17-654
Analysis of Software Artifacts

Midterm Review

Jonathan Aldrich

Analysis as an Approximation

- If S doesn't terminate normally, y cannot be 0
- Problem: undecidable to tell if S terminates!
- In general program analysis must compute an approximation

Safety and Precision

- ***Conservative/Safe Analysis***
  - *Computes a larger set of possibilities than will actually occur in program execution*
- ***Precise Analysis***
  - *Computes as small a set of possibilities for program execution as it can*

Safety and Precision

- ***Conservative/Safe Analysis***
  - *Computes a larger set of possibilities than will actually occur in program execution*
- ***Precise Analysis***
  - *Computes as small a set of possibilities for program execution as it can*

Finding the Fixpoint

- Why should we think we will find an n such that $F^{n+1}(\overline{RD_\emptyset}) = F^n(\overline{RD_\emptyset})$?

Finding the Fixpoint

- Why should we think we will find an n such that $F^{n+1}(\overline{RD_\emptyset}) = F^n(\overline{RD_\emptyset})$?
  - F is monotone
  - Therefore, every application of F either:
    - Does not change $\overline{RD}$ (and so we have a fixpoint)
    - Or increases the size of a set in $\overline{RD}$
  - The set of definitions is finite so the sets in $\overline{RD}$ cannot increase in size forever
  - Therefore the algorithm terminates with a fixpoint at some finite n

Reaching Defs. vs. Available Exp.

- Reaching Defs. ***May*** **analysis**
  - Initial dataflow values: *empty* sets
  - *Union* at control flow merge
  - Precision: want *least* fixed point
  - Safety: err on the side of *larger* sets
- Available Exp. ***Must*** **analysis**
  - Initial dataflow values: *universal* sets
  - *Intersection* at control flow merge
  - Precision: want *greatest* fixed point
  - Safety: err on the side of *smaller* sets

Monotone Framework

$$\text{Analysis}_\circ(\ell) = \iota \quad \text{if } \ell \in E$$
$$\qquad = \bigsqcup \{ \text{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F \} \quad \text{otherwise}$$

$$\text{Analysis}_\bullet(\ell) = f_\ell(\text{Analysis}_\circ(\ell))$$

where:

- $\circ$ means entry (forward) or exit (backward)
- $\bullet$ means exit (forward) or entry (backward)
- $\sqcup$ is $\cup$ (may) or $\cap$ (must)
- F is $flow(S_*)$ (forward) or $flow^R(S_*)$ (backward)
- E is $\{ init(S_*) \}$ (forward) or $final(S_*)$ (backward)
- ι specifies initial or final analysis information, and
- f_ℓ is a transfer function
  - Typically $f_\ell(x) = x \setminus kill_{Analysis}(B^\ell) \cup gen_{Analysis}(B^\ell)$

Monotone Framework

$$\text{Analysis}_\circ(\ell) = \iota \quad \text{if } \ell \in E$$
$$\qquad = \bigsqcup \{ \text{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F \} \quad \text{otherwise}$$

$$\text{Analysis}_\bullet(\ell) = f_\ell(\text{Analysis}_\circ(\ell))$$

	RD	AE	LV
$\sqcup$			
F			
E			
ι			

Monotone Framework

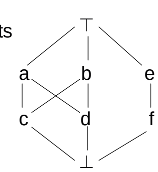
$$\text{Analysis}_\circ(\ell) = \iota \quad \text{if } \ell \in E$$
$$\qquad = \bigsqcup \{ \text{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F \} \quad \text{otherwise}$$

$$\text{Analysis}_\bullet(\ell) = f_\ell(\text{Analysis}_\circ(\ell))$$

	RD	AE	LV
$\sqcup$	$\cup$	$\cap$	$\cup$
F	$flow(S_*)$	$flow(S_*)$	$flow^R(S_*)$
E	$\{ init(S_*) \}$	$\{ init(S_*) \}$	$final(S_*)$
ι	$\{ (x,?) \mid x \in FV(S_*) \}$	$\varnothing$	$\varnothing$

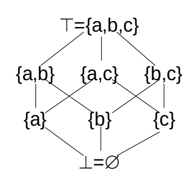
Complete Lattice

- Not all data flow analyses use sets
  - Lattice: a more general concept



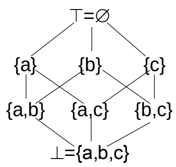
- A set L with:
  - A partial order $\sqsubseteq$
  - A combination operator $\sqcup$
  - A least element $\bot = \sqcup(\varnothing)$
  - A greatest element $\top = \sqcup(L)$
  - Each subset Y of L has a least upper bound $\sqcup(Y)$
- Typically we want the lattice to have finite height
  - A finite number of elements on each path from $\bot$ to $\top$
    - See NNH Appendix A.3

