A specialization calculus for Program Verification

Wei-Ngan Chin, **Cristian Gherghina**,
Razvan Voicu, Quang Loc Le
Florin Craciun, Shengchao Qin

Setting

- Program verification requires a constant balancing act between
  - Having to reason in rich logics in order to verify richer properties
  - Doing this reasoning in a tractable fashion

- Aim:
  - **Diminish the performance penalty incurred by some richer logics**
    - logics with inductive predicates

Focus

- Logics with inductive predicates commonly exhibit:
  - Folding
    - Tasked with abstracting the current state into a predicate instance
  - Unfolding
    - Concretizing the predicate by unfolding its definition
      - Is required in order to refine the available information
    - Big source of performance loss
      - Reasoning about concrete states is costlier than reasoning about abstracted states

Proposal

- We introduce a lightweight, sound and complete calculus
  - With the aim of alleviating the unfolding penalty through the application of predicate specialization

The Framework

- A fragment of Separation Logic with inductive predicates

$$
\begin{array}{lll}
pred & ::= p(v^*) \equiv \Phi & \\
\Phi & ::= \bigvee\ (\exists w^* \cdot \sigma)^* & \sigma ::= \kappa \wedge \pi \\
\kappa & ::= \mathtt{emp} \mid v \mapsto c(v^*) \mid p(v^*) \mid \kappa_1 * \kappa_2 &
\end{array}
$$

Motivation

$$\textbf{data}\ \texttt{node}\ \{\ \texttt{int val};\ \texttt{node prev};\ \texttt{node next};\ \}$$

$$\texttt{dll}(\texttt{root}, \texttt{p}, \texttt{n}, \texttt{S}) \equiv$$

$$\texttt{root}{=}\texttt{null} \wedge \texttt{n}{=}0 \wedge \texttt{S}{=}\{\}$$

$$\vee\ \exists \texttt{v}, \texttt{q}, \texttt{S}_1 \cdot \texttt{root} \mapsto \texttt{node}(\texttt{v}, \texttt{p}, \texttt{q}) * \texttt{dll}(\texttt{q}, \texttt{root}, \texttt{n}{-}1, \texttt{S}_1)$$
$$\wedge \texttt{S} = \texttt{S}_1 \cup \{\texttt{v}\} \wedge \forall \texttt{a} \in \texttt{S}_1 \cdot \texttt{v} \leq \texttt{a}$$

Motivation

$$\mathtt{dll}(\mathtt{root},\mathtt{p},\mathtt{n},\mathtt{S}) \equiv \mathtt{root}{=}\mathtt{null} \wedge \mathtt{n}{=}0 \wedge \mathtt{S}{=}\{\}$$
$$\vee\ \exists \mathtt{v},\mathtt{q},\mathtt{S}_1 \cdot \mathtt{root}{\mapsto}\mathtt{node}(\mathtt{v},\mathtt{p},\mathtt{q}) * \mathtt{dll}(\mathtt{q},\mathtt{root},\mathtt{n}{-}1,\mathtt{S}_1)$$
$$\wedge \mathtt{S} = \mathtt{S}_1 \cup \{\mathtt{v}\} \wedge \forall \mathtt{a} \in \mathtt{S}_1 \cdot \mathtt{v} \leq \mathtt{a}$$

- Given formula: $\mathtt{dll}(\mathtt{x},\mathtt{p}_1,\mathtt{n},\mathtt{S}_1) * \mathtt{dll}(\mathtt{y},\mathtt{p}_2,\mathtt{n},\mathtt{S}_2) \wedge \mathtt{x}{\neq}\mathtt{null}$
  - Can I prove: $S_2 \neq \{\}$?
- Unfolds to:

