Towards Abstraction Refinement in TVLA

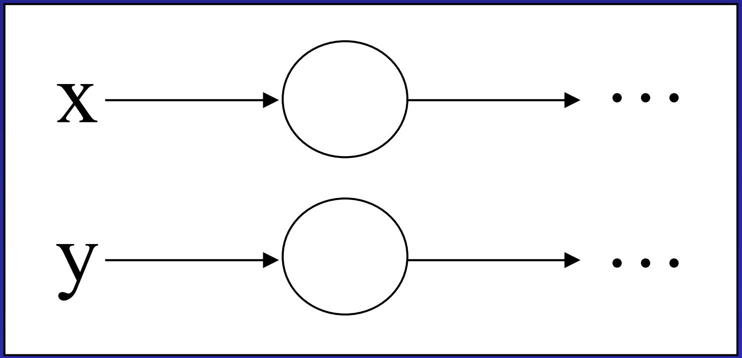
Alexey Loginov
UW Madison
alexey@cs.wisc.edu

Need for Heap Data Analysis

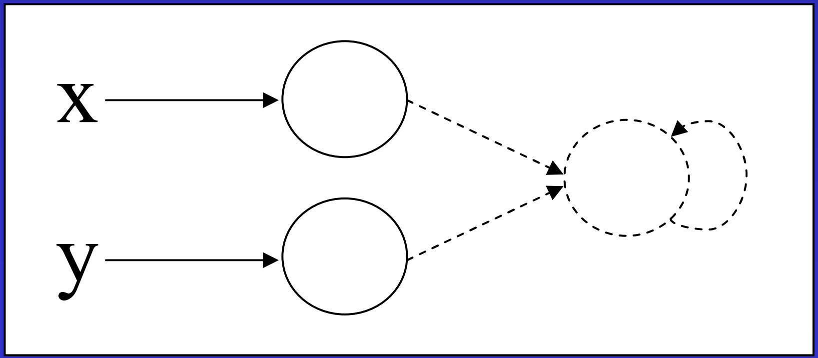
- Talks by WiSA students
  - Static analyses for de-obfuscation of code
  - Limited ability to analyze heap data
  - Need understanding of possible shapes
    - Want to handle unbounded heap object creation
    - Java objects (e.g. threads) are in heap

