Towards Abstraction Refinement in TVLA

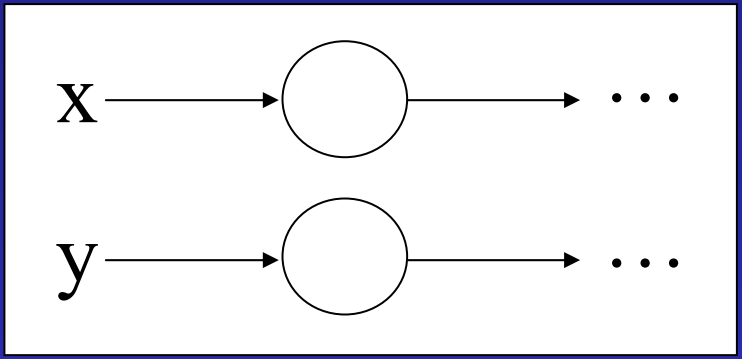
Alexey Loginov
UW Madison
alexey@cs.wisc.edu

Need for Heap Data Analysis

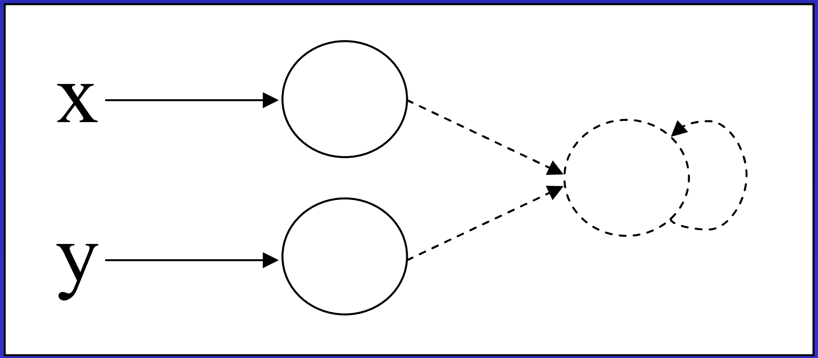
- Morning talks by UW students
  - Static analyses for de-obfuscation of code
  - Limited ability to analyze heap data
  - Need understanding of possible shapes
    - Want to handle unbounded heap object creation
    - Java objects (e.g. threads) are in heap