$$\mathtt{x}{=}\mathtt{null} \wedge \mathtt{y}{=}\mathtt{null} \wedge \mathtt{n}{=}0 \wedge \mathtt{S}_1{=}\{\} \wedge \mathtt{S}_2{=}\{\} \wedge \mathtt{x}{\neq}\mathtt{null}$$
$$\vee\ \mathtt{y}{\mapsto}\mathtt{node}(\mathtt{v}_2,\mathtt{p}_2,\mathtt{q}_2) * \mathtt{dll}(\mathtt{q}_2,\mathtt{y},\mathtt{n}{-}1,\mathtt{S}_4) \wedge \mathtt{x}{=}\mathtt{null}$$
$$\wedge\ \mathtt{S}_1{=}\{\} \wedge \mathtt{n}{=}0 \wedge \mathtt{S}_2{=}\{\mathtt{v}_2\} \cup \mathtt{S}_4 \wedge \mathtt{n}{-}1{\geq}0 \wedge \mathtt{x}{\neq}\mathtt{null}$$
$$\vee\ \mathtt{x}{\mapsto}\mathtt{node}(\mathtt{v}_1,\mathtt{p}_1,\mathtt{q}_1) * \mathtt{dll}(\mathtt{q}_1,\mathtt{x},\mathtt{n}{-}1,\mathtt{S}_3) \wedge \mathtt{y}{=}\mathtt{null} \wedge \mathtt{n}{=}0$$
$$\wedge\ \mathtt{S}_1{=}\{\mathtt{v}_1\} \cup \mathtt{S}_3 \wedge \mathtt{S}_2{=}\{\} \wedge \mathtt{n}{-}1{>}0 \wedge \mathtt{x}{\neq}\mathtt{null}$$
$$\vee\ \mathtt{x}{\mapsto}\mathtt{node}(\mathtt{v}_1,\mathtt{p}_1,\mathtt{q}_1) * \mathtt{y}{\mapsto}\mathtt{node}(\mathtt{v}_2,\mathtt{p}_2,\mathtt{q}_2) * \mathtt{dll}(\mathtt{q}_1,\mathtt{x},\mathtt{n}{-}1,\mathtt{S}_3)$$
$$* \mathtt{dll}(\mathtt{q}_2,\mathtt{y},\mathtt{n}{-}1,\mathtt{S}_4) \wedge \mathtt{S}_1{=}\{\mathtt{v}_1\} \cup \mathtt{S}_3 \wedge \mathtt{S}_2{=}\{\mathtt{v}_2\} \cup \mathtt{S}_4 \wedge \mathtt{n}{-}1{\geq}0 \wedge \mathtt{x}{\neq}\mathtt{null}$$

Motivation

- Tweaking the predicate definition:

$$\mathtt{dll}(\mathtt{root},\mathtt{p},\mathtt{n},\mathtt{S}) \equiv 1{:}(\mathtt{root}{=}\mathtt{null} \wedge \mathtt{n}{=}0 \wedge \mathtt{S}{=}\{\})$$
$$\vee\ 2{:}(\mathtt{root}{\mapsto}\mathtt{node}(\mathtt{v},\mathtt{p},\mathtt{q}) * \mathtt{dll}(\mathtt{q},\mathtt{root},\mathtt{n}{-}1,\mathtt{S}_1)\ \wedge\ \mathtt{S} = \mathtt{S}_1{\cup}\{\mathtt{v}\} \wedge \forall \mathtt{a}{\in}\mathtt{S}_1 \cdot \mathtt{v}{\leq}\mathtt{a})$$

- Annotated predicate instances:

$$\mathtt{dll}(\mathtt{x},\mathtt{p},\mathtt{n},\mathtt{S}) \implies \mathtt{dll}(\mathtt{x},\mathtt{p},\mathtt{n},\mathtt{S})@\boxed{\{1,2\}}$$

- Formula with annotated predicates:

$$\mathtt{dll}(\mathtt{x},\mathtt{p}_1,\mathtt{n},\mathtt{S}_1)@\boxed{\{1,2\}} * \mathtt{dll}(\mathtt{y},\mathtt{p}_2,\mathtt{n},\mathtt{S}_2)@\boxed{\{1,2\}} \wedge \boxed{\mathtt{x}{\neq}\mathtt{null}}$$

