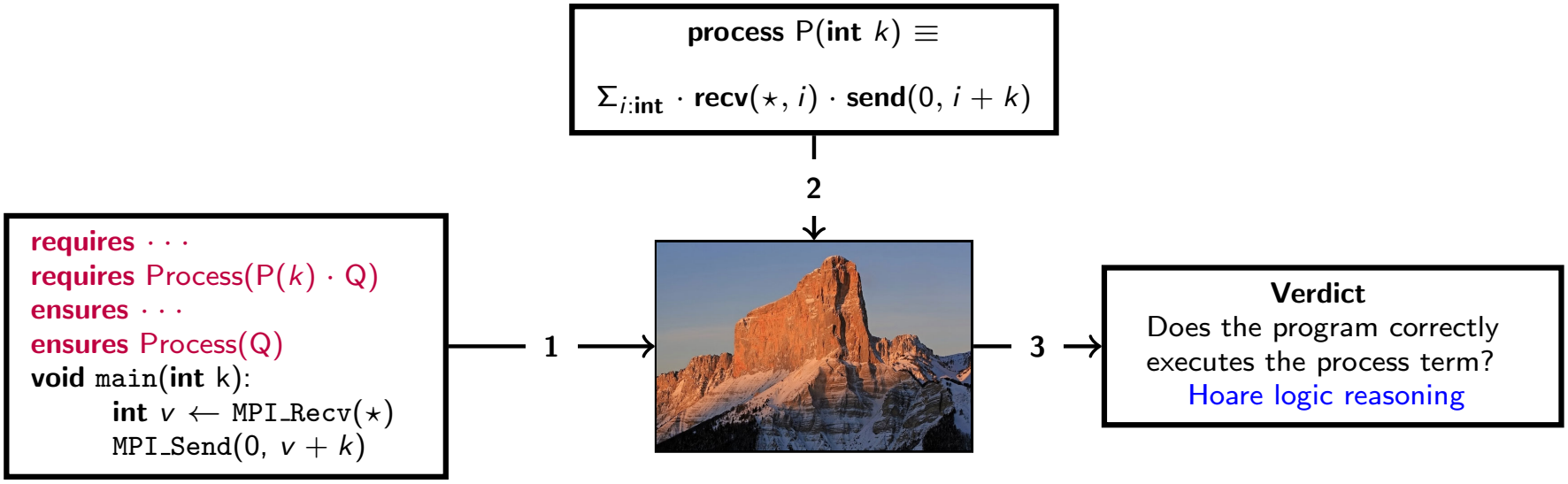
```
requires · · ·
requires Process(P(k) · Q)
ensures · · ·
ensures Process(Q)
void main(int k):
        int v ← MPI_Recv(⋆)
        MPI_Send(0, v + k)
```

