Proofs of concurrent programs

Matthew Parkinson
University of Cambridge
March 13th 2007

Joint work with Cristiano Calcagno and Viktor Vafeiadis
Motivation

• Concurrency is hard:
  “If you can get away with it, avoid using threads. Threads can be difficult to use, and they make programs harder to debug.”
  Java Sun Tutorial “Threads and Swing”

• Multi-core means concurrency everywhere!
Concurrent Libraries

- Library code - Modularity
- Dynamically allocated data structures
- Fine-grained concurrency

We want tractable verification of concurrent programs
Coarse-grain locking
Coarse-grain locking

2 → 3 → 5 → 7 → 11 → 13

Diagram of a lock with numbers 2 to 13 connected in sequence.
Coarse-grain locking

Inefficient as only one thread operates on the list at a time
Fine-grain locking
Fine-grain locking
Fine-grain locking
Fine-grain locking
Fine-grain locking
Fine-grain locking
Fine-grain locking
Fine-grain locking
Fine-grain locking
Fine-grain locking
Fine-grain locking
Fine-grain locking
Locking

Lock head

Lock next

Unlock
Adding and removing

Add node

Delete node
History

- Hoare [1971]
- Owicki-Gries [1975]
- Temporal Logic [1977]
- Rely-Guarantee [1981]
- Separation logic [1999, 2004]
History

- Hoare [1971]
- Owicki-Gries [1975]
- Temporal Logic [1977]
- Rely-Guarantee [1981]
- Separation logic [1999, 2004]

Can’t reason about the previous example!
Local reasoning

Unlock
Local reasoning

Unlock
Local reasoning

Unlock
Local reasoning

Unlock

2 -> 3 -> 5 -> 7 -> 11 -> 13

Unlock
Separation logic

• Provides local reasoning with
  • separating conjunction; and
  • frame rule.

\[
\begin{align*}
\{P\}C\{Q\} \\
\{P \ast R\}C\{Q \ast R\}
\end{align*}
\]
Frame rule

\[ \{P\} C \{Q\} \]

\[ \{P \ast R\} C \{Q \ast R\} \]

Unlock

\[ P \rightarrow Q \]
Frame rule

\[
\frac{\{P\}C\{Q\}}{\{P \ast R\}C\{Q \ast R\}}
\]

Unlock

\[
\{ P \} C \{ Q \} \\
\{ P \ast R \} C \{ Q \ast R \}
\]
Frame rule

\[
\begin{align*}
\{P\}C\{Q\} \\
\{P \ast R\}C\{Q \ast R\}
\end{align*}
\]

Unlock
Frame rule

\[
\frac{\{P\}C\{Q\}}{\{P \ast R\}C\{Q \ast R\}}
\]

Unlock
Interference:
other threads
Interference: other threads
Interference:
other threads
Stability

Lock head

Lock next

Unlock
Stability

Add node

Delete node
Ownership Transfer

Add node

Where did this node come from?

Delete node

Where did this node go to?
Local and shared state
Local and shared state
Local and shared state

Global

2 → 3 → 5

7 → 11 → 13

Local

6
Local and shared state
Local and shared state

Global

Local
Parallel rule

Splits local state, shares global state.

\[
\text{Act} \vdash \{p_1\} C_1 \{q_1\} \\
\text{Act} \vdash \{p_2\} C_2 \{q_2\} \\
\frac{}{\text{Act} \vdash \{p_1 \ast p_2\} C_1 \parallel C_2 \{q_1 \ast q_2\}}
\]
Atomic rule

\[
p_2 \rightarrow q_2 \in \text{Act}
\]

\[
q_2 \text{ stable under } \text{Act}
\]

\[
\text{Act} \vdash \{p_1 \ast p_2\} C \{q_1 \ast q_2\}
\]

\[
\text{Act} \vdash \{p_1 \ast [p_2]\} \text{atomic } C \{q_1 \ast [q_2]\}
\]
Summary

• Distinguish between global and local state.
• Ownership transfer.
• Specific actions that can be performed on shared state.
• Stability
Tool Support

- SmallfootRG
- Symbolic execution
- Stability inference - Shape inference
- Automatically verifies this example in 4.27 seconds.
Conclusion

• Rely-guarantee - actions

• Separation logic - ownership transfer, framing, disposal of memory.

• Verification of fine-grain, and non-blocking, concurrent algorithms.

• Logic is tractable enough that tools can be built.