- Specialized formula:

$$\mathtt{dll}(\mathtt{x},\mathtt{p}_1,\mathtt{n},\mathtt{S}_1)@\{2\} * \mathtt{dll}(\mathtt{y},\mathtt{p}_2,\mathtt{n},\mathtt{S}_2)@\{2\}$$

Observations

- Predicate annotations simplify considerably the unfolding operation
- Predicate annotations coupled with invariant enrichment can alleviate the need for unfolding

Specification language with annotations

$$
\begin{array}{lll}
spred & ::= p(v^*) \equiv \hat{\Phi}; \mathcal{I}; \mathcal{R} & \\
\hat{\Phi} & ::= \bigvee\ (\exists w^* \cdot \hat{\sigma} \mid C)^* & \hat{\sigma} ::= \hat{\kappa} \wedge \pi \\
\hat{\kappa} & ::= \mathtt{emp} \mid v \mapsto c(v^*) \mid p(v^*) @L\#R \mid \hat{\kappa}_1 * \hat{\kappa}_2 &
\end{array}
$$

- Where:
  - $\mathcal{I}$ denotes a family of predicate invariants
    - One invariant for each set of predicate branches
  - $\mathcal{R}$ $=\{c \mid c = \alpha \longleftarrow L_0\}$ denotes a set of pruning conditions
  - $L \subseteq \mathcal{P}(\mathcal{N})$ denotes a set of viable predicate branches
  - C is a pure formula used in the specialization process, such that C $\longrightarrow \pi$

Dll - annotated

- $dll(root, p, n, S) \equiv$
  $1: (root = null \wedge n = 0 \ \wedge S = \emptyset\ ) \vee$
  $2: \begin{pmatrix} root \longmapsto node(v, p, q) * dll(q, root, n-1, S_1) \wedge \\ S = S_1 \cup \{v\} \wedge \forall a \in S_1 . v \leq a \end{pmatrix}; \mathcal{I}\ ; \mathcal{R}$
- Where:
  - Invariant family $\mathcal{I} = \begin{Bmatrix} (1,2 \rightarrow 0 \leq n), \\ (1 \rightarrow S = \emptyset \wedge\ n = 0 \wedge root = null) \\ (2 \rightarrow S \neq \emptyset \wedge n > 0 \wedge root \neq null) \end{Bmatrix}$
  - Pruning conditions

$$\mathcal{R} = \begin{Bmatrix} (S = \emptyset \rightarrow 1\ ),\ (root = null \rightarrow 1),\ (n = 0 \rightarrow 1) \\ (S \neq \emptyset \rightarrow 2),\ (root \neq null \rightarrow 2),\ (n \neq 0 \rightarrow 2) \\ (0 \leq n\ \rightarrow 1,2) \end{Bmatrix}$$

Specialization transformations

- Predicate specialization

$$p(v^*)@L_1\#R_1 \mid C_1 \xrightarrow{sp} p(v^*)@L_2\#R_2 \mid C_2$$

- Aims for
  - less viable branches -> $L_2 \subseteq L_1$
  - less possible pruning conditions -> $R_2 \subset R_1$
  - Strengthening the context : $C_2 \models C_1$

Predicate specialization

- Given $p(v^*)@L\#R|C$
  1. Pick a pruning condition $(\alpha \longrightarrow L_1) \in R$ that contradicts C
  2. Compute the remaining branches (L_2) by dropping the infeasible branches (L_1) from L
  3. Add the stronger invariant, corresponding to L_2 , $to\ C$
     - Where $(L_2 \rightarrow C_1) \in \mathcal{I}$
  4. Drop irrelevant pruning conditions

$$\frac{\begin{array}{c}[\text{SP}-[\text{PRUNE}]] \\ C \wedge \alpha \Longrightarrow \texttt{false} \quad (\alpha \leftarrow L_0) \in R \quad L \cap L_0 \neq \emptyset \quad L_2 = L - L_0 \\ C_1 = \mathcal{I}nv(p(v^*), L_2) \qquad C \wedge C_1, L_2 \vdash R \xrightarrow{\text{filter}} R_1\end{array}}{p(v^*)@L\#R \mid C \xrightarrow{\text{sp}} p(v^*)@L_2\#R_1 \mid C \wedge C_1}$$