Linked List Abstractions

- Informally

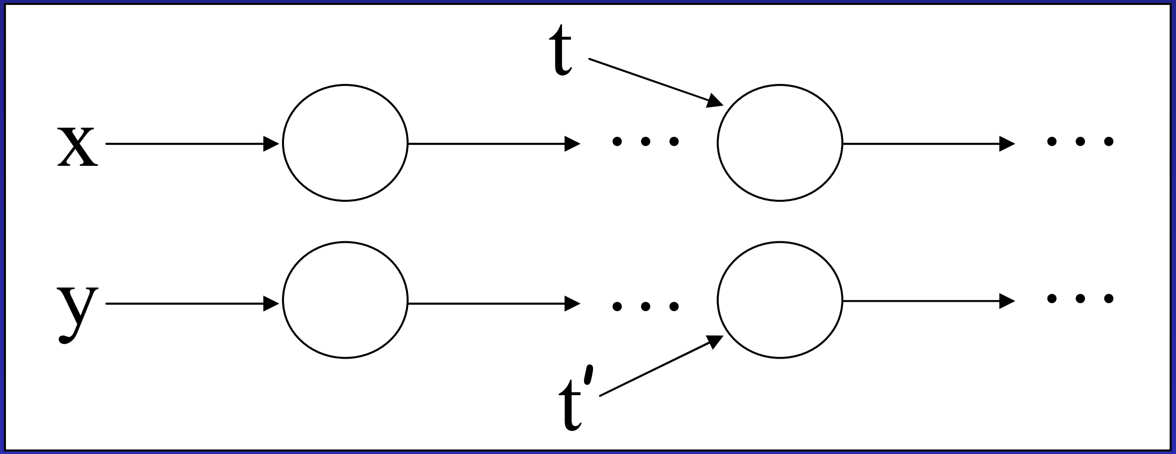


- Formally

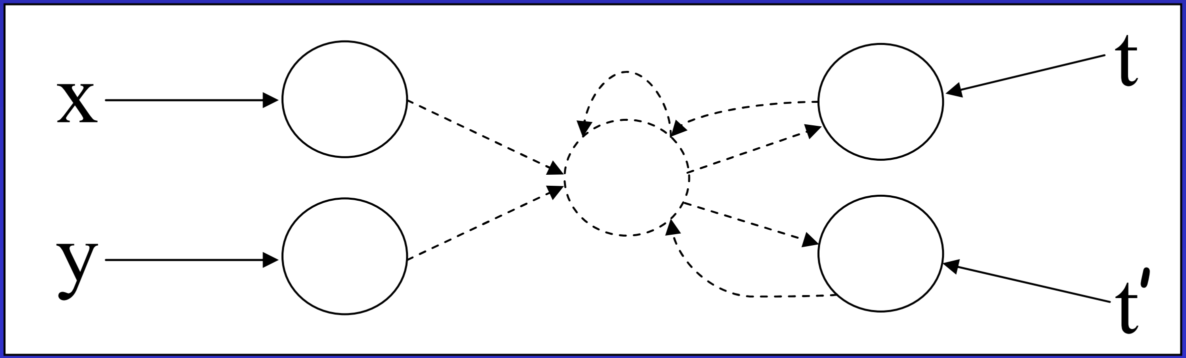


Linked List Abstractions

- Informally

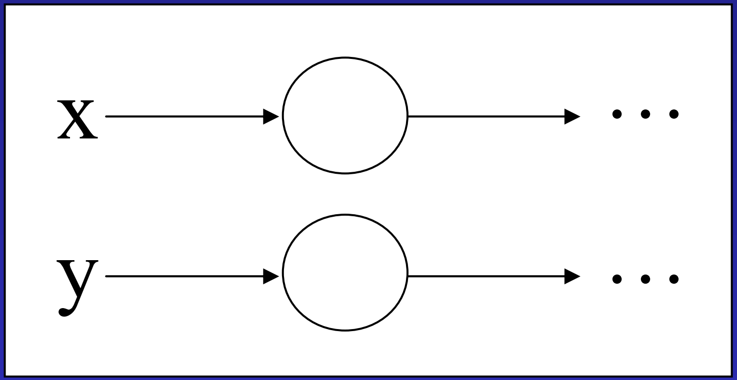


- Formally

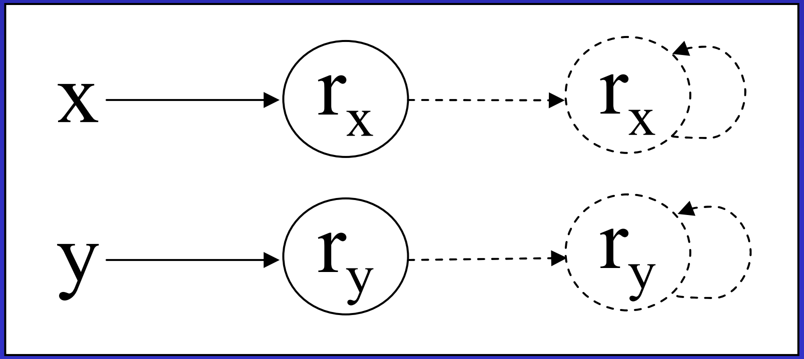


Linked List Abstractions

- Informally

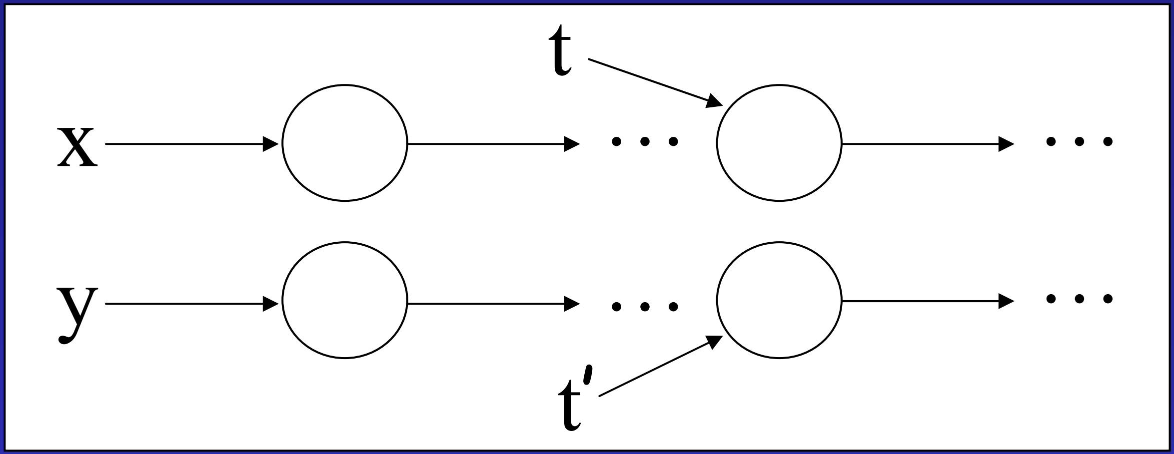


