RGSep

Viktor Vafeiadis
Coarse-grain locking
Fine-grain locking (pessimistic)

One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.
Fine-grain locking (pessimistic)

One lock per node:

— Traversals acquire locks in a “hand over hand” fashion.
— If node is locked, we can add a node after it.
— If two adjacent nodes are locked, we can delete the second.
Fine-grain locking (pessimistic)

One lock per node:

— Traversals acquire locks in a “hand over hand” fashion.
— If node is locked, we can add a node after it.
— If two adjacent nodes are locked, we can delete the second.
Fine-grain locking (pessimistic)

One lock per node:
- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.
Fine-grain locking (pessimistic)

One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.
Fine-grain locking (pessimistic)

One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.
Fine-grain locking (pessimistic)

One lock per node:

— Traversals acquire locks in a “hand over hand” fashion.
— If node is locked, we can add a node after it.
— If two adjacent nodes are locked, we can delete the second.
Fine-grain locking (pessimistic)

One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.
Fine-grain locking (optimistic)
Fine-grain locking (optimistic)
Fine-grain locking (optimistic)

Re-traverse the list OR perform deletions in two steps

Leaks memory: cannot dispose deleted nodes.
Actions (pessimistic algorithm)

Lock

Unlock

Add node

Delete node
Ownership transfer

Add node

Delete node
Local and shared state

Pessimistic algorithm

Shared

Local

Add node
Local and shared state

Pessimistic algorithm

Shared

Local

Add node
Local and shared state

Pessimistic algorithm

Shared

2 → 3 → 5 → 7 → 11 → 13

Local

Lock
Local and shared state

Pessimistic algorithm

Shared

Local

Lock
Local and shared state

Pessimistic algorithm

Shared

Local

Delete node
Local and shared state
Local and shared state

Pessimistic algorithm

Shared

2 3 5 7 11 13

Local

6

Now, the node is local; we can safely dispose it.
Local and shared state

Now, the node is local; we can safely dispose it.
Actions (optimistic algorithm)

- **Lock**
  - Add node
  - Unlock

- **Add node**

- **Delete node**
Local and shared state

Optimistic algorithm

Shared

Local

Delete node
Local and shared state

Optimistic algorithm

Shared

Local

Delete node
Interference: other threads
Interference: other threads
Interference: other threads
Interference: other threads
Stability

Lock

Unlock
Stability

Add node

Delete node
Assertions

P, Q, R, ...  — separation logic assertions
p, q, r, ...  — RGSep assertions

\[ p ::= P \mid p \cdot q \mid p \lor q \mid p \land q \mid \exists x. p \mid \forall x. p \]

\[ P(l, s) \overset{\text{def}}{=} P(l) \]
\[ \boxed{P}(l, s) \overset{\text{def}}{=} P(s) \]
\[ (p \cdot q)(l, s) \overset{\text{def}}{=} \exists l_1 l_2. \text{dom}(l_1) \cap \text{dom}(l_2) = \emptyset \land l = l_1 \cup l_2 \land p(l_1, s) \land q(l_2, s) \]
Actions

\[ x \mapsto 0, v, y \quad \Rightarrow \quad x \mapsto A, v, y \]
\[ x \mapsto A, v, y \quad \Rightarrow \quad x \mapsto 0, v, y \]
\[ x \mapsto A, v, y \quad \Rightarrow \quad x \mapsto A, v, z \]
\[ x \mapsto A, v, y \quad \Rightarrow \quad z \mapsto 0, w, y \]
\[ x \mapsto A, v, y \quad \Rightarrow \quad y \mapsto A, w, z \]
\[ x \mapsto A, v, y \quad \Rightarrow \quad y \mapsto A, w, z \]
Parallel composition

\[
\begin{align*}
C_1 & \text{ sat } (p_1, R \cup G_2, G_1, q_1) \\
C_2 & \text{ sat } (p_2, R \cup G_1, G_2, q_2) \\
(C_1 \parallel C_2) & \text{ sat } (p_1*p_2, R, G_1 \cup G_2, q_1*q_2)
\end{align*}
\]
Atomic commands

\[
\begin{align*}
\{ P \} \ C \ \{ Q \} & \quad \text{Local commands} \\
\text{C sat (} P, \ R, \ G, \ Q \text{)} & \\

(P_2, \ Q_2) \in \ G \\
\{ P_1 * P_2 \} \ C \ \{ Q_1 * Q_2 \} & \\
(\text{atomic C}) \ \text{sat (} P_1 * \boxed{P_2 * F}, \ \emptyset, \ G, \ Q_1 * \boxed{Q_2 * F} \text{)}
\end{align*}
\]

\[p, q \text{ stable under } R\]

(atomic C) \ \text{sat (} p, \ \emptyset, \ G, \ q \text{)}

(atomic C) \ \text{sat (} p, \ R, \ G, \ q \text{)}
Stability

— Local state assertions are trivially stable

— Shared state assertions:

\[ S \] is stable under \((P, Q)\)

if and only if

\[(P \land S) \land Q \implies S\]

\((P \land S)(h) \iff \exists h'. \text{ dom}(h) \cap \text{dom}(h') = \emptyset \land P(h') \land S(h \cup h')\)
Some further topics

Tool support:

– Symbolic execution with stabilization
– Action inference
– Linearization point inference (SmallfootRG & Cave)

Deny-guarantee & concurrent abstract predicates:

– Make interference specs first class
– Logical/abstract separation