Predicate specialization

$$\mathcal{R} = \left\{ \begin{array}{c} (S = \emptyset \rightarrow 1\,), (x = null \rightarrow 1), \\ (n \neq 0 \rightarrow 2), (n = 0 \rightarrow 1), (S \neq \emptyset \rightarrow 2), \\ (x \neq null \rightarrow 2), (0 \leq n \rightarrow 1{,}2) \end{array} \right\}$$

$$\frac{\begin{array}{c} [\textbf{SP}-[\textbf{PRUNE}]] \\ C \wedge \alpha \Longrightarrow \texttt{false} \quad (\alpha \leftarrow L_0) \in R \quad L \cap L_0 \neq \emptyset \quad L_2 = L - L_0 \\ C_1 = \mathcal{I}nv(p(v^*), L_2) \qquad C \wedge C_1, L_2 \vdash R \xrightarrow{\text{filter}} R_1 \end{array}}{p(v^*)@L\#R \mid C \xrightarrow{\text{sp}} p(v^*)@L_2\#R_1 \mid C \wedge C_1}$$

- $dll(x, p, n, S)@\{1{,}2\}\#\mathcal{R} \wedge n = 0 \mid n = 0$
  - L={1,2} ; C : $n = 0$
  - From $\mathcal{R}$ pick: $(n \neq 0 \rightarrow 2)$
    - Contradicts with C : $n = 0$ -> such checks can be syntactic
    - $\{1{,}2\} \cap \{2\} = \{2\} \neq \emptyset \;\Rightarrow\; L_2 = \{1\}$
    - $C_1 : S = \emptyset \wedge n = 0 \wedge x = null$
    - $S = \emptyset \wedge n = 0 \wedge x = null,\ \{1\} \vdash \mathcal{R} - filter \rightarrow \emptyset$

Predicate specialization

$$\frac{R_f = \{(\alpha \leftarrow L_0) \mid (\alpha \leftarrow L_0) \in R,\ (L \cap L_0 = \emptyset) \vee (C \Longrightarrow \alpha)\}}{C, L \vdash R \xrightarrow{\text{filter}} (R - R_f)}\ [\text{SP}-[\text{FILTER}]]$$

- Given:
  - C : $n = 0 \wedge S = \emptyset \wedge x = null$
  - L : $\{1\}$
- Then:
  - $R = \left\{ \begin{matrix} (x = null \rightarrow 1), & (S \neq 0 \rightarrow 2), & (0 \leq n \rightarrow 1,2) \\ (x \neq null \rightarrow 2), & (S = \emptyset \rightarrow 1), & (n = 0 \rightarrow 1) \end{matrix} \right\}$
  - $R_f = R \Rightarrow R - Rf = \emptyset$
- Result:
  - $dll(x, p, n, S)@\{1\}\#\emptyset \wedge n = 0 \mid n = 0 \wedge S = \emptyset \wedge x = null$

Predicate specialization gains

- Simple implication checks (mostly syntactic)
- Considerable drop in formula size after an unfold
- Increase in formula precision without an unfold

Specialization inference

- We need a mechanism for computing
  - Invariant family $p(v^*) \equiv \bigvee_{i=1}^{n} \mathtt{D_i}; \boxed{\mathcal{I}}; \mathcal{R}$
  - Pruning conditions $p(v^*) \equiv \bigvee_{i=1}^{n} \mathtt{D_i}; \mathcal{I}; \boxed{\mathcal{R}}$