- Formally

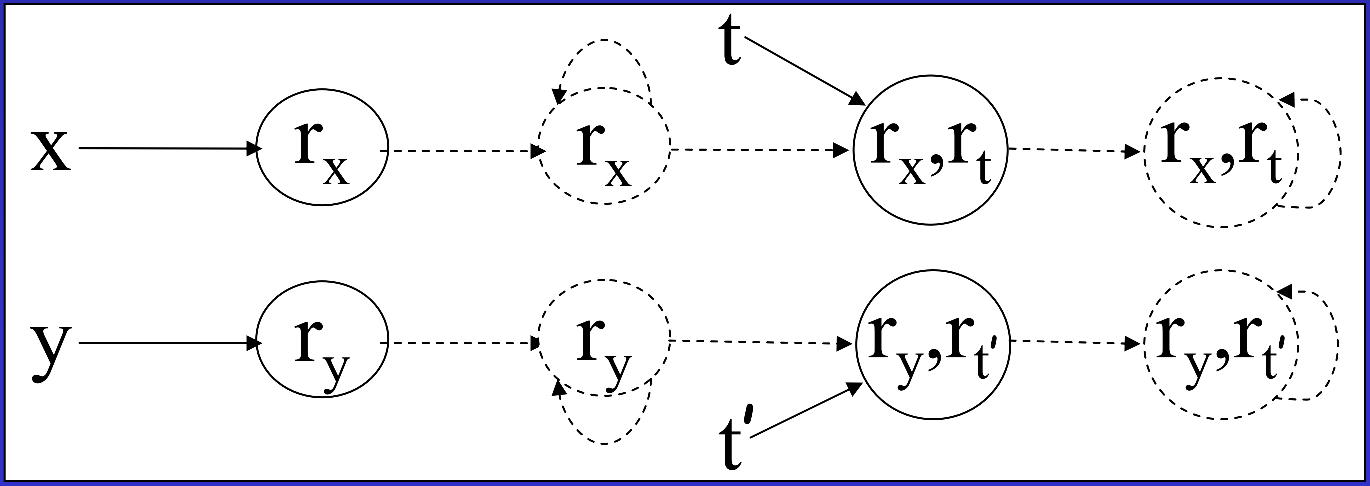


Linked List Abstractions

- 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

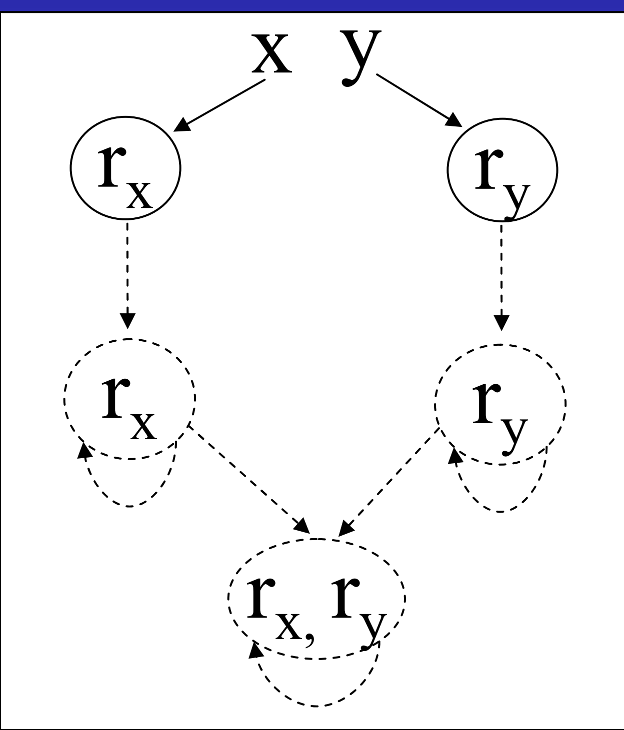
- Need to refine abstractions of linked data structures
  - Identify appropriate new distinctions between
    - Nodes
    - Structures
- Need to derive the associated abstract interpretation
