Towards Abstraction Refinement in TVLA

Alexey Loginov
UW Madison
alexey@cs.wisc.edu
Need for Heap Data Analysis

• Talks by WiSA students
  – Static analyses for de-obfuscation of code
  – Limited ability to analyze heap data
  – Need understanding of possible shapes
    • Want to handle unbounded heap object creation
    • Java objects (e.g. threads) are in heap
Linked List Abstractions

• Informally

• Formally
Linked List Abstractions

- Informally

- Formally
Linked List Abstractions

• Informally

• Formally

\[ x \rightarrow \quad \cdots \]
\[ y \rightarrow \quad \cdots \]

\[ x \rightarrow r_x \rightarrow \quad r_x \]
\[ y \rightarrow r_y \rightarrow \quad r_y \]
Linked List Abstractions

• Informally

• Formally
Abstraction Refinement

- Devising good analysis abstractions is hard
  - Precision/cost tradeoff
    - Too coarse: “Unknown” answer
    - Too refined: high space/time cost
- Start with simple (and cheap) abstraction
- Successively refine abstraction
  - Adaptive algorithm
Abstraction Refinement

- Iterative process
  - Create an abstraction (e.g. set of predicates)
  - Run analysis
  - Detect indefinite answers (stop if none)
  - Refine abstraction (e.g. add predicates)
  - Repeat above steps
Previous Work on Abstraction Refinement

- Counterexample guided [Clarke et al]
  - Finds shortest invalid prefix of spurious counterexample
  - Splits last state on prefix into less abstract ones
- SLAM toolkit [Ball, Rajamani]
  - Temporal safety properties of software
  - Identifies correlated branches
Abstraction Refinement for TVLA: New Challenges

• Need to refine abstractions of linked data structures
  – Identify appropriate new distinctions between
    • Nodes
    • Structures

• Need to derive the associated abstract interpretation
  – What are the update formulas?
Control Over the Merging of Nodes

• Unary abstraction predicates
  \[ r_x(v) = \exists v' : (x(v') \land n^*(v',v)) \]
  • Distinguish nodes reachable from x (and y)
  • Can now tell if lists are disjoint
Control Over the Merging of Structures

• Nullary abstraction predicates

\[ nn_x() = \exists v : x(v) \]

• Distinguish structures based on whether x is NULL
• Can now tell that x is NULL whenever y is (and vice versa)
Need for Update Formulas

- Re-evaluating formulas is imprecise

\[ r_x(v) = \exists v' : (x(v') \triangleright n^*(v',v)) \]

Action “\(x = x->n\)”
Need for Update Formulas

- Re-evaluating formulas is imprecise

\[ r_x(v) = \exists v' : (x(v') \leftarrow n^*(v',v)) \]

Action “\(x = x->n\)”
Goal: Create Update Formulas Automatically

• Currently: user provides all update formulas
  – A lot of work
  – Error prone
  – Precludes iterative refinement

• Idea: Finite differencing of formulas

\[ F(\phi) = p_\phi \phi^\Delta \]
Past Work on Finite Differencing

- Paige: SETL
- Horwitz & Teitelbaum: Relational Algebra
- Liu & Teitelbaum: Functional Programs
Database View Maintenance

- DB' - updated database
- U – database update
- ϕ - query
- A – answer
Finite Differencing of first-order formulas

- $S'$ - updated core structure
- $\tau$ – core update
- $\phi_p$ – instrumentation predicate formula
- $S_{\text{inst}}$ – instrumentation structure
\[ 0^\Delta = 0 \]
\[ 1^\Delta = 0 \]
\[ (\varphi \psi)^\Delta = \varphi^\Delta \psi^\Delta \]
\[ (\varphi \psi)^\Delta = \varphi^\Delta \psi^\Delta \]
\[ (\varphi \Delta \psi)^\Delta = \varphi^\Delta \psi^\Delta \]
\[ (\varphi \Delta \psi)^\Delta = \varphi^\Delta \psi^\Delta \]

\[ \text{Differencing } \approx \text{ Differentiation} \]

\[ c' = 0 \]
\[ (f + g)'(x) = f'(x) + g'(x) \]
\[ (f \cdot g)'(x) = f'(x) \cdot g(x) + f(x) \cdot g'(x) \]
Finite Differencing of Formulas

• $F(\phi) = p_\phi \varphi^\Delta$ - too naïve
  – yields $\frac{1}{2}$ when either argument is $\frac{1}{2}$
  – No ability to “generate” or “kill” facts

• Idea: split into negative and positive change
  – $F(\phi) = p_\phi \ ? \neg \Delta^-(\phi) : \Delta^+(\phi)$
  – More precise
  – Natural for static analysis problems
Laws for $\Delta^+$