Inferring the invariant family

- Given a predicate definition
  - Compute fixpoint for the predicate definition, the invariant of the entire predicate
  - For each possible set of branches compute a conjunctive invariant
    - Construct the disjunction of all the branches in the current set
    - Approximate the disjunction with one conjunctive invariant

$$spred_{old} = (\mathtt{p}(\mathtt{v}^*) \equiv \bigvee_{i=1}^{n} (\exists u_i^* \cdot \hat{\sigma}_i \mid C_i)) \qquad \rho = [inv_p(\mathtt{v}^*) \mapsto \mathtt{fix}(\bigvee_{i=1}^{n} \exists u_i^* \cdot C_i)]$$

$$\mathcal{I} = \{(L \rightarrow hull(\bigvee_{i \in L} \exists u_i^* \cdot \rho C_i) \mid \emptyset \subset L \subset \{1..n\}\} \cup \{\{1..n\} \rightarrow \rho(inv_p(\mathtt{v}^*))\}$$

$$spred_{new} = (\mathtt{p}(\mathtt{v}^*) \equiv \bigvee_{i=1}^{n} (\exists u_i^* \cdot \hat{\sigma}_i \mid \rho C_i);\ \mathcal{I})$$

Inferring the pruning conditions

- Given a predicate definition and the invariant families
  - Compute an approximation of the transitive closure of each branch invariant
  - For each atomic constraint in all closures construct the list of branches in which it appears (by which it is implied)

$$spred_{old} = (\mathtt{p}(\mathtt{v}^*) \equiv \bigvee_{i=1}^{n} (\exists u_i^* \cdot \hat{\sigma}_i \mid C_i);\ \mathcal{I}) \qquad G = \bigcup_{i=1}^{n} closure(\mathcal{I}(\{i\}))$$

$$\mathcal{R} = \bigcup_{\alpha \in G} \{\alpha \leftarrow \{i \mid 1 \leq i \leq n \wedge \mathcal{I}(\{i\}) \Longrightarrow \alpha\}\} \quad \forall i \in \{1, .., n\} \cdot \hat{\sigma}_i \mid C_i \xrightarrow{\text{sf}}{}^{*} \hat{\sigma}_{i,2} \mid C_{i,2}$$

$$spred_{new} = (\mathtt{p}(\mathtt{v}^*) \equiv \bigvee_{i=1}^{n} (\exists u_i^* \cdot \hat{\sigma}_{i,2} \mid C_{i,2});\ \mathcal{I};\ \mathcal{R})$$

