Flow Interfaces
Compositional Abstractions of Concurrent Data Structures

Siddharth Krishna,
Dennis Shasha, and Thomas Wies
Verifying Data Structures

x → null
Separation Logic + Inductive Predicates

\[ ls(x) \overset{\text{def}}{=} x = \text{null} \land \text{emp} \lor \exists y. x \mapsto y \ast ls(y) \]

\[ ls(x) \Rightarrow \exists y. x \mapsto y \ast ls(y) \]

\[ \Rightarrow \exists y, z. x \mapsto y \ast y \mapsto z \ast ls(z) \]

\[ \Rightarrow \exists y, z, w. x \mapsto y \ast y \mapsto z \ast z \mapsto w \ast ls(w) \]

\[ \Rightarrow \exists y, z, w. x \mapsto y \ast y \mapsto z \ast z \mapsto w \ast w = \text{null} \land \text{emp} \]

\[ \Rightarrow \exists y, z, w. x \mapsto y \ast y \mapsto z \ast z \mapsto \text{null} \]
Harris' Non-blocking List
Harris' Non-blocking List
Limitations of Inductive Predicates

Traversals need to visit each node exactly once.

\[ ls(mh, null) \times ls(fh, null) \]
Limitations of Inductive Predicates

Traversals need to visit each node exactly once.

\[ \text{harris}(mh, fh, \text{null}) \overset{\text{def}}{=} \ldots \ast \text{harris}(\ldots) \]

Unbounded sharing
Other Approaches

Iterated separating conjunction: $\bigcirc_{x \in X} \phi(x)$

Can’t do better than closed set of nodes
Every node reachable – memory leaks
The Problem

An abstraction mechanism that can handle data structures like the Harris list?
(i.e. handle overlays, sharing, arbitrary traversals & have easy reasoning)
The Idea

- Inductive predicates:
  - Pro: inductive properties
  - Con: fixed traversals – no overlays/sharing
- Iterated $\ast$: $\bigotimes_{x \in X} \phi(x)$
  - Pro: easy reasoning
  - Con: only local properties
- Best of both?
- Inductive properties $\rightarrow$ local conditions
- But allow dependence on inductive quantity: flow
Can we express the property that root points to a tree as a local condition of each node in the graph?

Path counting!

∀n. pc(n) ≤ 1
Harris List

• Use two path-counting flows
  • One from mh and one from fh
  • Every node is on at least one of these lists

• Nodes labelled: marked/unmarked
  • All nodes in free list are marked
Separation Logic with Flows

- Graphs + Flows are a separation algebra
  - Use as semantic model
- Flow Interface: abstraction of flow graphs
  - To reason about modifications
- Logic: $\text{Gr}(I)$
  - A flow graph with interface $I$
- Expressivity:
  - Lists, trees, nested, sortedness, threaded, DAGs,...
Data-Structure-Agnostic Lemmas

• Decomposition
\[ \text{Gr}(I) \land x \in I \equiv \exists I_1, I_2. \text{N}(x, I_1) \ast \text{Gr}(I_2) \land I = I_1 \oplus I_2 \]

• Step
\[ I = I_1 \oplus I_2 \land (x, y) \in I_1^f \land I^f = \varepsilon \equiv y \in I_2 \]

• Composition
\[ \text{Gr}(I_1) \ast \text{Gr}(I_2) \equiv \text{Gr}(I_1 \oplus I_2) \]
Conclusion

• Flow Interfaces
  • can handle unbounded sharing and overlays
  • treat structural and data constraints uniformly
  • do not encode specific traversal strategies
  • provide data-structure-agnostic composition and decomposition rules
  • remain within general theory of separation logic

• Also: generic linearizable dictionary algorithm