Linked List Abstractions

- Informally

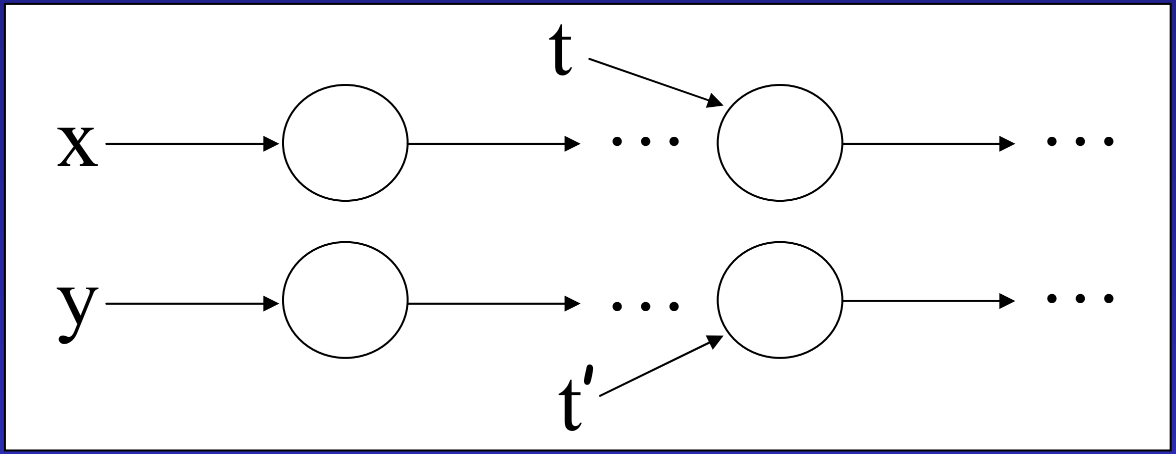


- Formally

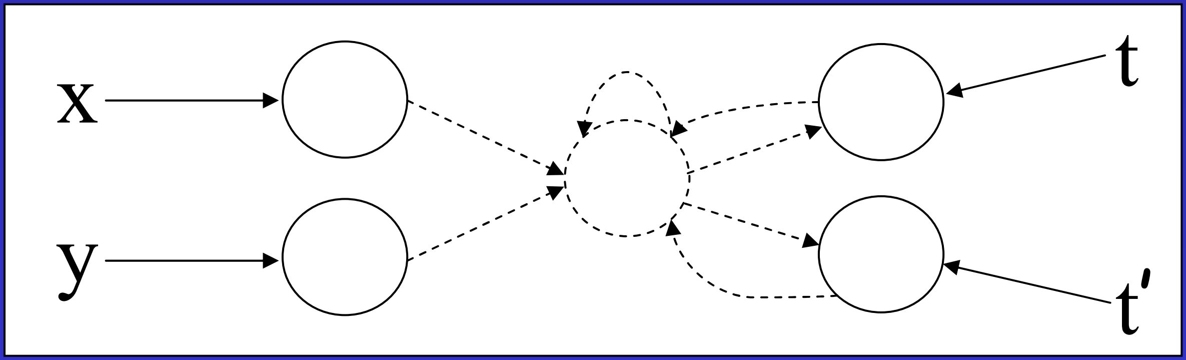


Linked List Abstractions

- Informally

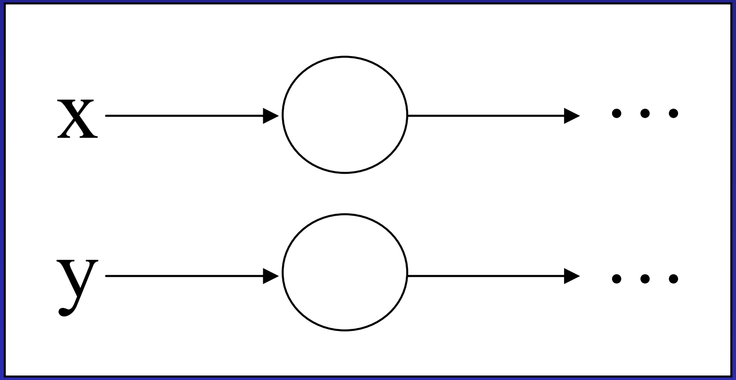


- Formally

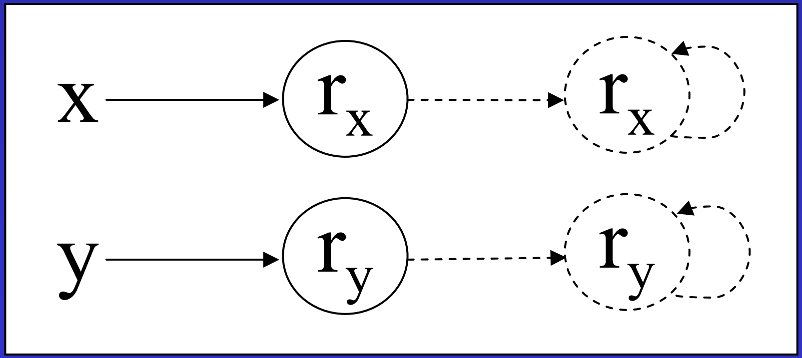


Linked List Abstractions

- Informally