  - What are the update formulas?

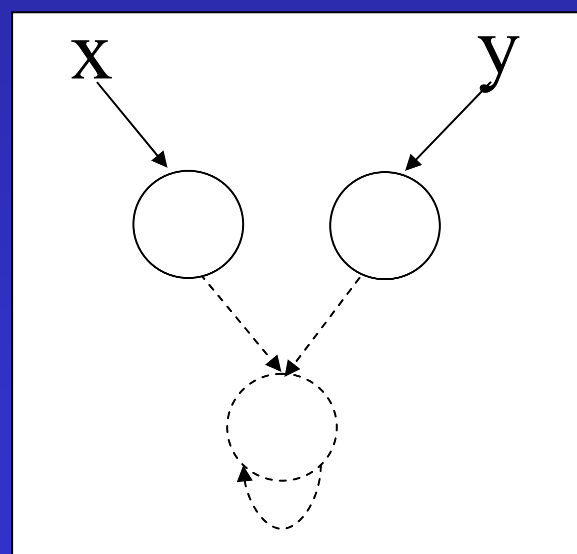
Control Over the Merging of Nodes

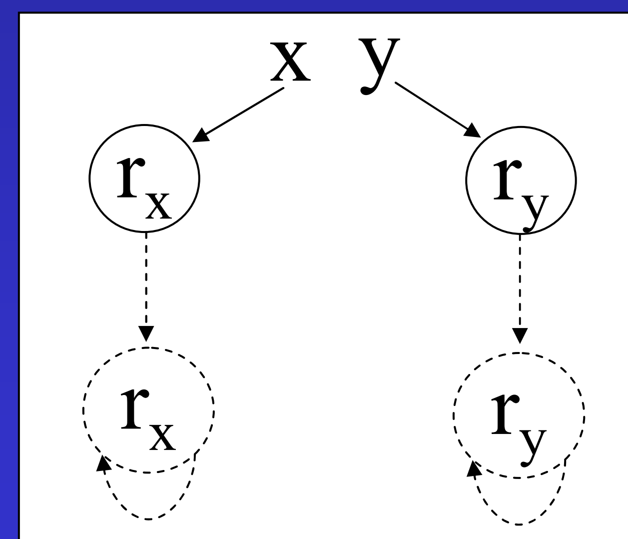
- Unary abstraction predicates

  $r_x(v) = \exists v' : (x(v') \wedge n^*(v',v))$

  - Distinguish nodes reachable from x (and y)
  - Can now tell if lists are disjoint





