CoLoSL
Concurrent Local Subjective Logic

Azalea Raad  Jules Villard  Philippa Gardner

Imperial College London

24 April 2015
Global Shared Resources

\[ P_1 \land P_2 \]

\[
\begin{array}{c}
\{P_1\} \quad C_1 \quad \{Q_1\} \\
\{P_2\} \quad C_2 \quad \{Q_2\} \\
\{P_1 \land P_2\} \quad C_1 \parallel C_2 \quad \{Q_1 \land Q_2\}
\end{array}
\]

Owicki-Gries, Rely-Guarantee
Global Shared Resources

\[
\begin{align*}
\{P_1\} & \quad C_1 & \quad \{Q_1\} & \quad \{P_2\} & \quad C_2 & \quad \{Q_2\} \\
\{P_1 \land P_2\} & \quad C_1 & \quad C_2 & \quad \{Q_1 \land Q_2\}
\end{align*}
\]

- **No framing on shared resources / interference**
  - Reasoning on GLOBAL resources
  - Interference on ALL resources considered

- **No extension**
  - cannot dynamically share resources/extend interference
Disjoint Shared Resources

Local

\[ R \]

\[ * \]

Shared

\[ r_1 : I_1 \]
\[ r_2 : I_2 \]
\[ ... \]

\[ P_1 \]
\[ P_2 \]

CSL, FCSL, CAP, HOCAP, iCAP, TaDA
Disjoint Shared Resources

\[
\begin{align*}
\{P_1_{I_1}\} & \quad C_1 \quad \{Q_1_{I_1}\} \\
\{P_2_{I_2}\} & \quad C_1 \quad \{Q_2_{I_2}\} \\
\{P_1_{I_1} \ast \ P_2_{I_2}\} & \quad C_1 \parallel C_2 \quad \{Q_1_{I_1} \ast \ Q_2_{I_2}\}
\end{align*}
\]

- **Limited framing on shared resources / interference**
  - Static (pre-determined) frames (regions/ invariants)
  - Physically disjoint frames

- **Limited extension**
  - Can create new regions / invariants
  - **Cannot** extend regions with more resources/invariants
CoLoSL: Concurrent Local Subjective Logic
**CoLoSL: Concurrent Local Subjective Logic**

\[
\begin{align*}
\{P_1\}_{I_1} & \quad C_1 \quad \{Q_1\}_{I_1} \\
\{P_2\}_{I_2} & \quad C_1 \quad \{Q_2\}_{I_2} \\
\{P_1, P_2\}_{I_1 \ast I_2} & \quad C_1 \parallel C_2 \quad \{Q_1, Q_2\}_{I_1 \ast I_2}
\end{align*}
\]

- **Flexible framing on shared resources/invariants**
  - Overlapping frames
  - Dynamic framing/rewriting of interference
- **Dynamic extension**
  - Extension of shared state with new resources/interference
Examples

- **B+ Tree** (thanks to Shale Xiong)
  - Module composition; proof modularity; better abstraction
  - B+ Tree = List + BSTree
Ordered Singly Linked-List

List([[(k_1, v_1), (k_2, v_2), \ldots, (k_n, v_n)]] =

- Value update
- Map (e.g. increment all elements)
- Insertion (involves pointer surgery)
- Removal (involves pointer surgery)
Concurrent Ordered Singly Linked-List

\[
\text{List}([ (k_1, v_1), (k_2, v_2), \ldots, (k_n, v_n) ]) =
\]

\[\mathbb{I}_L = \mathbb{I}_{\text{up}} \cup \mathbb{I}_{\text{map}} \cup \mathbb{I}_{\text{add}} \cup \mathbb{I}_{\text{rem}}\]

- **Value update**
- **Map** (e.g. increment all elements)
- **Insertion** (involves pointer surgery)
- **Removal** (involves pointer surgery)
Balanced Search Tree (Degree 2)

- **Balanced**: all immediate subtrees of a node have the same height
- **Leaf-heavy**: data (values) stored in leaf nodes
- **Degree (d)**: no. of children (m) on each node \( d \leq m \leq 2d \)
Balanced Search Tree (Degree 2)