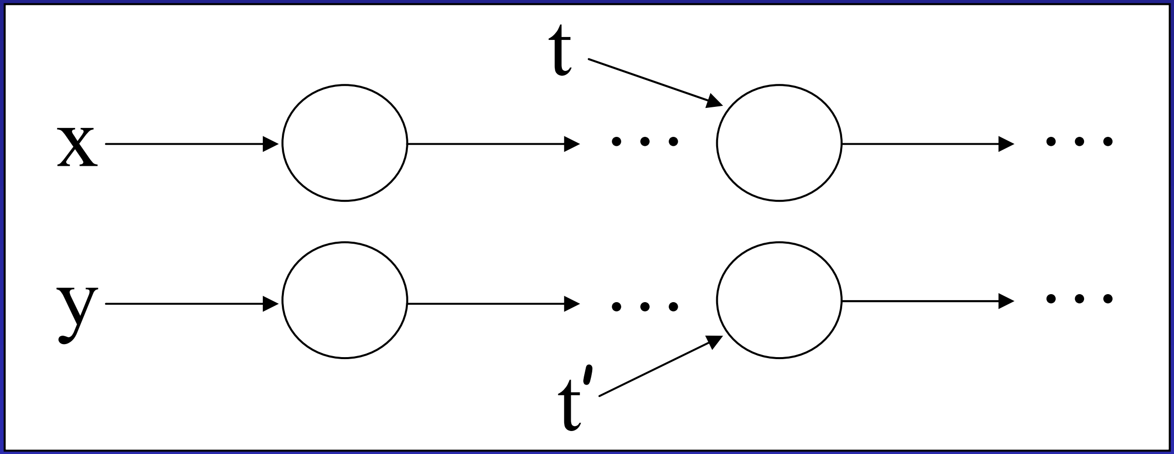


- Formally

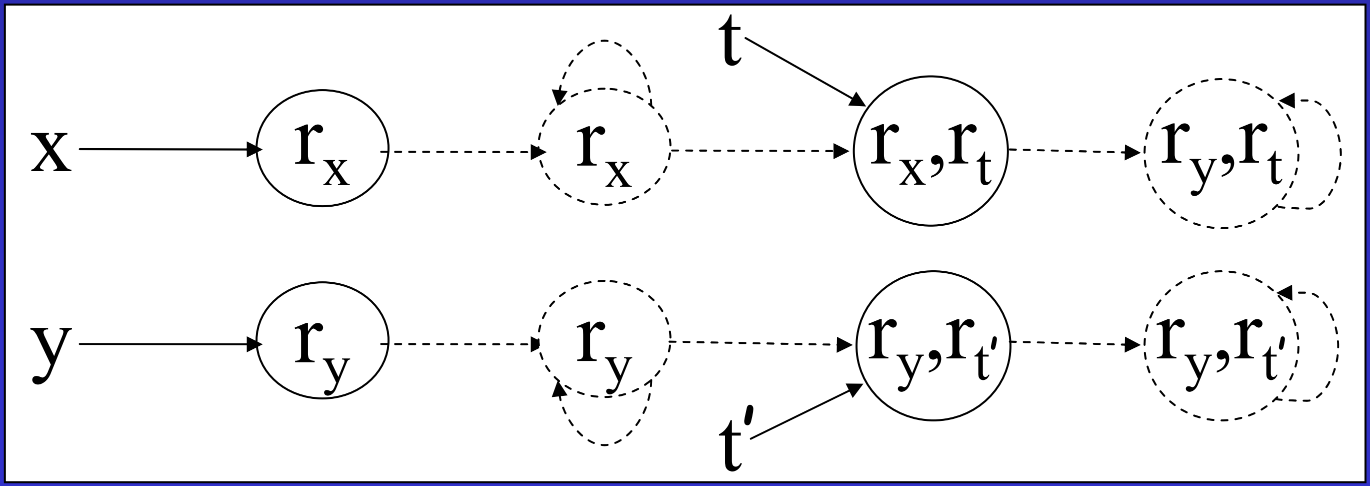


Linked List Abstractions

- 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

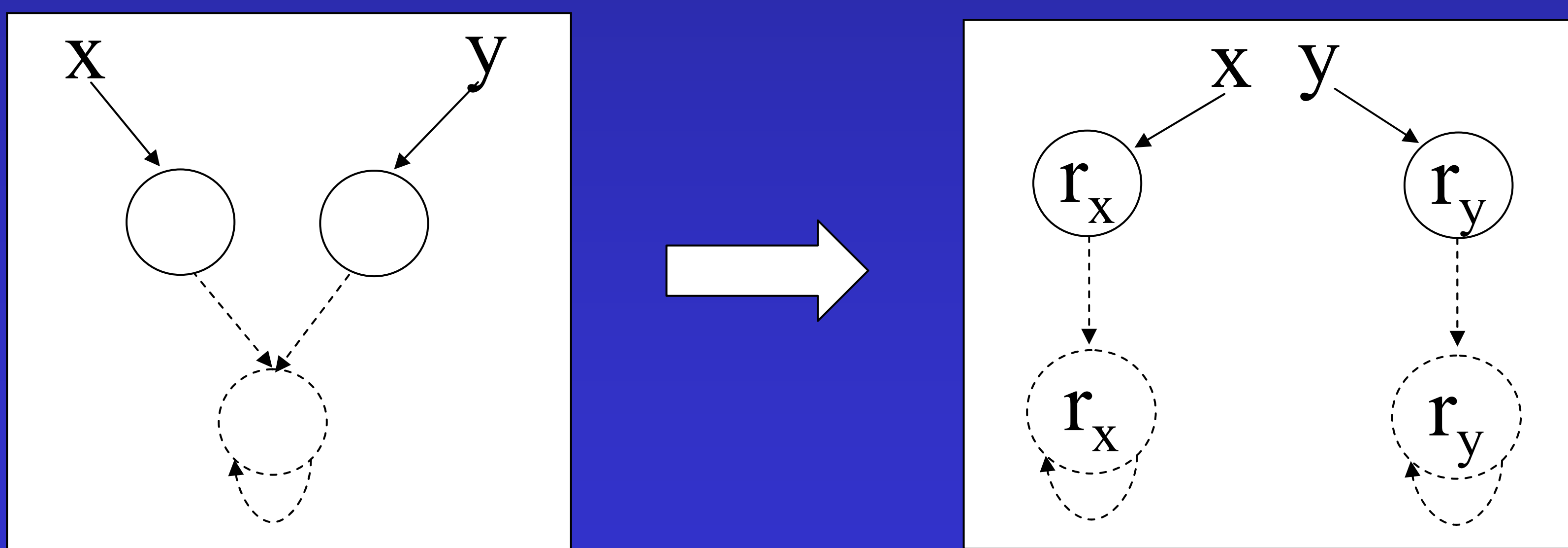
- Need to refine abstractions of linked data structures
  - Identify appropriate new distinctions between