$\Delta^+(0) = 0 \quad \Delta^+(1) = 0$

$\Delta^+(\neg \varphi) = \Delta^-(\varphi)$

$\Delta^+(\varphi \land \psi) = (\Delta^+(\varphi) \land \neg \psi) \lor (\neg \varphi \land \psi)$

$\Delta^+(\psi))$

$\Delta^+(\exists v : \varphi) = (\exists v : \Delta^+(\varphi)) \lor (\neg (\exists v : \varphi))$
Example: reachability

$$\Delta^+(r_x(v)) = \Delta^+(\exists v' : (x(v') \mathbin{\rightarrow} n^*(v',v)))$$

$$= (\exists v' : \Delta^+(x(v') \mathbin{\rightarrow} n^*(v',v)))$$

$$\mathbin{\rightarrow} (\exists v : (x(v') \mathbin{\rightarrow} n^*(v',v)))$$

$$= (\exists v' : \Delta^+(x(v')) \mathbin{\rightarrow} F(n^*(v',v))) \mathbin{\rightarrow} F(x(v')) \mathbin{\rightarrow} \Delta^+(n^*(v',v)))$$

$$\mathbin{\rightarrow} (\exists v' : (x(v') \mathbin{\rightarrow} n^*(v',v)))$$
Evaluation

• Programs
  – Singly linked list manipulation
  – Doubly linked list manipulation
  – SLL sorting routines
  – Information flow tests

• Measure of success
  – Fraction of automatically generated formulas without loss of precision relative to best hand-crafted formulas
  – Greedy exploration
Results

• More than half of formulas auto generated
  – Worst case: 0 of 2, best case 11 of 15
  – All problems due to transitive closure
• Analysis time increase
  -3% (decrease) to 9% (avg. <2%)
Example revisitted: reachability

\[ \Delta^+(r_x(v)) = \]
\[
(\exists v' : \Delta^+(x(v'))) \quad F(n^*(v',v))
\]
\[
F(x(v')) \quad \Delta^+(n^*(v',v)))
\]
\[
(\exists v' : (x(v') \quad n^*(v',v)))
\]
Solution: factor out TC

• Use power of instrumentation
  – Save TC info in instrumentation predicate
  – $t_n(v_1, v_2) = n^*(v_1, v_2)$

• Use in other instrumentation predicates
  – $r_x(v) = \exists v' : x(v') \bowtie t_n(v', v)$
  – $c_n(v) = \exists v' : n(v', v) \bowtie t_n(v, v')$
Example revisited: reachability

\[ \Delta^+(r_x(v)) = \]
\[ (\exists v' : \Delta^+(x(v'))) \quad F(t_n(v',v)) \]
\[ F(x(v')) \quad \Delta^+(t_n(v',v))) \]
\[ \neg(\exists v' : (x(v') \quad t_n(v',v))) \]
Results (improved)

• Almost 90% of formulas auto generated
  – Worst case: 9 of 11, best cases 3 of 3, 11 of 12
  – Only two kinds of hand-crafted updates needed
    • predicate $t_n(v1,v2)$ for $x->n = \text{null}$
    • predicate $t_n(v1,v2)$ for $x->n = y$

• Analysis time increase
  -5% (decrease) to 11% (avg. <4%)
Conclusions

- **FD** – fully handles first-order formulas
- **Performance effect** – minimal
- **Challenges** – Transitive Closure
  - Updating TC information
  - Preserving unchanged TC information
    - Key: avoid recomputation
Future Work

• Generation of new instrumentation predicates
  – Formula generation
  – Generate update formulas via finite differencing
• Abstraction refinement loop
Towards Abstraction Refinement in TVLA

Alexey Loginov
UW Madison
alexey@cs.wisc.edu
Laws for $\Delta^+$

$\Delta^+(0) = 0$  $\Delta^+(1) = 0$

$\Delta^+(v = w) = 0$

$\Delta^+(\neg \varphi) = \Delta^-(\varphi)$

$\Delta^+(\varphi \land \neg \psi) = (\Delta^+(\varphi) \land \neg \psi) \land (\neg \varphi \land \Delta^+(\psi))$

$\Delta^+(\varphi \lor \psi) = (\Delta^+(\varphi) \lor F(\psi)) \lor (F(\varphi) \lor \Delta^+(\psi))$

$\Delta^+(\psi))$

$\Delta^+(\exists v : \varphi) = (\exists v : \Delta^+(\varphi)) \lor \neg(\exists v : \varphi)$

$\Delta^+(\exists v : \varphi) = (\exists v : \Delta^+(\varphi)) \lor (\not\exists v : F(\varphi))$