Example: Subset Lattice



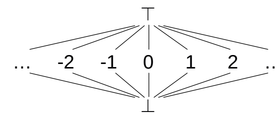
- Reaching Definitions
- The set $L=\mathcal{P}(\{a,b,c\})$ with:
  - $\sqsubseteq = \subseteq$
  - $\sqcup = \cup$ (may analysis)
  - $\bot = \varnothing$ (the most precise and starting element)
  - $\top = \{a,b,c\}$ (the least precise element)

Example: Superset Lattice



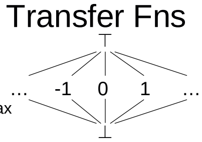
- Available Expressions
- The set $L=\mathcal{P}(\{a,b,c\})$ with:
  - $\sqsubseteq\ = \supseteq$
  - $\sqcup = \cap$ (must analysis)
  - $\bot = \{a,b,c\}$ (the most precise and starting element)
  - $\top = \emptyset$ (the least precise element)

Constant Propagation Lattice



- More efficient than the set of possible values
  - Don't want to store sets
  - If more than one value, give up and assume any ($\top$)
- The set $L=\{\bot,\top\} \cup$ NAT with:
  - $x \sqsubseteq \top$, $\bot \sqsubseteq x$, $x \sqsubseteq x$
  - $x \sqcup \bot = x$, $x \sqcup \top = \top$, $n \sqcup m = \top$ (for $n \neq m$)
- $\iota = \top$

Constant Propagation Transfer Fns



- Can't use gen and kill sets
  - Data flow values aren't sets anymore!
- Instead, define function by cases on syntax
  - Input is incoming data flow value σ

- $f^{CP}([x := a],\sigma) = \sigma\,[x \mapsto CP(a,\sigma)]$
- $f^{CP}([skip],\sigma) = \sigma$
- $f^{CP}([b],\sigma) = \sigma$

- $CP(n,\sigma) = n$
- $CP(x,\sigma) = \sigma(x)$
- $CP(a_1\ op_a\ a_2,\sigma) = CP(a_1, \sigma)\ \widehat{op_a}\ CP(a_2, \sigma)$

- $z_1\ \widehat{op_a}\ z_2$ $= z_1\ \widehat{op_a}\ z_2$ if $z_1, z_2 \in$ NAT
  $= \top$ if $z_1 = \top$ or $z_2 = \top$
  $= z_1\,(z_2)$ if $z_2\,(z_1) = \bot$

Execution Traces

- Sequence of <pp,mem> pairs
  - pp is a program point
    - Just before statement pp
  - mem is the state of variables in memory

[y := x]^1;
[z := 1]^2;
while [y>1]^3 do
  [z := z * y]^4;
  [y := y – 1]^5;
[y := 0]^6;

pp	x	y	z
1	2	0	0
2	2	2	0
3	2	2	1
4	2	2	1
5	2	2	2
3	2	1	2
6	2	1	2
-	2	0	2

Execution Traces

- Sequence of <pp,mem> pairs
  - pp is a program point
    - Just before statement pp
  - mem is the state of variables in memory

[y := x]^1;
[z := 1]^2;
while [y>1]^3 do
  [z := z * y]^4;
  [y := y – 1]^5;
[y := 0]^6;

pp	x	y	z
1	1	0	0
2	1	1	0
3	1	1	1
6	1	1	1
-	1	0	1

Execution Traces

- Sequence of <pp,mem> pairs
  - pp is a program point
    - Just before statement pp
  - mem is the state of variables in memory

[y := x]^1;
[z := 1]^2;
while [y>1]^3 do
  [z := z * y]^4;
  [y := y – 1]^5;
[y := 0]^6;

pp	x	y	z
1	3	0	0
2	3	3	0
3	3	3	1
4	3	3	1
5	3	3	3
3	3	2	3
4	3	2	3
5	3	2	6
3	3	1	6
6	3	1	6
-	3	0	6

Execution Traces

- Sequence of <pp,mem> pairs
  - pp is a program point
    - Just before statement pp
  - mem is the state of variables in memory

$[y := x]^1$;
$[z := 1]^2$;
while $[y>1]^3$ do
  $[z := z * y]^4$;
  $[y := y – 1]^5$;
$[y := 0]^6$;

pp	x	y	z

Repeat for all possible initial values of x,y,z!