    - Nodes
    - Structures
- Need to derive the associated abstract interpretation
  - What are the update formulas?

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

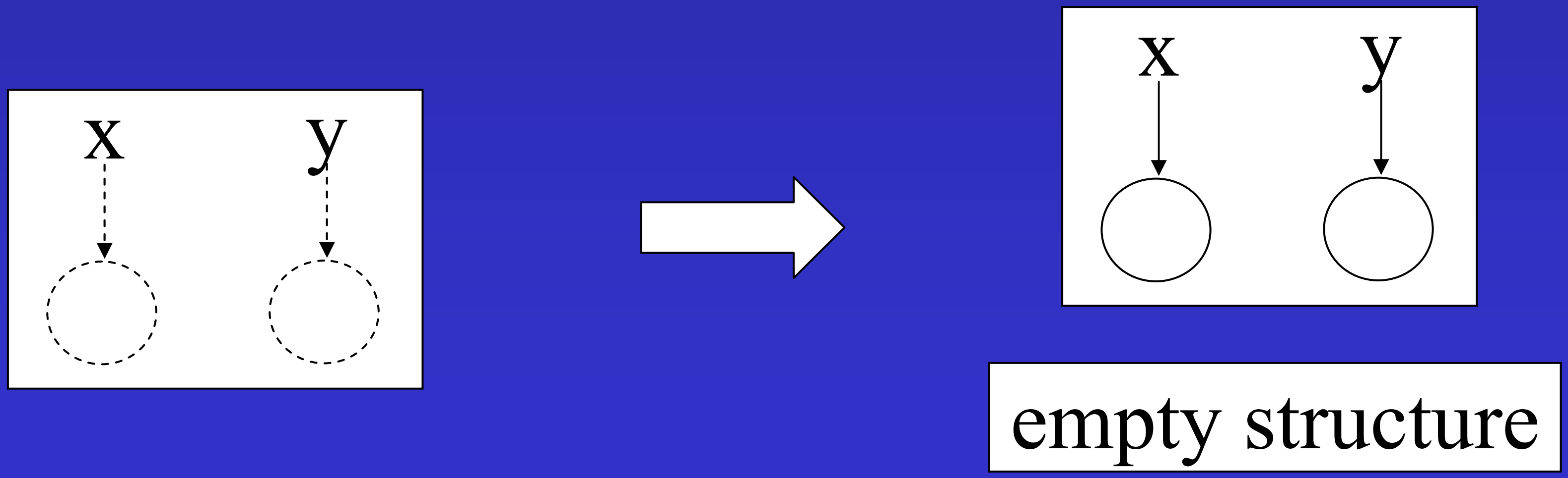


Control Over the Merging of Structures

- 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)