- **Search; Value update**
- **Insertion** may require **splitting**
- **Removal**

Insertion may require splitting.
Balanced Search Tree

- Insertion may require splitting

Search; Value update

- Insertion may require splitting
- Removal may require merging

at min capacity
Balanced Search Tree

- Search; Value update
- Insertion
- Removal may require merging

```
remove(k7)
```
Balanced Search Tree

BSTree 2 \(((k_1, d_1) \ldots (k_8, d_8))\) =

\[
\begin{align*}
&\begin{array}{c}
&k_2 \quad k_6
\\&d_1 \quad d_2
\\&d_3 \quad d_4
\\&d_5 \quad d_6
\\&d_7 \quad d_8
\\&k_1 \quad k_2
\\&k_3 \quad k_4
\\&k_5 \quad k_6
\\&k_7 \quad k_8
\\end{array}
\\&2
\end{align*}
\]
Balanced Search Tree

\[\text{BSTree} \ (h+1) \ L = \exists \ k_1, \ldots, k_m, L_1, \ldots, L_{(m+1)} \ L = L_1 \cup \ldots \cup L_{(m+1)} \text{ Node} \ (k_1, \ldots, k_m) \ (\text{BSTree} \ h \ L_1) \ldots \ (\text{BSTree} \ h \ L_{(m+1)})\]
Concurrent Balanced Search Tree

BSTree([(k1, d1) … (k8, d8)]) =

IB = Iup ∪ Ifind ∪ Iadd ∪ Irem
B+ Tree
B+ Tree

B+ Tree $h \ L \ \Longleftrightarrow \ \text{BSTree} \ h \ L \ \ast \ \text{List} \ L
Concurrent B+ Tree

\[ I = I_B \cup I_L \]
Concurrent B+ Tree Wish List

\[ B+ \text{ Tree } h \ L \ L_{B} \cup L_{L} \leftrightarrow \text{BSTree } h \ L \ L_{B} \uplus \text{List } L \ L_{L} \]

\[ I = L_{B} \cup L_{L} \]

\[ \text{IB} \cup \text{IL} \]

\[ \text{I} \]
Concurrent B+ Tree
(Existing Approaches)

\[ I \equiv I_B \cup I_L \]
Concurrent B+ Tree
(CoLoSL)

B+ Tree $h \mathcal{L} \mathcal{B}_B \cup \mathcal{L}_L$ $\iff$ BSTree $h \mathcal{L} \mathcal{B}_B \cup \mathcal{L}_L$

$I = I_B \cup I_L$
B+ Tree Wish List

- **Module Composition**
  
  $\text{B+ Tree h } L \bigcup I_B \bigcap I_L \Leftrightarrow \text{BTree h } L \bigstar \text{List L}$

- **Proof Modularity (overlapping frames)**
  
  - **Reuse** list-only operation (e.g. map)
    
    $\text{BTree h } L \bigstar \text{List L} \overset{\text{(frame)}}{\Rightarrow} \text{List L}$
  
  - **Reuse** tree-only operation (e.g. search)
    
    $\text{BTree h } L \bigstar \text{List L} \overset{\text{(frame)}}{\Rightarrow} \text{BTree h } L$
  
  - **Combine** tree/list module operations to implement B+ tree operations (e.g. remove)
CoLoSL Principles

\[
\begin{align*}
\text{B+ Tree } h \cdot L & \quad \text{I}_B \cup \text{I}_L \\
\text{BSTree } h \cdot L & \quad \text{CB} \ast \quad \text{List } L \\
\Downarrow & \\
\text{BSTree } h \cdot L \ast \text{List } L & \quad \text{I}_B \cup \text{I}_L
\end{align*}
\]
Duplicating Resources

\[ P \implies P \cup \ast \]
CoLoSL Principles

BSTree h L ⋃ List L

(BSTree h L ⋃ List L) ⋃ (BSTree h L ⋃ List L)

(BSTree h L ⋃ List L) ⋃ (BSTree h L ⋃ List L)
Forgetting Resources

\[ P \ast \cup Q \implies P \]