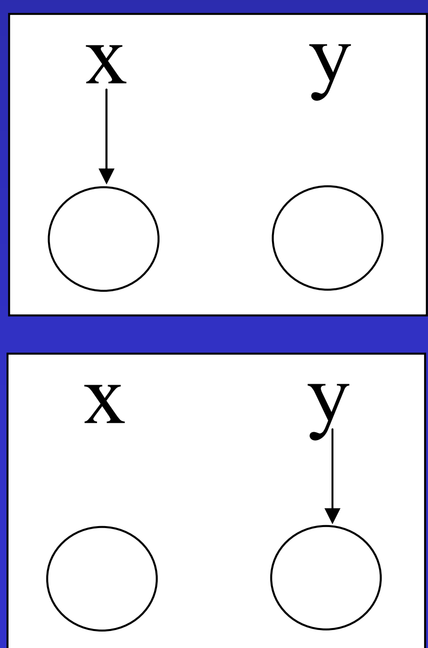


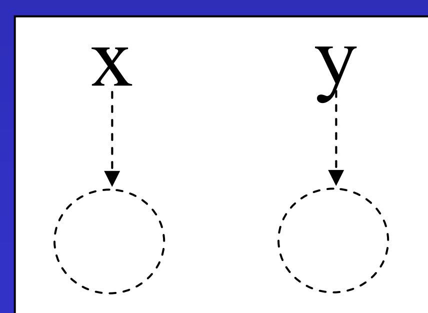
Control Over the Merging of Structures

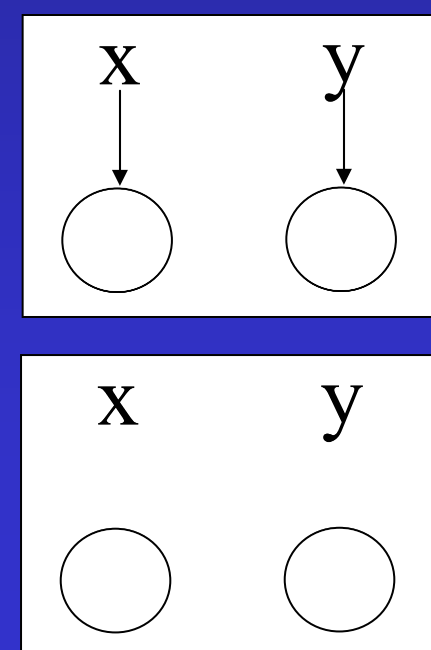
- Nullary abstraction predicates

  $nn_x() = \exists v : x(v)$

  - Distinguish structures based on whether x is NULL
  - Can now tell that x is NULL whenever y is (and vice versa)





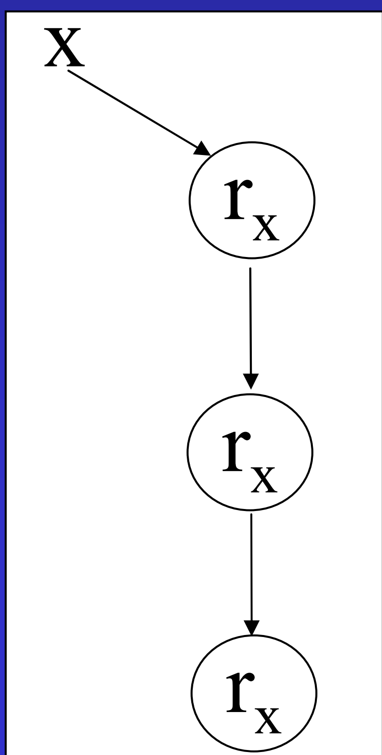


Need for Update Formulas

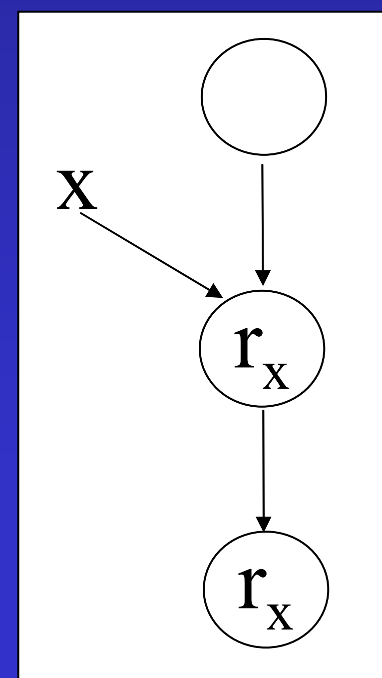
- Re-evaluating formulas is imprecise

  $r_x(v) = \exists v' : (x(v') \land n^*(v',v))$

  Action “x = x->n”