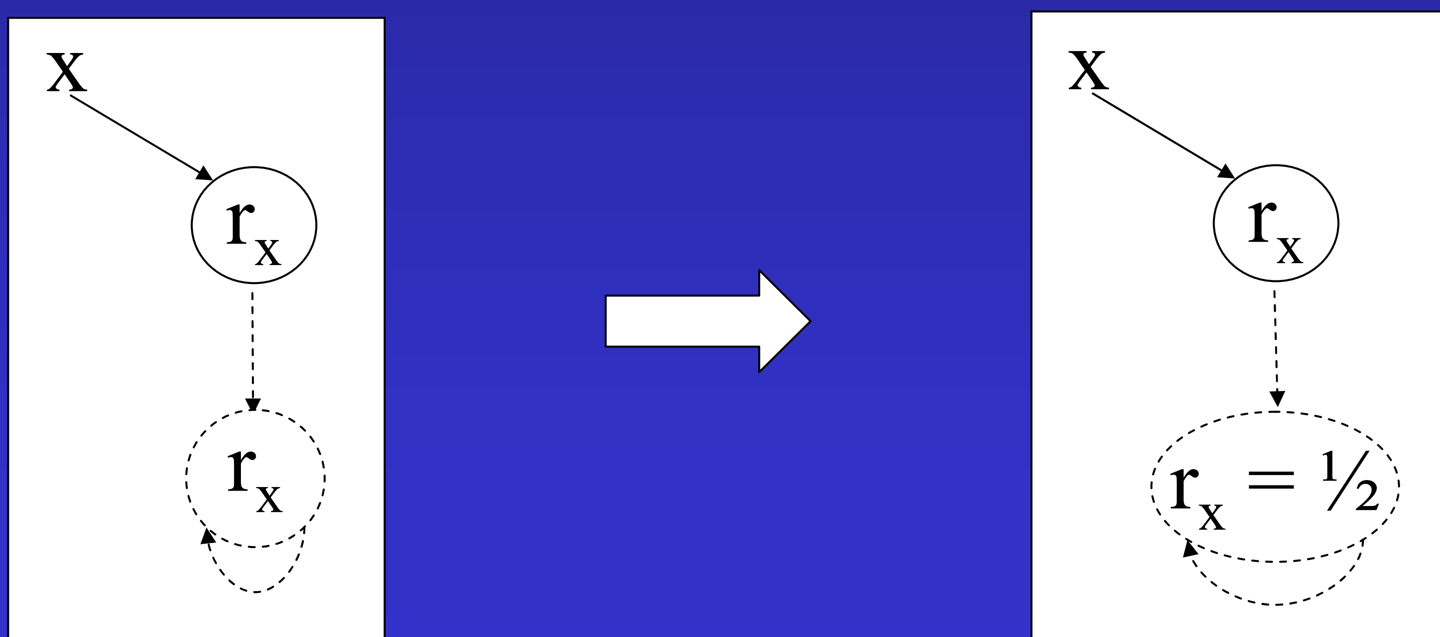


Need for Update Formulas

- Re-evaluating formula is imprecise

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

  Action "x == NULL" (no change to heap)



Creating Update Formulas Automatically

- Frees user from having to provide update formulas
  - A lot of work (esp. with iterative refinement)
  - Error prone
- Idea: Finite differencing of formulas

$F(\varphi) = p_{\varphi}$ 📪 φ^{Δ}

Differencing ≈ Differentiation

$0^{\Delta} = 0$

$1^{\Delta} = 0$

(φ📪ψ)$^{\Delta}$ = φ$^{\Delta}$📪ψ$^{\Delta}$

(φ📫ψ)$^{\Delta}$ = φ📫ψ$^{\Delta}$
📪 φ$^{\Delta}$📫ψ
📪 φ$^{\Delta}$📫ψ$^{\Delta}$

$c' = 0$

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

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

Laws for φ^{Δ}

$0^{\Delta} = 0$

$1^{\Delta} = 0$

$(v = w)^{\Delta} = 0$

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

$(\varphi$📫$\psi)^{\Delta} = \varphi$📫ψ^{Δ} 📪 φ^{Δ}📫ψ 📪 φ^{Δ}📫ψ^{Δ}

$($🕕$v{:}\varphi)^{\Delta} = ($🕕$v{:}\varphi)$ 📪 $($🕕$v{:}\varphi$📪$\varphi^{\Delta})$

