Future-based Analysis of Message Passing Programs

via static analysis and model checking

Wytse Oortwijn, Stefan Blom, and Marieke Huisman
Formal Methods and Tools - Department of Computer Science - University of Twente

CHALLENGE

*Statically proving **functional correctness** of message passing progams is very **hard**...*

Message exchanges are often **concurrent**, the number of proceses are often **unknown**, and the number of possible behaviours is often **infinite**.

MOTIVATION

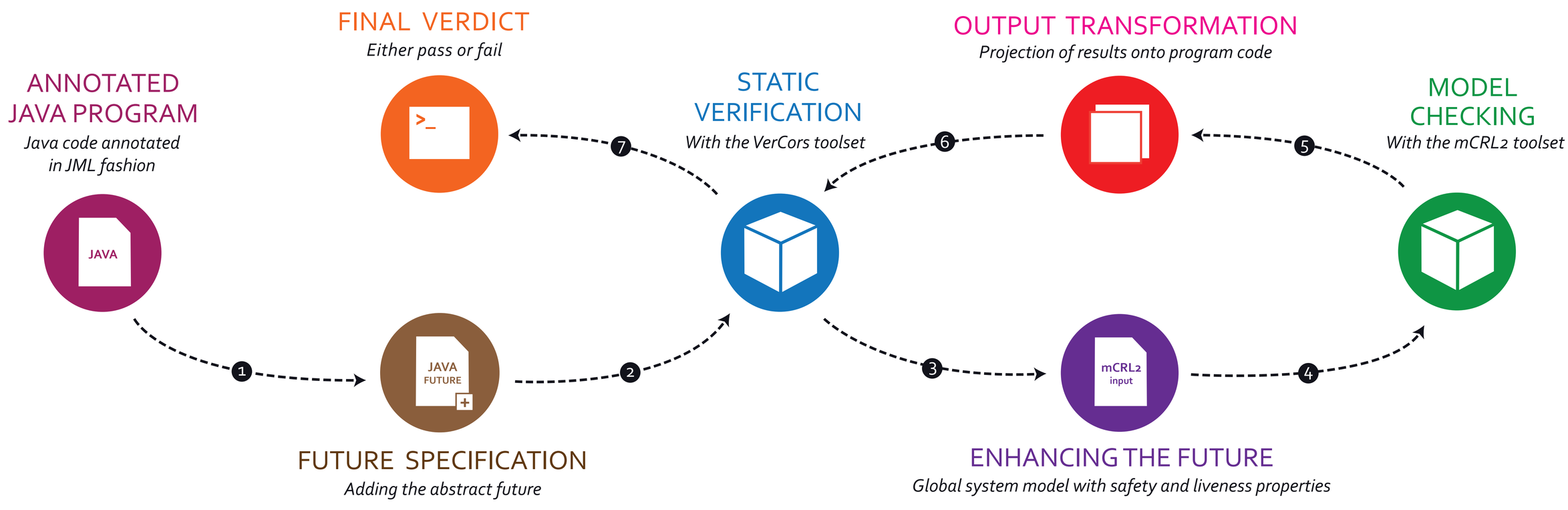
*...but having the ability to proof their correctness is **crucial**....*

Classical examples of software faults include the $400 million **Pentium bug** and the **THERAC-25**, which caused fatal injuries to numerous people.

CONTRIBUTION

*...we propose **future-based analysis** to verify the communicational aspects.*

We use **separation logic** to show local correctness and capture communicatical behaviour in abstract models, called **futures**, which we analyze to also proof **global** and **functional correctness**.



1 Adding a future specification

As input we consider Java MPJ programs, annotated with permission-based separation logic. In addition, we specify a future that predicts the communication behaviour of the program.

2 Static verification with VerCors

Future annotations are added to the program annotations to specify the correspondence between futures and the program code. To illustrate, an annotated example program is given below.

3 Transforming the future

If a correspondence can be proven with VerCors, the futures are transformed into input for mCRL2, otherwise a fail verdict is reported.

4 Adding the semantics of MPI

In addition, the futures are combined with an mCRL2 process that describes the semantics of MPI.

5 Transforming model checking results

The output of mCRL2 is analyzed and transformed to allow its projection onto the concrete source code.

6 Projecting model checking results

The output of mCRL2 is projected onto the source code, so that violations found after analyzing the futures can be linked to the corresponding function calls in the program code that caused the violation.

7 Final verdict

The VerCors toolset either successfully verifies the program with the given future specification, or indicates a violation and gives a report.

Example program

An example with 1 server and *N* clients. Each client sends an integer to the server. The server sums up and broadcasts all received integers. Annotated pseudocode is given below:

```
requires 0 ≤ n ≤ N
requires Future(Server(n,t)·F)
ensures Future(F)
def server(int n, int t):
  int N ← MPI_Size()
  if (n < N) then
    int x ← MPI_Recv(*)
    server(n+1, t+x)
  else MPI_Bcast(i, t)

requires Future(Client(v)·F)
ensures Future(F)
def client(int v):
  int i ← MPI_Rank()
  MPI_Send(0, v)
  int sum ← MPI_Recv(0)
```

Initially, the server starts as `server(0,0)` and each client i as `client(`v_i`)`. The program code is annotated with permission-based separation logic and futures.

Example future specification

The future annotations, which are mCRL2 process terms, describe the communication behaviour. For every MPI function we define corresponding actions. For example, for `MPI_Send`, `MPI_Recv`, and `MPI_Bcast` we may define:

```
action send, recv : rank × msg
action bcast : msg
```

These actions are used in futures to specify the communication behaviour. For example, the following two processes correspond to the example program:

```
process Server(int n, int t) ≡ (n < N) →
  sum x:msg · recv(*, x) · Server(n+1, t+x)
     ◇ bcast(t);

process Client(int v) ≡ send(0, v) ·
  sum x:msg · recv(0, x);
```

With these futures we predict that clients send their integer v to the server and receive a value x, and that the server receives N values and broadcasts the sum of all values . The futures are analyzed to prove functional and communication correctness.

Static verification + model checking

The VerCors toolset is used to find a correspondence between futures and program code via Hoare logic reasoning. By proving properties over futures we prove equivalent properties over the actual program. For example, we may use the following Hoare triple axioms:

$$[\text{send}]\ \frac{}{\{\text{send}(i,j,m)\cdot F\}\,\text{MPI\_Send}(i,j,m)\,\{F\}}$$

$$[\text{recv}]\ \frac{}{\{\text{recv}(i,j,m)\cdot F\}\,m \leftarrow \text{MPI\_Recv}(i,j)\,\{F\}}$$

$$[\text{bcast}]\ \frac{}{\{\text{bcast}(i,m)\cdot F\}\,\text{MPI\_Bcast}(i,m)\,\{F\}}$$

Afterwards, we use the mCRL2 toolset to analyze the futures, combined with a `Network` process that models the semantics of MPI. To illustrate, we may analyze:

$$\texttt{Server} \parallel (\underbrace{\texttt{Client} \parallel \cdots \parallel \texttt{Client}}_{N \text{ times}}) \parallel \texttt{Network}$$

With mCRL2 we check for communication correctness, e.g. validity of message exchanges, resource leakage, and other interesting safety and liveness properties.

w.h.m.oortwijn@utwente.nl
http://fmt.cs.utwente.nl/research

UNIVERSITY OF TWENTE.