1

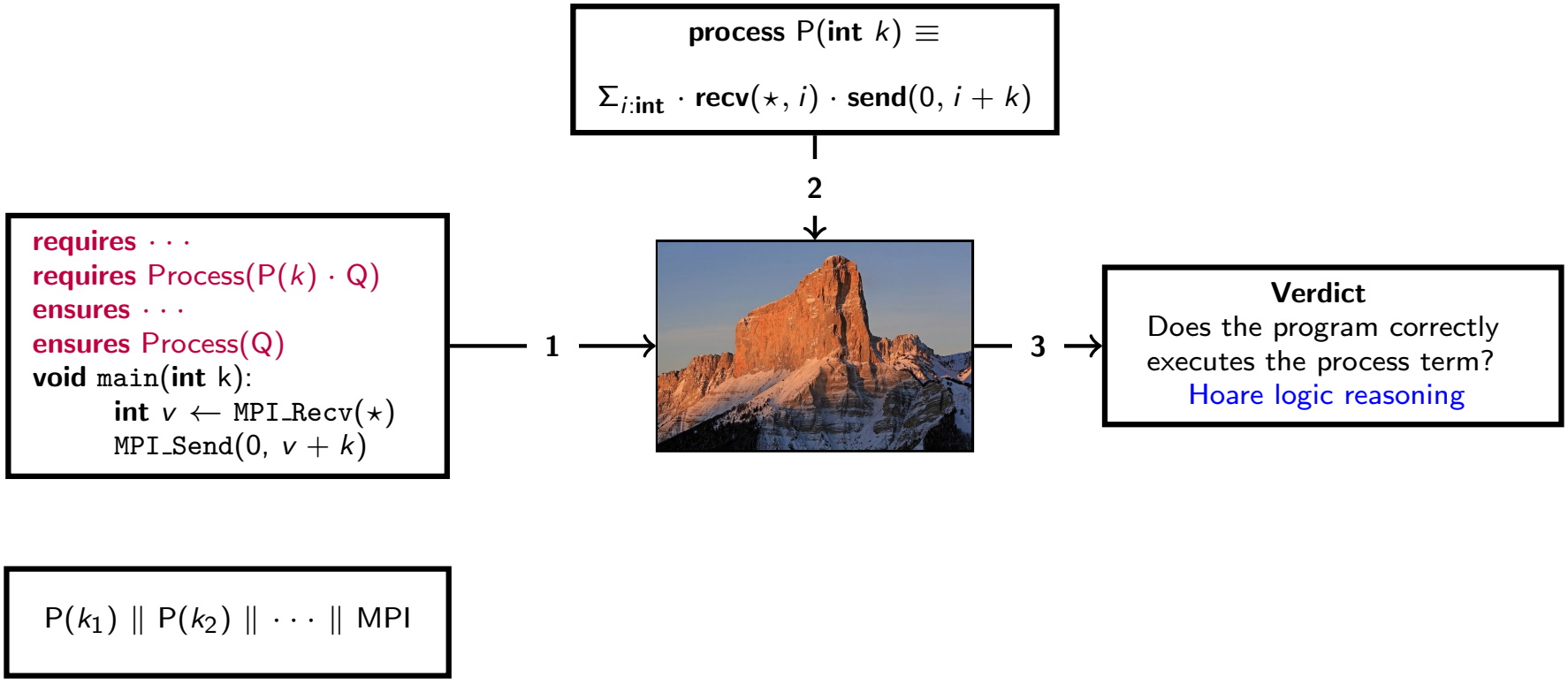
Model-based verification: future directions



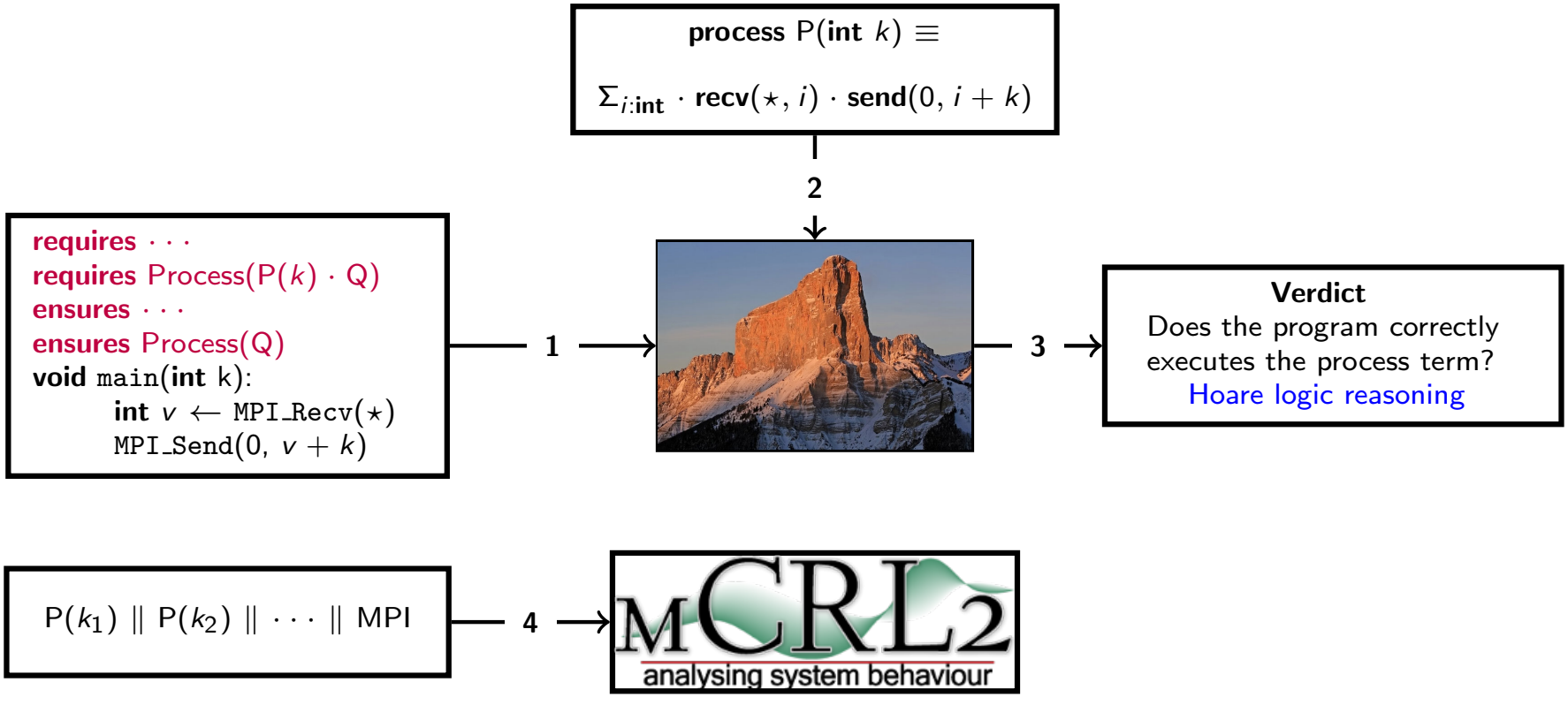
Model-based verification: future directions