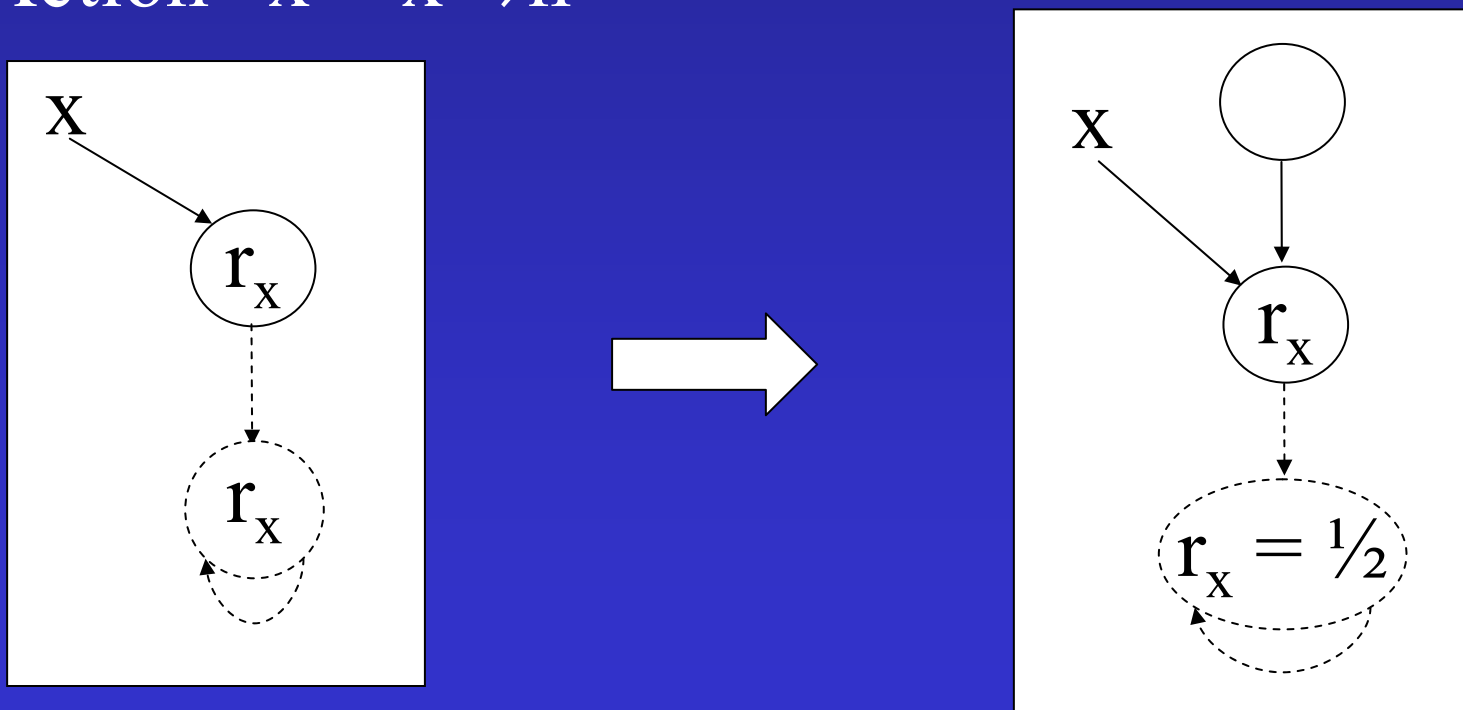
$(TC{:}\varphi)(v,w)^{\Delta} = (TC{:}\varphi)(v,w)$ 📪 $(TC{:}\varphi$📪$\varphi^{\Delta})(v,w)$

Formula Differencing Limitations

- Differencing output often imprecise

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

  Action "x = x→n"

