Thread-Modular Reasoning for Heap-Manipulating Programs: Exploiting Pointer Race Freedom

— Sebastian Wolff —
Treiber’s stack

ToS

...
Treiber’s stack

push: 1. allocate new node

diagram: node

ToS

...
Treiber’s stack

push: 2. read top of stack

node

ToS

top

...
Treiber’s stack

push: 3. connect new node

node  ToS  top  ...

Treiber’s stack

push: 4. move top of stack if consistent ($\text{CAS}$), otherwise go back to step 2
Treiber’s stack

push: 4. move top of stack if consistent (CAS), otherwise go back to step 2
Treiber’s stack

pop: 1. read top of stack

ToS

top
Treiber’s stack

pop: 2. read the second topmost node
Treiber’s stack

pop: 3. move top of stack if consistent \((\text{CAS})\), go back to step 1 otherwise

![Diagram of Treiber's stack](image)
Treiber's stack

pop: 3. move top of stack if consistent (CAS),
go back to step 1 otherwise
Treiber’s stack

pop: 4. read out data, then free
There is a bug

thread 1: is in step 3 of pop, but interrupted
There is a bug

thread 2: pops
There is a bug

thread 2: pops, pushes
There is a bug

thread 2: pops, pushes, and pushes again
There is a bug

thread 1: continues and moves the top of stack
Analysis Requirements

• prove correctness (linearisability)
• unbounded number of threads (library)
• unbounded heap
• low-level memory operations (C-like)
• scalability
Thread-Modular Reasoning

- abstract domain: set of views
- single thread
- heap reachable by that thread
- relation among threads lost
- sequential + interference steps
Example

T1.pc = pop: CAS
T2.pc = push: CAS
T3.pc = push: CAS
Example

T1.pc = pop:CAS
T2.pc = push:CAS
T3.pc = push:CAS

cpc = pop:CAS

diagram:

T2:node
T3:node
ToS
T1:top
T2:top
T3:top
T1:next

top
next
Example

T1.pc = pop:CAS
T2.pc = push:CAS
T3.pc = push:CAS
Example

T1.pc = pop: CAS
T2.pc = push: CAS
T3.pc = push: CAS
Interference

pc = push:CAS

node → top → ... → node2 → top2 → ...

pc = push:CAS = pc2
Interference

victim

interferer

pc = push:CAS

node

top

ToS

pc = push:CAS

node

top

node2

top2

pc = push:CAS = pc2

ToS

...
Interference

- Victim:
  - Node
  - Top
  - \( pc = \text{push: CAS} \)

- Interferer:
  - Node
  - Top
  - \( pc = \text{push: CAS} \)

- Push to ToS

Diagram:
- Node to Top
- Victim to Interferer
- Push: CAS
Sequential Steps

pc = push:CAS
Sequential Steps

ToS
node → top

pc = push:CAS

ToS
node
top

pc = push:next
Sequential Steps

ToS
node → top

pc = push:CAS

ToS
node → top

pc = push:CAS
False Positives

• Problem: relation among local heaps is lost (due to thread modularity)

• Solution for GC: ownership

• Solution for MM: ?? (key question)
False Positives

• Problem: relation among *local* heaps is lost (due to thread modularity)

• Solution for GC: *ownership*

• Solution for MM: *ownership!*

  (key question)
Ownership for MM

- weak ownership
- granted upon allocation
- removed upon publishing
- dangling readers allowed

⇒ write privilege
Ownership for MM

dangling pointer

owned

...
Pointer Races Freedom

• establishes weak ownership for MM

• Pointer Race
  • writing through dangling pointers (invalid pointers)
  • following dangling pointers (strongly invalid pointers)
How to use PRF

• add validity and ownership information to views
• improve interference precision

➤ dangling pointers:
  • are allowed
  • but invalid
Example

\[
\text{pc} = \text{push:CAS} \\
\text{inv} = \emptyset, \text{sin} = \emptyset
\]
Example

\[
\begin{align*}
\text{top} & \quad \text{top}2 \\
\text{node} & \quad \text{node2} \\
\text{pc} = \text{push:CAS} & \quad \text{pc} = \text{push:CAS} \\
\text{inv} = \emptyset, \text{sin} = \emptyset & \quad \text{inv} = \emptyset, \text{sin} = \emptyset
\end{align*}
\]
Example

\[
\text{node} \rightarrow \text{top} \rightarrow \ldots
\]
\[
\text{pc} = \text{push:CAS} \quad \text{inv} = \emptyset, \sin = \emptyset
\]

\[
\text{node} \rightarrow \text{top} \rightarrow \ldots
\]
\[
\text{pc} = \text{push:CAS} \quad \text{inv} = \emptyset, \sin = \emptyset
\]

\[
\text{node2} \rightarrow \text{node} \rightarrow \text{top} \rightarrow \ldots
\]
\[
\text{inv} = \emptyset, \sin = \emptyset \quad \text{pc} = \text{push:CAS} = \text{pc2}
\]
ToS

inv = ∅, sin = ∅

pc = push:CAS

ToS

inv = ∅, sin = ∅

pc = push:CAS

ToS

inv = ∅, sin = ∅

pc = push:CAS

inv = ∅, sin = ∅
Prototype

- specification via finite observers (data independence)
- thread-modular framework
- shape analysis (heap abstraction)
- supports GC, MM, PRF
- checks linearisability
- ~5000 lines of code (C++)
### Evaluation: Treiber’s stack

<table>
<thead>
<tr>
<th></th>
<th>MM</th>
<th>PRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>runtime in seconds:</td>
<td>612</td>
<td>2.37</td>
</tr>
<tr>
<td>state space:</td>
<td>116776</td>
<td>744</td>
</tr>
<tr>
<td>sequential steps:</td>
<td>322328</td>
<td>2656</td>
</tr>
<tr>
<td>interference steps:</td>
<td>7913705</td>
<td>45815</td>
</tr>
</tbody>
</table>