Abstraction

- Abstraction function α
  - maps traces to data flow values at a certain time *t* in the trace
- $\alpha_{CP}(<p_1,m_1>\ldots<p_n,m_n>,t)$
  $= m_t$
- $\alpha_{SA}(<p_1,m_1>\ldots<p_n,m_n>,t)$
  $= sign(m_t)$
- Also define program point function *pp*
- $pp(<p_1,m_1>\ldots<p_n,m_n>,t)$
  $= p_t$

t	pp	x	y	z
0	1	3	0	0
1	2	3	3	0
2	3	3	3	1
3	4	3	3	1
4	5	3	3	3
5	3	3	2	3
6	4	3	2	3
7	5	3	2	6
8	3	3	1	6
9	6	3	1	6
10	-	3	0	6

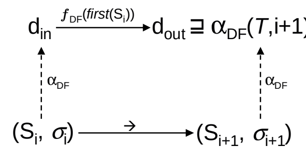
$\alpha_{CP}(T,0) = (x=3,y=0,z=0)$
$\alpha_{CP}(T,10) = (x=3,y=0,z=6)$

$\alpha_{SA}(T,10) = (x=+,y=0,z=+)$

WHILE Traces, Formally

- A trace for program S_1 and initial state σ_1 is either:
  - a finite sequence $(S_1, \sigma_1), \ldots, ([], \sigma_n)$, where $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1})$ for $i \in 1, \ldots, n-1$
  - an infinite sequence $(S_1, \sigma_1), \ldots, (S_i, \sigma_i), \ldots$ where $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1})$ for $i \geq 1$
- Slight notational simplification
  - We will abbreviate $(S_1, \sigma_1), \ldots, (S_n, \sigma_n)$ as $(first(S_1), \sigma_1), \ldots, (first(S_n), \sigma_n)$
    - Uses program counter labels instead of complete programs

Local Soundness



To show:
  if $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$
  and $d_{in} = \alpha_{DF}(T,i)$
  and $d_{out} = f_{DF}(first(S_i), d_{in})$
  then $\alpha_{DF}(T,i+1) \sqsubseteq d_{out}$

Intuitively, translating from concrete to abstract and applying the flow function will safely approximate ($\sqsupseteq$) taking a step in the trace and translating from concrete to abstract

What does Correctness Mean?

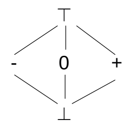
- Intuition
  - At a fixed point, analysis results are a *conservative abstraction* of *program execution*
- *Soundness* condition
  - When data flow analysis reaches a fixed point *F*, then for all traces *T* and all times *t* in each trace, $\alpha(T,t) \sqsubseteq F(pp(T,t))$

Global Soundness

- Intuition
  - We begin with initial dataflow facts ι that safely approximate ($\sqsupseteq$) all initial stores σ_1
  - By local soundness, each transfer function when given safe input information yields safe output information
  - By induction, any fixed point of the analysis is sound

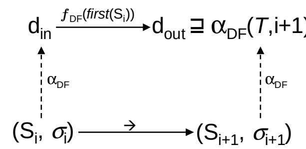
Soundness Example: Sign Analysis

Custom Lattice



- Transfer functions
  - *σ is input data flow value*
  - f^{SA}([x := a],σ) = σ [x ↦ SA(a,σ)]
  - f^{SA}([skip],σ) = σ
  - f^{SA}([b],σ) = σ
  - SA(n,σ) = *sign*(n) *// returns sign of n*
  - SA(x,σ) = σ(x)
  - SA(a_1 + a_2,σ) = + ***// is this sound?***
  - SA(a_1 op_a a_2,σ) = ⊤ *for op_a ≠ +*

Local Soundness

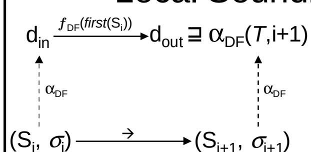


To show:

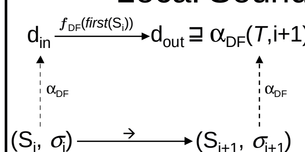
if (S_i, σ_i) → (S_{i+1}, σ_{i+1}) ∈ *T*
and $d_{in} = \alpha_{DF}(T,i)$
and $d_{out} = f_{DF}(first(S_i), d_{in})$
then $\alpha_{DF}(T,i+1) \sqsubseteq d_{out}$

