Starling: automating concurrency verification

Mike Dodds⁽¹⁾, **Matthew Parkinson**⁽²⁾, Matt Windsor⁽¹⁾

University of York

(2) Microsoft Research

Work In Progress Funding from Royal Society Industrial Fellowship for Dodds.

Views: Compositional Reasoning for Concurrent Programs

Thomas Dinsdale-Young Imperial College

td202@doc.ic.ac.uk

Matthew Parkinson Microsoft Research

mattpark@microsoft.com

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

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

Type systems and program logics are two common forms of invari
Type systems and program logics are two common forms of Type systems and program logics are two common forms of invariant compositional reasoning. They strike a balance the execution of a must be preserved during the execution of a properties that must be preserved during the execution of a must b compositional reasoning. They strike a balance between invari-tion of a during the execution of a compositional reasoning. Stan-ant properties that must be preserved that may be performed. Stan-concurrent program, and operations that may be performed. ant properties that must be preserved during the execution of a stan-concurrent program, and operations that may be performed. Stan-concurrent program, and rely-guarantee methods [20] focus on meconcurrent program, and operations that may be performed. Stan-encurrent program, and rely-guarantee methods [20] focus on pre-dard type systems and rely-guarantee, the typing of the memory-dard type global properties; for example, the typing of the memorydard type systems and rely guarantee methods [20] focus on pre-serving global properties: for example, the typing and interference serving global properties: at handling sharing and interference such approaches are good at handling sharing and interference. serving global properties: for example, the typing of the memory, senaral serving global properties: for example, the typing and interference, senaral serving and interference, senaral serving global properties: for example, the typing of the memory, senaral serving global properties: for example, the typing of the memory, senaral serving global properties: for example, the typing of the memory, senaral serving global properties: for example, the typing of the memory, senaral serving global properties: for example, the typing of the memory, senaral serving global properties: for example, the typing of the memory, senaral serving global properties: for example, the typing of the memory, senaral serving global properties: for example, the typing and interference, senaral serving global properties: for example, the typing of the memory. Such approaches are good at handling sharing and interference, senara-but work less well with stateful behaviour Contracting to

Ticketed Lock

```
global int ticket; // The next ticket to hand out.
global int serving; // The current ticket holding the lock.
                                   method unlock() {
method lock() {
                                     <serving++>;
  <t = ticket++>;
 do {
   <s = serving>;
  } while (s != t);
```

Ticketed Lock

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

Ticketed Lock

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

Demo

Views

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

Checking proof outline

$$\forall (\{p\}a\{q\}) \in \mathsf{Axioms}. \ \forall c \in \mathsf{Views}. \ \llbracket a \rrbracket \lfloor p*c \rfloor \subseteq \lfloor q*c \rfloor$$

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

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

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

Check proof outlines

$$\forall (\{p\}a\{q\}) \in \mathsf{Axioms}. \ \forall r \in dom(\mathcal{D}). \ \llbracket a \rrbracket \lfloor p * (r \setminus_m q) \rfloor \subseteq \mathcal{D}(r)$$

Proof

```
\forall r \in dom(\mathcal{D}). \ [a] \ [p*(r \setminus_m q)] s \Rightarrow \mathcal{D}(r)(s)
\implies \forall c. \ \forall r \in dom(\mathcal{D}). \ r \setminus_m q \subseteq_m c \Rightarrow [a] \ [p*c] s \Rightarrow \mathcal{D}(r)(s)
\implies \forall c. \ \forall r \in dom(\mathcal{D}). \ r \subseteq_m q*c \Rightarrow [a] \ [p*c] s \Rightarrow \mathcal{D}(r)(s)
\implies \forall c. \ [a] \ [p*c] s \Rightarrow \forall r \in dom(\mathcal{D}). \ r \subseteq_m q*c \Rightarrow \mathcal{D}(r)(s)
\implies \forall c. \ [a] \ [p*c] s \Rightarrow [q*c] s
```

Open Development

Follow the project on GitHub:

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