Diagram explanation:
- The diagram illustrates a relationship between sets or resources being forgotten.
- From the initial set or collection of resources represented by \( P \ast \cup Q \), it is forgotten or removed
- The resulting set is represented by \( P \)
CoLoSL Principles

\[ \text{BTree} \, h \, L \]

\[ \text{List} \, L \]

\( \bigcup \)

\[ \ast \]

\[ \text{BSTree} \, h \, L \]

\[ \text{List} \, L \]

\( \downarrow \)

\[ \text{BSTree} \, h \, L \]

\[ \ast \]

\[ \text{List} \, L \]

\( \downarrow \)

\[ \text{List} \, L \]

\( \downarrow \)

\( \text{Copy} \)

\( \text{Forget} \)

\[ \text{BSTree} \, h \, L \]

\[ \ast \]

\[ \text{List} \, L \]

\( \downarrow \)

\[ \text{List} \, L \]

\( \downarrow \)

\[ \text{List} \, L \]

\( \downarrow \)

\[ \text{BTree} \, h \, L \]

\[ \ast \]

\[ \text{List} \, L \]

\[ \text{List} \, L \]
Forgetting Interference (Shift)

if \( I \subseteq^P I' \)

then \[ P \] \[ I \] \[ \Rightarrow \] \[ P \] \[ I' \]
CoLoSL Principles

\[ \text{BTree } h \text{ L} \]

\[ \bullet \]

\[ \text{List } L \]

\[ \text{BSTree } h \text{ L} \]

\[ \bullet \]

\[ \text{List } L \]

\[ \text{BSTree } h \text{ L} \]

\[ \bullet \]

\[ \text{List } L \]

\[ \text{BSTree } h \text{ L} \]

\[ \bullet \]

\[ \text{List } L \]

\[ \text{BSTree } h \text{ L} \]

\[ \bullet \]

\[ \text{List } L \]

\[ \text{BSTree } h \text{ L} \]

\[ \bullet \]

\[ \text{List } L \]
CoLoSL Principles

\[ \text{B+ Tree } h \ L \quad I_B \cup I_L \quad \Rightarrow \quad \text{BSTree } h \ L \quad I_B \ast \quad \text{List } L \quad I_L \]

\[ \text{B+ Tree } h \ L \quad I_B \cup I_L \quad \Leftarrow \quad \text{BSTree } h \ L \quad I_B \ast \quad \text{List } L \quad I_L \]
Merging Resources

\[ P \cup_* Q \] \Rightarrow \[ P \cup_* Q \]

\[ I \quad I' \quad I \cup I' \]
CoLoSL Principles

\[ \text{BSTree} \, h \, L \, I_B \biguplus \, \text{List} \, L \, I_L \]

\[ \Downarrow \]

\[ \text{BSTree} \, h \, L \, \biguplus \, \text{List} \, L \, I_B \biguplus I_L \]

\[ \Downarrow \]

\[ \text{B+ Tree} \, h \, L \, I_B \biguplus I_L \]
Examples

- B+ Tree
  - Module composition; proof modularity; better abstraction
  - Proof reuse - local proofs done in the largest context possible
B+ Tree: Possible Extension

- Re-degree the tree
  - periodically change the degree (d) for better search time
- Re-degreeing ONLY affects the tree structure
- Re-degreeing does NOT affect the list structure
  - Should not have to reprove list operations
Re-degreeing extension only affects the tree

Re-degreeing extension does NOT affect the list proofs

- Can reuse the proofs as before
Examples

- **B+ Tree**
  - Module composition; proof modularity; better abstraction
  - Proof reuse

- **Concurrent List**
  - Dynamic extension

- **Spanning Tree**
  - Recursive overlapping graph predicate
  - Local proof; proof structures matches the algorithm

- **Dijkstra’s Self-stabilising Token Ring**
  - Local proof; proof Reuse
Conclusions

- From OG/Rg to CAP/TaDA
  - Huge steps towards compositionality/locality
  - Are we there yet? No!

- CoLoSL
  - Subjective/overlapping views
  - Flexible framing on shared resource/interference
  - Dynamic extension
  - More modular; better proof reuse
  - Are we there yet? Still No!
    - Abstraction layers; abstract atomicity, …

Thank you for listening!