Starling: automating concurrency verification

Mike Dodds[1], **Matthew Parkinson**[2], Matt Windsor[1]

(1) University of York (2) Microsoft Research

Funding from Royal Society Industrial Fellowship for Dodds.

Work In Progress

Views: Compositional Reasoning for Concurrent Programs

Thomas Dinsdale-Young
Imperial College
td202@doc.ic.ac.uk

Lars Birkedal
IT University of Copenhagen
birkedal@itu.dk

Philippa Gardner
Imperial College
pg@doc.ic.ac.uk

Matthew Parkinson
Microsoft Research
mattpark@microsoft.com

Hongseok Yang
University of Oxford
hongseok00@gmail.com

Abstract

Compositional abstractions underly many reasoning principles for concurrent programs: the concurrent environment is abstracted in order to reason about a thread in isolation; and these abstractions are composed to reason about a program consisting of many threads. For instance, separation logic uses formulae that describe part of the state, abstracting the rest; when two threads use disjoint state, their specifications can be composed with the separating conjunction. Type systems abstract the state to the types of variables; threads may be composed when they agree on the types of shared

In a concurrent setting, compositional reasoning allows a thread to be considered in isolation, rather than considering all possible interleavings of a program.

Type systems and program logics are two common forms of compositional reasoning. They strike a balance between invariant properties that must be preserved during the execution of a concurrent program, and operations that may be performed. Standard type systems and rely-guarantee methods [20] focus on preserving global properties; for example, the typing of the memory. Such approaches are good at handling sharing and interference, but work less well with stateful behaviour. Contrastingly, separa-

Ticketed Lock

```
global int ticket;  // The next ticket to hand out.
global int serving; // The current ticket holding the lock.

method lock() {
  <t = ticket++>;
  do {
    <s = serving>;
  } while (s != t);
}
```

```
method unlock() {
  <serving++>;
}
```

Ticketed Lock

```
method lock() {
  {| emp |}
  <t = ticket++>;
  do {
    <s = serving>;
  } while (s != t);
  {| holdLock() |}
}
```

```
method unlock() {
  {| holdLock() |}
  <serving++>;
  {| emp |}
}
```

```
constraint holdLock()   * holdLock()   -> false;
```

Ticketed Lock

```
method lock() {
  {| emp |}
  <t = ticket++>;
  {| holdTick(t) |}
  do {
    {| holdTick(t) |}
    <s = serving>;
    {| if s==t then holdLock() else holdTick(t) |}
  } while (s != t);
  {| holdLock() |}
}
```

```
method unlock() {
  {| holdLock() |}
  <serving++>;
  {| emp |}
}
```

```
constraint holdLock()  * holdLock()  -> false;
```

Demo

Views

$$\forall(\{p\}a\{q\}) \in \mathsf{Axioms}.\, \forall c \in \mathsf{Views}.\, [\![a]\!]\lfloor p * c \rfloor \subseteq \lfloor q * c \rfloor$$

Checking proof outline

$$\forall(\{p\}a\{q\}) \in \mathsf{Axioms}.\, \forall c \in \mathsf{Views}.\, [\![a]\!]\lfloor p * c \rfloor \subseteq \lfloor q * c \rfloor$$

Reification

$$\mathcal{D}_{\uparrow}(p) = \begin{cases} \mathcal{S} & \text{if } p \notin dom(\mathcal{D}) \\ \mathcal{D}(p) & \text{if } p \in dom(\mathcal{D}) \end{cases}$$

$$\mathcal{D} : \text{Views} \rightharpoonup \mathcal{P}(\mathcal{S})$$

$$\texttt{constraint } \overbrace{\texttt{holdLock() * holdLock()}}^{r} \texttt{ -> } \overbrace{\texttt{false;}}^{\mathcal{D}(r)}$$

$$\lfloor q \rfloor = \bigcap_{p \subseteq_m q} \mathcal{D}_{\uparrow}(p)$$

Check proof outlines

$$\forall(\{p\}a\{q\}) \in \mathsf{Axioms}.\, \forall r \in dom(\mathcal{D}).\, [\![a]\!] \lfloor p * (r \setminus_m q) \rfloor \subseteq \mathcal{D}(r)$$

Proof

$$
\begin{aligned}
&\forall r \in dom(\mathcal{D}).\, [\![a]\!]\lfloor p * (r \setminus_m q)\rfloor s \Rightarrow \mathcal{D}(r)(s) \\
\Longrightarrow\ & \forall c.\, \forall r \in dom(\mathcal{D}).\, r \setminus_m q \subseteq_m c \Rightarrow [\![a]\!]\lfloor p * c\rfloor s \Rightarrow \mathcal{D}(r)(s) \\
\Longrightarrow\ & \forall c.\, \forall r \in dom(\mathcal{D}).\, r \subseteq_m q * c \Rightarrow [\![a]\!]\lfloor p * c\rfloor s \Rightarrow \mathcal{D}(r)(s) \\
\Longrightarrow\ & \forall c.\, [\![a]\!]\lfloor p * c\rfloor s \Rightarrow \forall r \in dom(\mathcal{D}).\, r \subseteq_m q * c \Rightarrow \mathcal{D}(r)(s) \\
\Longrightarrow\ & \forall c.\, [\![a]\!]\lfloor p * c\rfloor s \Rightarrow \lfloor q * c\rfloor s
\end{aligned}
$$

Open Development

Follow the project on GitHub:

http://github.com/septract/starling-tool/