x = x->n

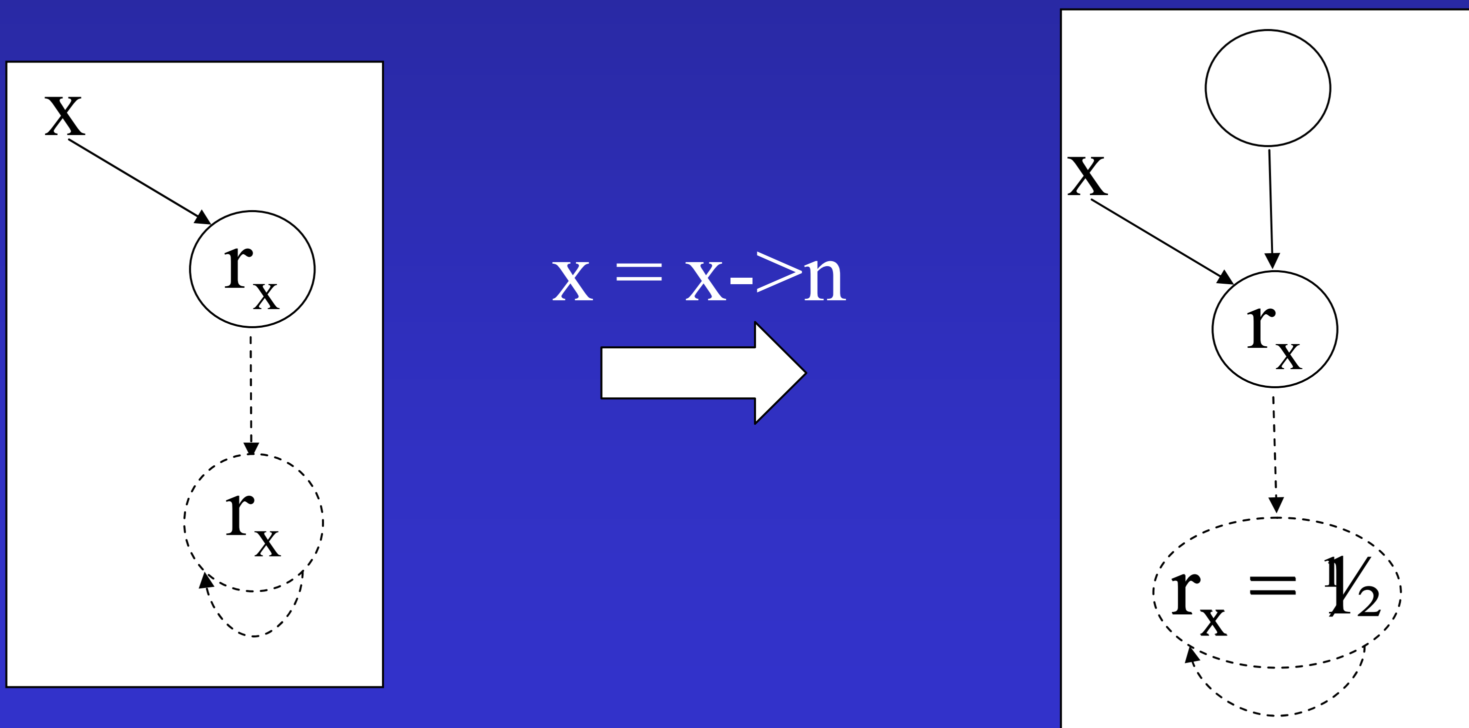


Need for Update Formulas

- Re-evaluating formulas is imprecise

  $r_x(v) = \exists v' : (x(v') \wedge n^*(v',v))$

  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

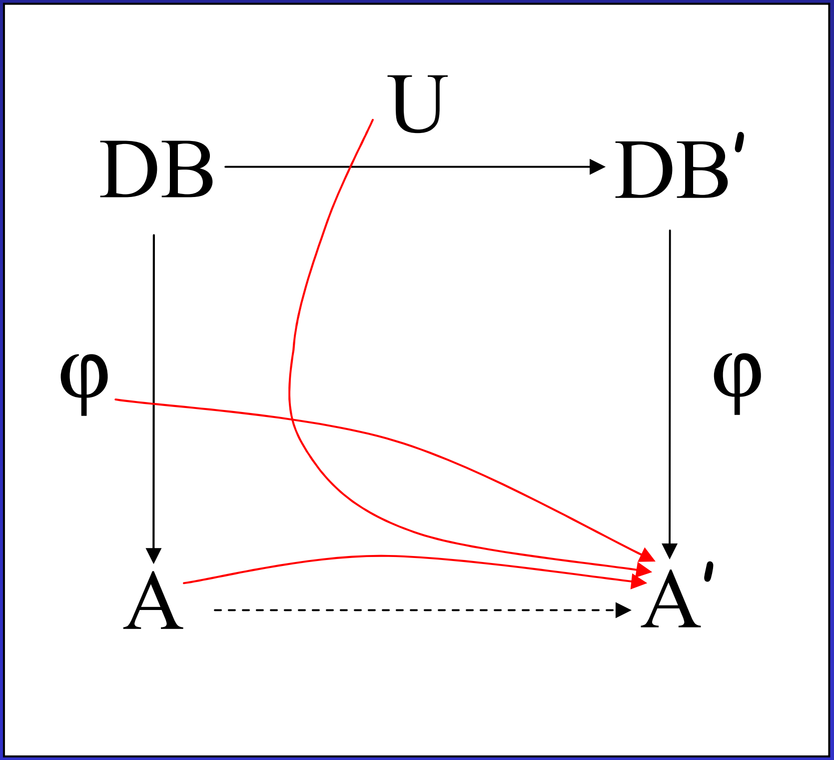
$F(\varphi) = p_{\varphi}$ 📭 φ^{Δ}

Past Work on Finite Differencing

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

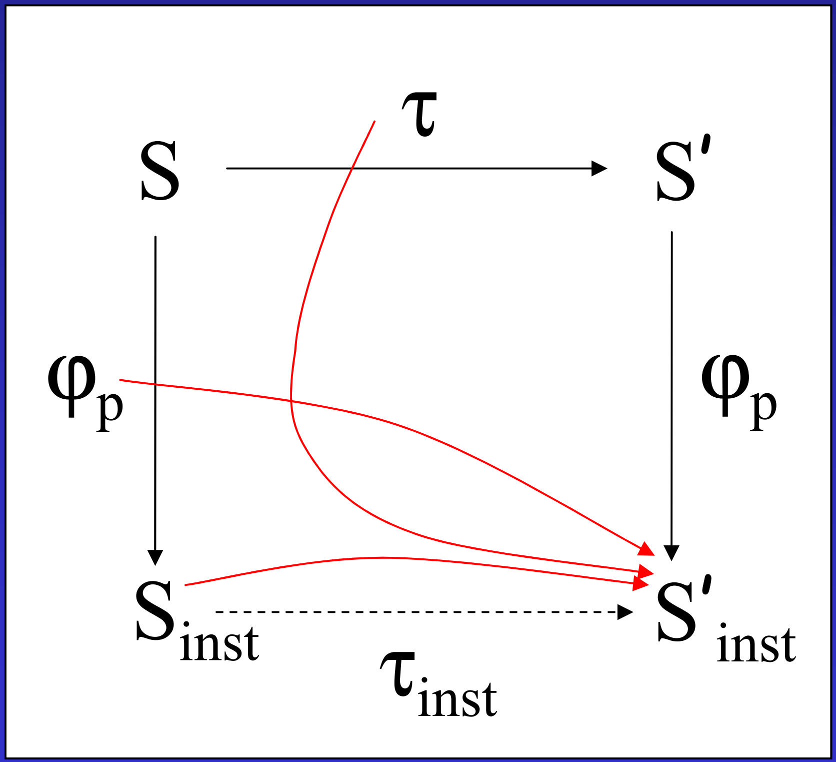
Database View Maintenance

- DB′ - updated database
- U – database update
- φ - query
- A – answer



Finite Differencing of first-order formulas

- S' - updated core structure
- τ – core update
- φ_p – instrumentation predicate formula
- S_{inst} – instrumentation structure



Differencing ≈ Differentiation

$0^{\Delta} = 0$

$1^{\Delta} = 0$

$(\varphi \text{📪} \psi)^{\Delta} = \varphi^{\Delta} \text{📪} \psi^{\Delta}$

$$(\varphi \text{📬} \psi)^{\Delta} = \varphi \text{📬} \psi^{\Delta}$$
$$\text{📪} \; \varphi^{\Delta} \text{📬} \psi$$
$$\boxed{\text{📪} \; \varphi^{\Delta} \text{📬}} \psi^{\Delta}$$

$c' = 0$

$(f+g)'(x) = f'(x)+g'(x)$

$$(f*g)'(x) = f'(x)*g(x)$$
$$+ f(x)*g'(x)$$

