Parameterised Linearisability

Andrea Cerone

Joint work with Alexey Gotsman and Hongseok Yang

York Concurrency Workshop - April 28, 2014
A Simple Example

- Converting a sequential data structure into a concurrent one

**Trivial Solution:**

```c
LOCK lock;
do_push(z):
    lock.acquire();
    int retval = push(z);
    lock.release();
return retval;
```

- Works for any implementation of `push`
- but it’s inefficient (we can do much better…)
**Flat Combiners (Hendler et al, 2010)**

**Idea:** let a single thread handle all requests

**Stack** provides methods `push`, `pop` to clients

**FC\_stack** : access to methods in **Stack** regulated by `do_{push}`, `do_{pop}`

```plaintext
0
1
... 

\( t_1 \) push(5)

\( t_2 \) push(7)

\( t_3 \)
```

```plaintext
\( t_1 \) do_{push}(5)
\( t_2 \) do_{push}(7)
```
Flat Combiners (Hendler et al, 2010)

**Idea:** let a single thread handle all requests

Stack provides methods \texttt{push}, \texttt{pop} to clients

\texttt{FC}_{\text{stack}}: access to methods in Stack regulated by \texttt{do}_{\text{push}}, \texttt{do}_{\text{pop}}

---

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{t}_1 \texttt{push}(5)</td>
<td>\texttt{t}<em>1 \texttt{do}</em>{\texttt{push}}(5)</td>
<td></td>
</tr>
<tr>
<td>\texttt{t}_2 \texttt{push}(7)</td>
<td>\texttt{t}<em>2 \texttt{do}</em>{\texttt{push}}(7)</td>
<td></td>
</tr>
</tbody>
</table>

It appears to behave as the naive solution

- \texttt{push}, \texttt{pop} not implemented (correct methods)
- Guaranteed to have correct behaviour for (almost) any implementation of abstract methods
Public methods define the operations made available to clients.

Abstract methods (without implementation) can be declared.

$$L : M' \rightarrow M$$

Methods with no code associated

Code for methods declared here is defined.
The implementation of abstract methods can be specified by another library.

\[ L_2 : M \rightarrow M' \]

\[ L_1 : M' \rightarrow M'' \]
The implementation of abstract methods can be specified by another library

\[ L_1 \circ L_2 : M \rightarrow M'' \]
Observational Refinement

**Program:**  
\[ \text{let } L \text{ in } C_1 \parallel \cdots \parallel C_n \]

only call public methods implemented in \( L \)

\( L \) has no abstract methods

**Observational Refinement:**

inclusion of client traces, for every possible client and every possible library parameter

\[ L_1 \sqsubseteq_{\text{obs}} L_2 : \forall \text{Cl.} \forall L : \emptyset \rightarrow M. \text{Obs}(\text{let } (L_1 \circ L) \text{ in Cl}) \subseteq \text{Obs}(\text{let } (L_2 \circ L) \text{ in Cl}) \]

Quantification over clients and library parameters  
A proof technique is needed
Specifying the behaviour of libraries

**Types for Libraries**

$L : M' \rightarrow M$

**Actions:**

Observable behaviour of Libraries

(Different entities execute in different memory spaces)

Interactions with the client

(t, ?call $m(z)$) \quad (t, !return $m(z)$) \quad $m \in M \setminus M'$

Interactions with the library parameter

(t, !call $m'(z)$) \quad (t, ?return $m'(z)$) \quad $m \in M'$

Abstract methods invoked by the client (not observable)

$m \in M \cap M'$
Specifying the behaviour of libraries

**Types for Libraries**

\[ L : M' \rightarrow M \]

**Actions:**

Observable behaviour of Libraries

(Different entities execute in different memory spaces)

**Histories:**

\[(t, \text{?call } m_1(z)) \quad (t, \text{!call } m'(z)) \quad (t, \text{?return } m'(z)) \quad (t', \text{!return } m_1(z'_1))\]

\[(t_2, \text{?call } m_2(z_2)) \quad (t_2, \text{!return } m_2(z'_2))\]

---

Implemented Code

Running

Execution of Abstract Method
\( h_1 \subseteq h_2 : h_2 \) preserves thread-local subhistories of \( h_1 \)

+ the following order of pairs of actions:
Encapsulated Linearisability

\[ L : M' \rightarrow M \quad M' \cap M = \emptyset \]

\( h_1 \subseteq_e h_2 \) preserves thread-local subhistories of \( h_1 \) + the following order of pairs of actions:

1. \((t_1, \text{!return } m_1(z_1))\)
2. \((t_2, \text{?return } m_2(z_2))\)

\[
\begin{align*}
\text{Implemented Code} & \\
\text{Running} & \\
\text{Execution of Abstract Method} & \\
\end{align*}
\]
Applying Linearisability to Libraries

\[ \llbracket L \rrbracket : \text{histories generated by a library} \]
(\text{arbitrary behaviour of client and library parameter})

- Relating libraries:

\[ L_1, L_2 : M \rightarrow M' \]

\[ L_1 \sqsubseteq L_2 : \forall \ h_1 \in \llbracket L_1 \rrbracket . \exists \ h_2 \in \llbracket L_2 \rrbracket . h_1 \sqsubseteq h_2 \]