Intuitively, translating from concrete to abstract and applying the flow function will safely approximate (⊒) taking a step in the trace and translating from concrete to abstract

Local Soundness Fails

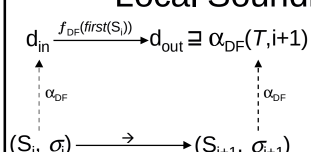


Local Soundness Fails



Program:
[z := y-7][1]
[x := y+z][2]

Local Soundness Fails

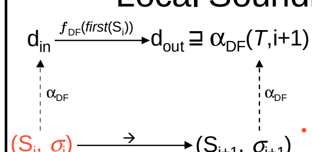


Program:
[z := y-7][1]
[x := y+z][2]

Trace *T*:

t	pp	x	y	z
0	1	0	3	0
1	2	0	3	-4
2	-	-1	3	-4

Local Soundness Fails



Program:
[z := y-7][1]
[x := y+z][2]

- σ_1 = (x=0, y=3, z=-4)

Trace *T*:

t	pp	x	y	z
0	1	0	3	0
1	2	0	3	-4
2	-	-1	3	-4

Local Soundness Fails

$d_{in} \xrightarrow{f_{DF}(first(S_i))} d_{out} \sqsupseteq \alpha_{DF}(T,i+1)$

$\uparrow \alpha_{DF} \qquad \uparrow \alpha_{DF}$

$(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1})$

Program:
[z := y-7]^1
[x := y+z]^2
[done]

- σ_1 = (x=0, y=3, z=-4)
- σ_2 = (x=-1, y=3, z=-4)

Trace *T*:

t	pp	x	y	z
0	1	0	3	0
1	2	0	3	-4
2	-	-1	3	-4

Local Soundness Fails

$d_{in} \xrightarrow{f_{DF}(first(S_i))} d_{out} \sqsupseteq \alpha_{DF}(T,i+1)$

$\uparrow \alpha_{DF} \qquad \uparrow \alpha_{DF}$

$(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1})$

Program:
[z := y-7]^1
[x := y+z]^2

- σ_1 = (x=0, y=3, z=-4)
- σ_2 = (x=-1, y=3, z=-4)
- $d_{in} = \alpha_{DF}(T,1)$=(x=0, y=+, z=-)

Trace *T*:

t	pp	x	y	z
0	1	0	3	0
1	2	0	3	-4
2	-	-1	3	-4

Local Soundness Fails

$d_{in} \xrightarrow{f_{DF}(first(S_i))} d_{out} \sqsupseteq \alpha_{DF}(T,i+1)$

$\uparrow \alpha_{DF} \qquad \uparrow \alpha_{DF}$

$(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1})$

Program:
[z := y-7]^1
[x := y+z]^2

- σ_1 = (x=0, y=3, z=-4)
- σ_2 = (x=-1, y=3, z=-4)
- $d_{in} = \alpha_{DF}(T,1)$=(x=0, y=+, z=-)
- $\alpha_{DF}(T,2)$ = (x=-, y=+, z=-)

Trace *T*:

t	pp	x	y	z
0	1	0	3	0
1	2	0	3	-4
2	-	-1	3	-4

Local Soundness Fails

$d_{in} \xrightarrow{f_{DF}(first(S_i))} d_{out} \sqsupseteq \alpha_{DF}(T,i+1)$

$\uparrow \alpha_{DF} \qquad \uparrow \alpha_{DF}$

$(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1})$

Program:
[z := y-7]^1
[x := y+z]^2

- σ_1 = (x=0, y=3, z=-4)
- σ_2 = (x=-1, y=3, z=-4)
- $d_{in} = \alpha_{DF}(T,1)$=(x=0, y=+, z=-)
- $\alpha_{DF}(T,2)$ = (x=-, y=+, z=-)
- $d_{out} = f^{SA}$([x := y+z]^2, d_{in})
  $= d_{in}$[x ↦ SA(y+z, d_{in})]
  $= d_{in}$[x ↦ +]
  = (x=+, y=+, z=-)