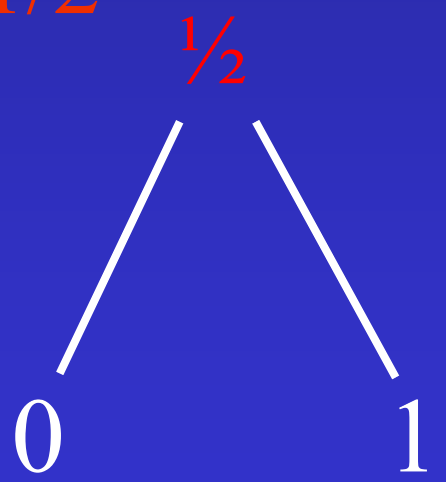


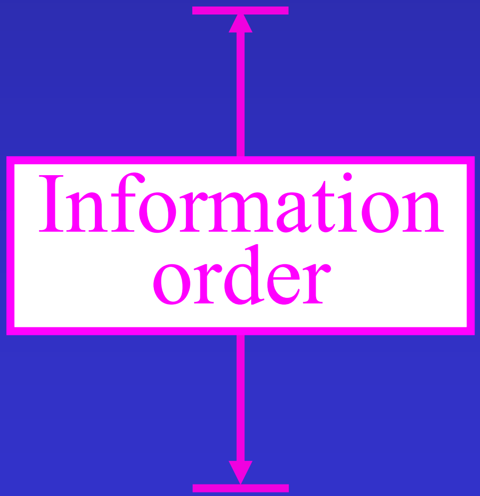
Reducing Loss of Precision from Differencing

- Semantic minimization of formulas
  - Simplest case: propositional formulas
- Work in progress
  - Removing unnecessary reevaluation
    - Finding common sub-formulas
  - Improvements to materialization
  - Exploiting important special cases
    - E.g., reachability in lists

Three-Valued Logic

- 1: True
- 0: False
- 1/2: Unknown
- A join semi-lattice: 0 ⌨ 1 = 1/2





0 ↖ ½

1 ↖ ½

Semantic Minimization

- ←φ→(A): Value of formula φ under assignment A
- In 3-valued logic, ←φ→(A) may equal ½

  ←$p + p'$→([p ♒ 0]) = 1

  ←$p + p'$→([p ♒ ½]) = ½

  ←$p + p'$→([p ♒ 1]) = 1

- However,

  ←1→([p ♒ 0]) = 1

  ←1→([p ♒ ½]) = 1

  ←1→([p ♒ 1]) = 1

Semantic Minimization

$\leftarrow 1 \rightarrow ([p$ ♒ $0]) = 1 = \leftarrow p + p' \rightarrow ([p$ ♒ $0])$ $\leftarrow 1 \rightarrow ([p$ ♒ $½]) = 1$ ↖ $½ = \leftarrow p + p' \rightarrow ([p$ ♒ $½])$ $\leftarrow 1 \rightarrow ([p$ ♒ $1]) = 1 = \leftarrow p + p' \rightarrow ([p$ ♒ $1])$

2-valued logic: 1 is equivalent to $p + p'$

3-valued logic: 1 is better than $p + p'$

For a given φ, is there a best formula?

Yes!

Minimal?

$x + x'$	No!
x ⌨ x'	Yes!
$xy + x'z$	No!
$xy + x'y'$	Yes!
$xy + x'z + yz$	Yes!
$xy' + x'z' + yz$	No!

Example

Original formula (φ)

$$xy + x'z$$

Minimal formula (ψ)

$$xy + x'z + yz$$

A	$\leftarrow\psi\rightarrow(A)$	$\leftarrow\varphi\rightarrow(A)$
[x ♒ ½, y ♒ 1, z ♒ 1]	1	½

Example

Original formula (φ)

$$xy' + x'z' + yz$$

Minimal formula (ψ)

$$x'y + x'z' + yz + xy' + xz + y'z'$$

A	←ψ→(A)	←φ→(A)
[x ♒ ½, y ♒ 0, z ♒ 0]	1	½
[x ♒ 0, y ♒ 1, z ♒ ½]	1	½
[x ♒ 1, y ♒ ½, z ♒ 1]	1	½

Semantic Minimization

- When φ contains no occurrences of ½ and ⌨

  📫 primes(←φ→)

- In general, somewhat more complicated
  - Represent ←φ→ with a pair

    floor: ⌊ ←φ→ ⌋ ⌊½⌋ = 0

    ceiling: ⌈ ←φ→ ⌉ ⌈½⌉ = 1

  - Semantically minimal formula

📫 primes(⌊ ←φ→ ⌋) ⌨ ¬(📫 primes(¬ ⌈

Current and Future Work

- Finite differencing of formulas
- Minimization of first-order formulas
- Generation of instrumentation predicates

Towards Abstraction Refinement in TVLA

Alexey Loginov
UW Madison
alexey@cs.wisc.edu