Correctness

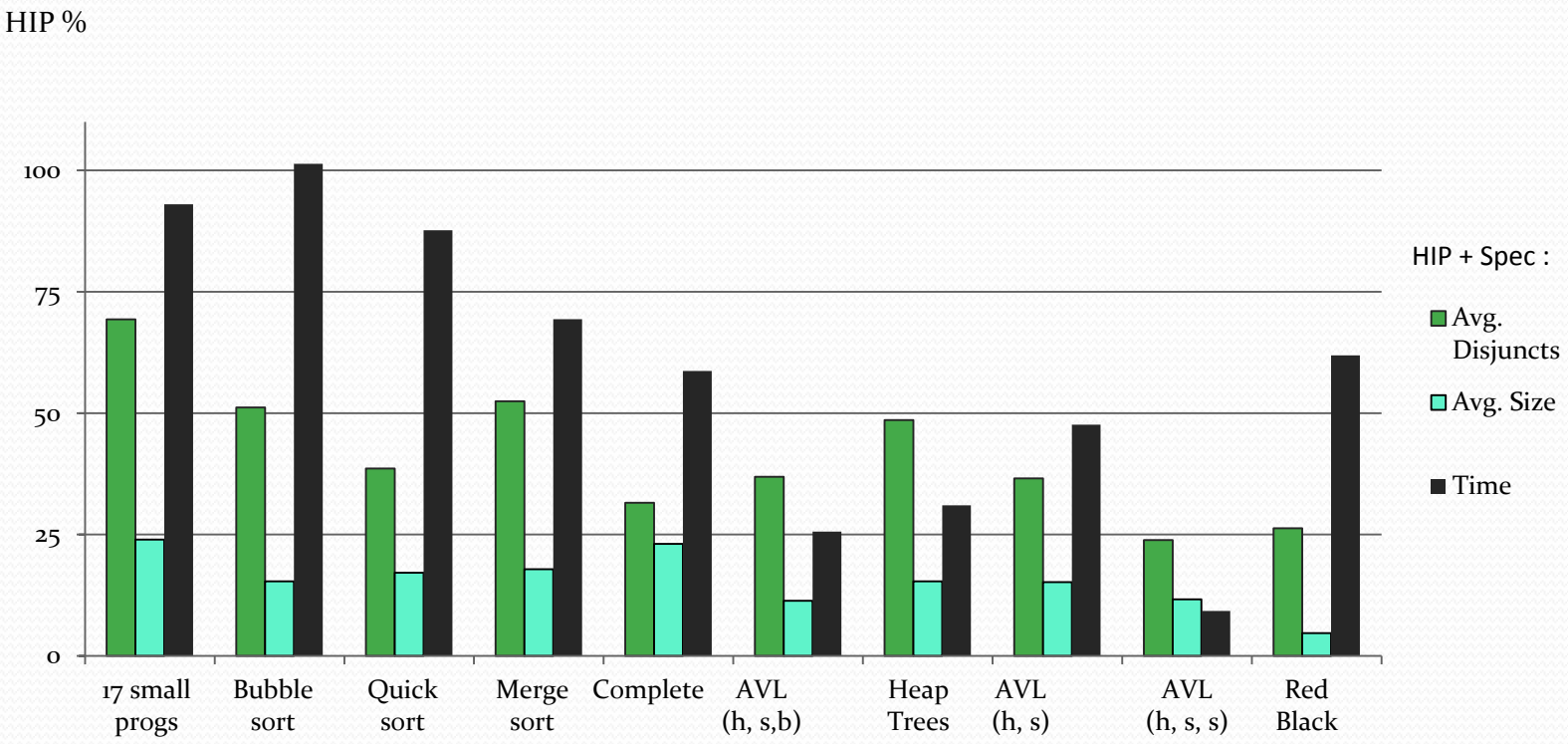
- We have proven the correctness of both
  - The specialization calculus
  - The invariant and pruning conditions inference algorithm

Experiments

- Verified full functional correctness for varied sized tests
  - A benchmark of 17 small programs (7% faster)

    Singly, doubly, sorted and circular linked lists, selection-sort, insertion-sort, methods for handling heaps an perfect trees
  - Complex shapes and invariants (12-90% faster)
    - Red black trees, balanced binary trees, quick sort, merge sort

Experiments



Experiments

Progs (specified props)	LOC	HIP	HIP+Spec	HIP			HIP+Spec		
		Time(s)	Time(s)	Count	Disj	Size	Count	Disj	Size
17 small progs (size)	87	0.86	0.80	229	1.63	12.39	612	1.13	2.97
Bubble sort (size,sets)	80	2.20	2.23	296	2.13	18.18	876	1.09	2.79
Quick sort (size,sets)	115	2.43	2.13	255	3.29	17.97	771	1.27	3.08
Merge sort (size,sets)	128	3.10	2.15	286	2.04	16.74	1079	1.07	2.99
Complete(size,minh)	137	5.01	2.94	463	3.52	43.75	2134	1.11	10.10
AVL (height, size,bal)	160	64.1	16.4	764	2.90	85.02	6451	1.07	9.66
Heap Trees(size, maxel)	208	14.9	4.62	649	2.10	56.46	2392	1.02	8.68
AVL (height, size)	340	27.5	13.1	704	2.98	70.65	7078	1.09	10.74
AVL (height, size, sets)	500	657	60.7	1758	8.00	86.79	14662	1.91	10.11
Red Black (size, height)	630	25.2	15.6	2225	3.84	80.91	7697	1.01	3.79

Conclusions

- We have
  - Presented a simple, flexible calculus for specialization
  - Proven it sound
  - We have exposed considerable performance gains in a general logic framework

Questions?

Thank you!