Finite Differencing of Formulas

- $F(\varphi) = p_\varphi$ φ^Δ - too naïve
  - – yields ½ when either argument is ½
  - – No ability to "generate" or "kill" facts
- Idea: split into negative and positive change
  - – $F(\varphi) = p_\varphi \;?\; \neg\Delta^-(\varphi) : \Delta^+(\varphi)$
  - – More precise
  - – Natural for static analysis problems

Laws for Δ^+

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

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

$\Delta^+(\varphi$📫$\psi) = (\Delta^+(\varphi)$ 📬 $\neg\psi)$ 📫 $(\neg\varphi$ 📬 $\Delta^+(\psi))$

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

Example: reachability

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

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

$$\land \neg(\exists v : (x(v') \land n^*(v',v)))$$

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

$$F(x(v')) \land \Delta^{+}(n^*(v',v)))$$

$$\land \neg(\exists v' : (x(v') \land n^*(v',v)))$$

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

$$\Delta^{+}(r_x(v)) =$$
$$(\exists v' : \Delta^{+}(x(v')) \wedge F(n^*(v',v)) \vee$$
$$F(x(v')) \wedge \Delta^{+}(n^*(v',v)))$$
$$\wedge \neg(\exists v' : (x(v') \wedge n^*(v',v)))$$

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
  - $r_x(v) = \exists v' : x(v') \wedge t_n(v',v)$
  - $c_n(v) = \exists v' : n(v',v) \wedge t_n(v,v')$

Example revisited: reachability

$$\Delta^{+}(r_x(v)) =$$
$$(\exists v' : \Delta^{+}(x(v')) \wedge F(t_n(v',v)) \vee$$
$$F(x(v')) \wedge \Delta^{+}(t_n(v',v)))$$
$$\wedge \neg(\exists v' : (x(v') \wedge t_n(v',v)))$$

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 = 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 Δ^+

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

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

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

$\Delta^+(\varphi \vee \psi) = (\Delta^+(\varphi) \wedge \neg\psi) \vee (\neg\varphi \wedge \Delta^+(\psi))$

$\Delta^+(\varphi \wedge \psi) = (\Delta^+(\varphi) \wedge F(\psi)) \vee (F(\varphi) \wedge \Delta^+(\psi))$

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

$\Delta^+(\forall v : \varphi) = (\exists v : \Delta^+(\varphi)) \wedge (\forall v : F(\varphi))$