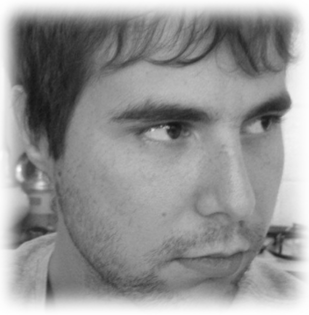
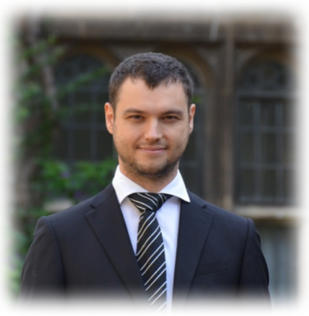
Reasoning about Concurrent Programs

Program Specification and Verification Group
Department of Computing
Imperial College London



Andrea Cerone
Research Associate



Emanuele D'Osualdo
Research Associate

Pedro da Rocha Pinto
Now in industry

Philippa Gardner
Leader

Gian Ntzik
Now in Amadeus

Azalea Raad
Now in MPI Germany

Julian Sutherland
PhD student

Shale Xiong
PhD student

Hoare Logic/Hoare Triple $\vdash \{P\}\ \mathbb{C}\ \{Q\}$

- First-order assertions P and Q describe the machine states before and after the sequential program $\mathbb{C}$.
- It cannot prove the program $\mathbb{C}$ terminates, but in the case of non-termination, the triple guarantees no crash.
- It cannot deal with reasoning about the heap.

Motivation $\mathbb{C}_1 \parallel \mathbb{C}_2$

Concurrency aims to increases performance. However, changes to a part of a concurrent program might "leak" to other part of the program and potentially modify the expected behaviour. Compared with traditional tests, formal verification offers higher guarantees, which is particularly important for critical systems.

Separation Logic (Sep.)

$$\frac{\vdash \{P\}\ \mathbb{C}\ \{Q\}}{\vdash \{P * R\}\ \mathbb{C}\ \{Q * R\}}\ \textsc{Frame}$$

- Hoare logic has difficulty guaranteeing locality of side-effects when aliasing is possible, for example the heap.
- Sep. Logic extends assertion grammar with a **separational conjunction**, $P * Q$, which asserts that the heap has two disjoint sub-heaps, each satisfying P and Q respectively.
- The frame rule allows localised reasoning.

Rely-guarantee (RG)

- Threads sharing resources in a "well-designed" concurrent program will modify the resources following a "contract". It allows an abstraction of the collaboration between threads.
- **Rely** (R) and **guarantee** (G) abstract the behaviour of the environment and the current thread respectively.

Concurrent Abstract Predicate (CAP)

- CAP introduces **abstract specifications** allowing clients and module implementations to be verified separately.
- CAP has a notion of **abstract separation**. For example on the right, the predicate $\texttt{Counter}\ (x, v, \pi)$ keeps track of a thread's contribution v to a counter at x and the permission π. When the permission is 1, then the contribution v is the true value of the counter.

CAP

$$\frac{\vdash \{P_1\}\ \mathbb{C}_1\ \{Q_1\} \qquad \vdash \{P_2\}\ \mathbb{C}_2\ \{Q_2\}}{\vdash \{P_1 * P_2\}\ \mathbb{C}_1 \parallel \mathbb{C}_2\ \{Q_1 * Q_2\}}$$

location "value" permission

$$\{\texttt{Counter}\ (\mathtt{x}, 0, 1)\}$$
$$\{\texttt{Counter}\ (\mathtt{x}, 0, 0.6) * \texttt{Counter}\ (\mathtt{x}, 0, 0.4)\}$$
$$\begin{array}{c|c} \{\texttt{Counter}\ (\mathtt{x}, 0, 0.6)\} & \{\texttt{Counter}\ (\mathtt{x}, 0, 0.4)\} \\ \texttt{inc(x)} & \texttt{inc(x)} \\ \{\texttt{Counter}\ (\mathtt{x}, 1, 0.6)\} & \{\texttt{Counter}\ (\mathtt{x}, 1, 0.4)\} \end{array}$$
$$\{\texttt{Counter}\ (\mathtt{x}, 1, 0.6) * \texttt{Counter}\ (\mathtt{x}, 1, 0.4)\}$$
$$\{\texttt{Counter}\ (\mathtt{x}, 2, 1)\}$$

$$\texttt{Counter}\ (x, v_1, \pi_1) * \texttt{Counter}\ (x, v_2, \pi_2) \iff \texttt{Counter}\ (x, v_1 + v_2, \pi_1 + \pi_2)$$

Time and Data Abstraction (TaDA)

- TaDA introduces **abstract atomicity** based on a correctness condition known as **linearizability**.
- Abstract atomicity allows for more precise specifications that are not weakened to account for interference from the environment.

Concurrent Local Subjective Logic (CoLoSL)

- CoLoSL introduces a way to **re-organise the boundaries** of modules and their interferences ("contract") at the logic level.

Applications

- **Specifications for Concurrent Maps/Indexes.** We give specifications that allow reasoning about client programs such as producer-consumer and parallel sieve of Eratosthenes and verifying the implementations such as B-tree and skiplist.
- **Concurrent Specifications for file systems.** Specifications of POSIX are written in English and sometime can be ambiguous, particularly in the concurrent case. TaDA-Refine, an extension of TaDA, has been used to formally specify the core parts of the POSIX file system operations.

On-going work: Total-TaDA

- Extends TaDA to verifying the termination of a subtle subset of concurrent, heap manipulating programs known as "wait-free" programs.
- Attempts to maintain the separation of client and module verification, possible with abstract TaDA specifications.

On-going work: Snapshot

- Transactions running under snapshot isolation, a weak consistency model, take a copy of the entire database, update locally and can commit if no write conflict. This leads to some unintuitive behaviours.
- We extend RG to capture the unintuitive behaviours.

This research is supported by the EPSRC Programme Grant **REMS**: Rigorous Engineering for Mainstream Systems and the **Dept. of Computing**, Imperial College London.

Reference:

1. *Steps in Modular Specifications for Concurrent Modules (Invited Tutorial Paper), Pedro da Rocha Pinto, Thomas Dinsdale-Young and Philippa Gardner, Proceedings of the 31st MFPS '15, pp. 3–18*
2. *TaDA: A Logic for Time and Data Abstraction, Pedro da Rocha Pinto, Thomas Dinsdale-Young and Philippa Gardner, Proceedings of the 28th ECOOP '14, pp. 207–231*
3. *CoLoSL: Concurrent Local Subjective Logic, Azalea Raad, Jules Villard and Philippa Gardner, Proceedings of the 24th ESOP '15, pp. 710–735*
4. *Modular Termination Verification for Non-blocking Concurrency, Pedro da Rocha Pinto, Thomas Dinsdale-Young, Philippa Gardner and Julian Sutherland, Proceedings of the 25th ESOP '16, pp. 176–201*
5. *Abstract Specifications for Concurrent Maps, Shale Xiong, Pedro da Rocha Pinto, Gian Ntzik and Philippa Gardner, Proceedings of the 26th ESOP '17, pp. 964–990*
6. *Reasoning About POSIX File Systems, Gian Ntzik, Ph.D. Thesis, Imperial College London*