- if \( M \cap M' = \emptyset \)

\[ L_1 \sqsubseteq e L_2 : \forall \ h_1 \in \llbracket L_1 \rrbracket . \exists \ h_2 \in \llbracket L_2 \rrbracket . h_1 \sqsubseteq_e h_2 \]
Contextuality of Linearisability

**Theorem:** \( L_1, L_2 : M' \rightarrow M'' \quad L_{\text{in}} : M \rightarrow M' \)

\[ L_1 \sqsubseteq L_2 \iff (L_1 \circ L_{\text{in}}) \sqsubseteq (L_2 \circ L_{\text{in}}) \]

**Corollary:** \( L_1 \sqsubseteq L_2 \iff L_1 \sqsubseteq_{\text{obs}} L_2 \)
Linearisability does not suffice for Flat Combiners

\[ FC : \{m_i\}_{i \in I} \rightarrow \{\text{do}_{m_i}\}_{i \in I} \]

a single thread handles all concurrent requests

\[ FC \# : \{m_i\}_{i \in I} \rightarrow \{\text{do}_{m_i}\}_{i \in I} \]
calls to abstract methods regulated by a global lock

**Instantiating the library parameter**

\[
\text{int } m_i() \{ \text{return getTid(); } \}
\]

**Instantiating the Client**

\[
t_1 : \text{do}_{m_i}(); \parallel t_2 : \text{do}_{m_i}();
\]

\[
FC \not\sqsubseteq_{\text{Obs}} FC \#
\]

\[
\rightarrow
\]

\[
FC \not\sqsubseteq FC \#
\]

using \( FC \# : \)

\[
t_1 : \text{do}_{m_i}(); \parallel t_2 : \text{do}_{m_i}();
\]

always returns \( t_1 \) always returns \( t_2 \)

using \( FC : \)

\[
t_1 : \text{do}_{m_i}(); \parallel t_2 : \text{do}_{m_i}();
\]

can return \( t_2 \) can return \( t_1 \)
A Weaker Linearisability Definition

$FC \#_{\#}$ linearises $FC$

if the library parameter does not use thread local information
Up-to Linearisability and Observational Refinement

\[ M' \rightarrow M, M' \cap M = \emptyset \]

\( \mathcal{R} : \) binary relation between sequences of calls and returns to methods in \( M' \)

\[ h_1 \sqsubseteq_{\mathcal{R}} h_2 : \bullet h_1|_{\text{ClAct}} \sqsubseteq h_2|_{\text{ClAct}} \quad \bullet h_1|_{\text{AbsAct}} \mathcal{R} h_2|_{\text{AbsAct}} \]

(a variant of contextuality holds for \( \sqsubseteq_{\mathcal{R}} \))

\textbf{Soundness:} \[ L_1 \sqsubseteq_{\mathcal{R}} L_2 \implies L_1 \sqsubseteq_{\text{obs}} L_2 \]

only \( \mathcal{R} \)-closed library parameters are considered

\[ \text{FC} \sqsubseteq_{\mathcal{R}_t} \text{FC}^\# \implies \text{FC} \sqsubseteq_{\text{obs}} \mathcal{R}_t \text{FC}^\# \]

\( \bullet \) Guaranteed to have correct behaviour for \textit{(almost)} any implementation of abstract methods
Conclusions

Future Research:

- Other applications (Joins, Iterators, …)
- STM and Transactional Boosting
- Linearisability for Higher Order Objects
- Connections with other forms of HO-reasoning (e.g. CaReSL)

Thank you!
Contextuality of Up-to Linearisability

\[ L_{\text{in}} : M \to M' \]

- \( \mathcal{R} \) relates sequences of actions belonging to \( M' \setminus M \)
- \( \mathcal{G} \) relates sequences of actions belonging to \( M \)

(\( \mathcal{R}, \mathcal{G} \))-closure: defines how closure properties of library parameters have to be changed when an inner library is instantiated.

\( L_{\text{in}} \): rearranges this part of a history according to \( \mathcal{R} \)

\( \mathcal{G} \): causes this part of the history to be rearranged according to \( \mathcal{G} \)
Contextuality of Up-to Linearisability

\[ L_1, L_2 : M' \to M'' \]
\[ L_1 \sqsubseteq^R L_2 \]
\[ L_{\text{in}} : M \to M' \]
\[ L_{\text{in}} \text{ is } (\mathcal{R})\text{-closed} \]
Contextuality of Up-to Linearisability

\[ L_1, L_2 : M' \to M'' \]
\[ L_1 \sqsubseteq \mathcal{R} L_2 \]
\[ L_{\text{in}} : M \to M' \]
\[ L_{\text{in}} \text{ is } (\mathcal{R})\text{-closed} \]

\[ M' \cap M'' = \emptyset \land \]
\[ M \cap M' = \emptyset \implies \]
\[ (L_1 \circ L_{\text{in}}) \sqsubseteq_G (L_2 \circ L_{\text{in}}) \]