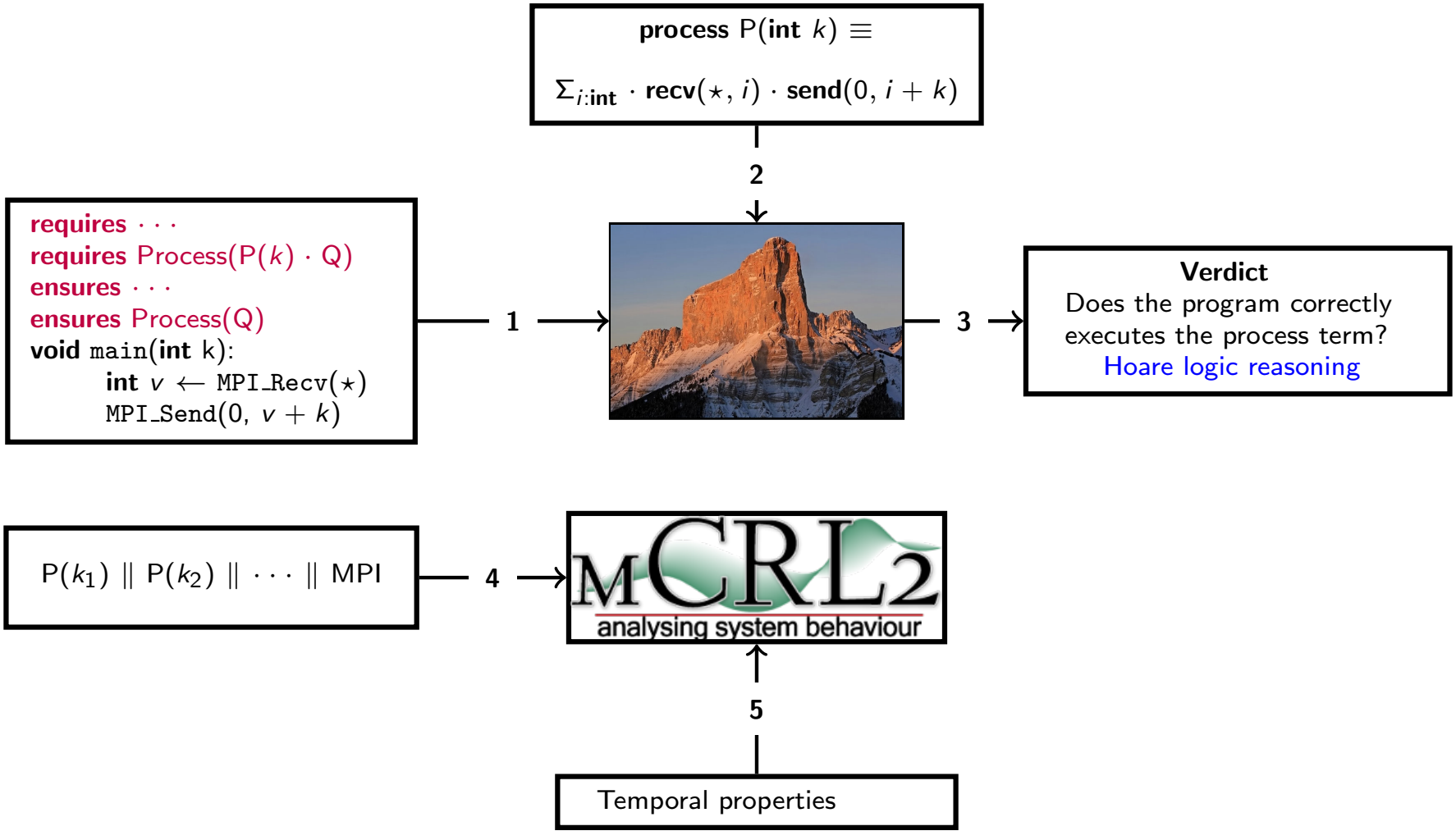
Model-based verification: future directions



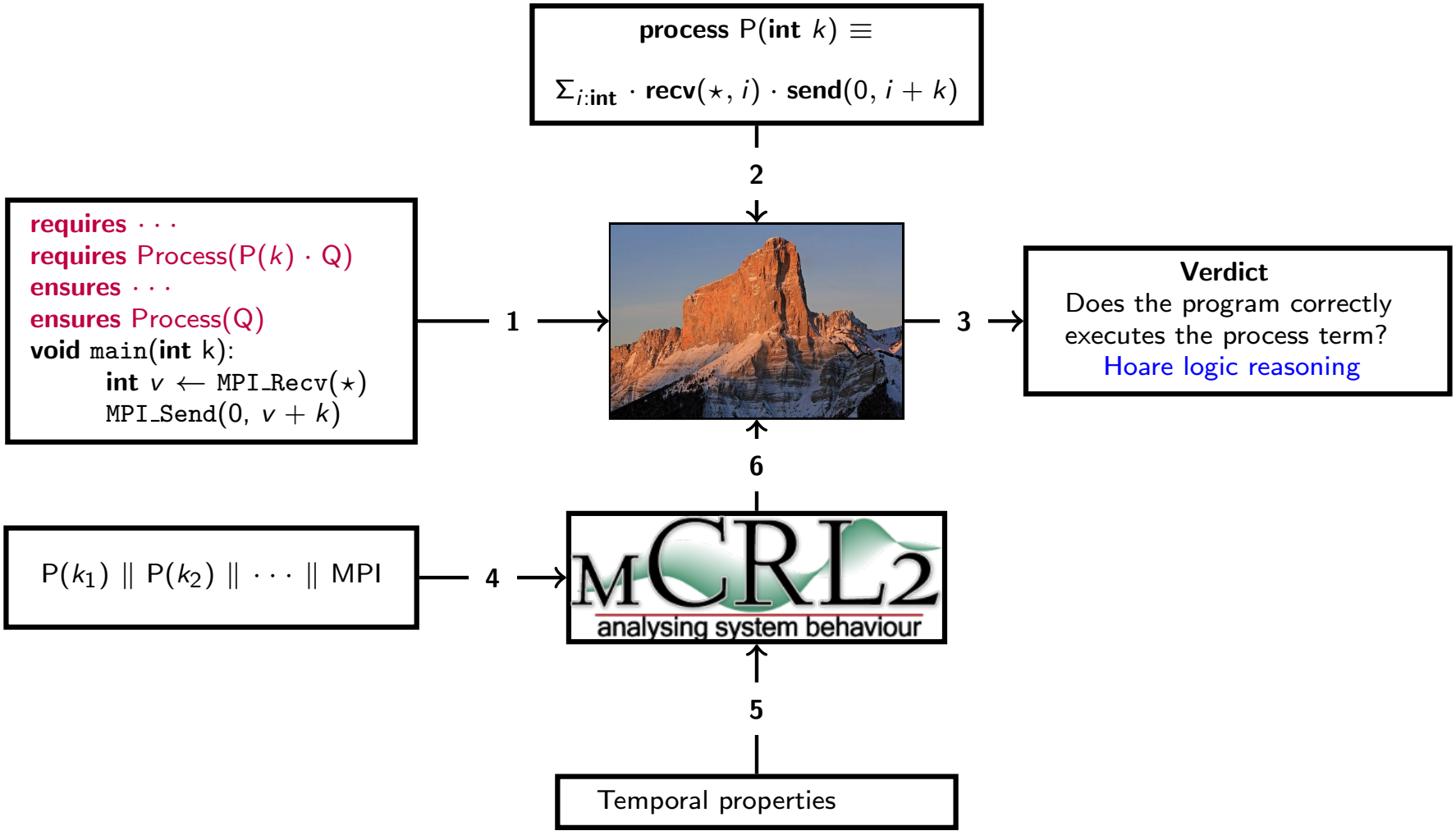
Model-based verification: future directions



Model-based verification: future directions



Model-based verification: future directions



Model-based verification: future directions