Trace *T*:

t	pp	x	y	z
0	1	0	3	0
1	2	0	3	-4
2	-	-1	3	-4

Local Soundness Fails

$d_{in} \xrightarrow{f_{DF}(first(S_i))} d_{out} \sqsupseteq \alpha_{DF}(T,i+1)$

$\uparrow \alpha_{DF} \qquad \uparrow \alpha_{DF}$

$(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1})$

Program:
[z := y-7]^1
[x := y+z]^2

- σ_1 = (x=0, y=3, z=-4)
- σ_2 = (x=-1, y=3, z=-4)
- $d_{in} = \alpha_{DF}(T,1)$=(x=0, y=+, z=-)
- $\alpha_{DF}(T,2)$ = (x=-, y=+, z=-)
- $d_{out} = f^{SA}$([x := y+z]^2, d_{in})
  $= d_{in}$[x ↦ SA(y+z, d_{in})]
  $= d_{in}$[x ↦ +]
  = (x=+, y=+, z=-)
- (x=+, y=+, z=-) $\not\sqsupseteq$ (x=-, y=+, z=-)

Trace *T*:

t	pp	x	y	z
0	1	0	3	0
1	2	0	3	-4
2	-	-1	3	-4

Local Soundness for Constant Propagation

To show:
if $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$
and $d_{in} = \alpha_{CP}(T,i)$
and $d_{out} = f_{CP}(first(S_i), d_{in})$
then $\alpha_{CP}(T,i+1) \sqsubseteq d_{out}$

- Case: S_i = [x := a]$^\ell$
  - $\sigma_{i+1} = \sigma_i$[x ↦ $\mathcal{A}$(a, σ_i)]
  - $d_{in} = \alpha_{CP}(T,i) = \sigma_i$
  - $d_{out} = f_{CP}$([x := a]$^\ell$, σ_i)
    $= \sigma_i$[x ↦ CP(a, σ_i)]
  - $\alpha_{CP}(T,i+1) = \sigma_{i+1}$
    $= \sigma_i$[x ↦ $\mathcal{A}$(a, σ_i)]
  - Lemma: $\mathcal{A}$(a, σ_i) = CP(a, σ_i)
  - Thus σ_i[x ↦ $\mathcal{A}$(a, σ_i)]
    $\sqsubseteq \sigma_i$[x ↦ CP(a, σ_i)]

Abstraction for Reaching Definitions

- $\alpha_{RD}(\langle p_1,m_1\rangle \ldots \langle p_n,m_n\rangle, t) =$
  $\{ (x, p_k) \mid x \in FV(S_*)$
  and $k < t$
  and $stmt(p_k) = [x := a]$
  and $\forall j,\ k<j<t\ stmt(p_j) \neq [x := a']\}$

Local Soundness for Reaching Definitions

To show:

if $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$
and $d_{in} = \alpha_{RD}(T,i)$
and $d_{out} = f_{RD}(first(S_i), d_{in})$
then $\alpha_{RD}(T,i+1) \sqsubseteq d_{out}$

- Case: $S_i = [x := a]^\ell$
  - $d_{in} = \alpha_{RD}(T,i)$
  - $d_{out} = f_{RD}([x := a]^\ell, d_{in})$
    $= (\alpha_{RD}(T,i) \setminus \{(x,*)\}) \cup \{(x,\ell)\}$
  - Lemma: $\alpha_{RD}(T,i+1)$
    $= (\alpha_{RD}(T,i) \setminus \{(x,*)\}) \cup \{(x,\ell)\}$
  - So $\alpha_{RD}(T,i+1) = d_{out}$
  - Thus $\alpha_{RD}(T,i+1) \sqsubseteq d_{out}$

Abstraction for Live Variables

- $\alpha_{LV}(\langle p_1,m_1\rangle \ldots \langle p_n,m_n\rangle, t) =$
  $\{ x \mid x \in FV(stmt(p_k))$ where $k > t$
  and $\forall j,\ t<j<k\ stmt(p_j) \neq [x := a']\}$

Local Soundness for Live Variables

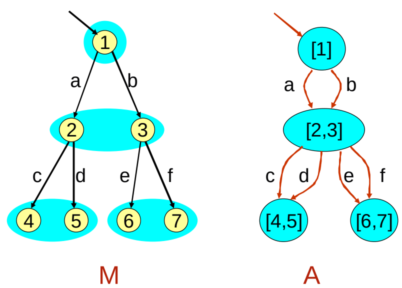
To show:

if $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$
and $d_{in} = \alpha_{LV}(T,i+1)$
and $d_{out} = f_{LV}(first(S_{i+1}), d_{in})$
then $\alpha_{LV}(T,i) \sqsubseteq d_{out}$

Note: i and i+1 are swapped due to reverse analysis

- Case: $S_{i+1} = [x := a]^\ell$
  - $d_{in} = \alpha_{RD}(T,i+1)$
  - $d_{out} = f_{RD}([x := a]^\ell, d_{in})$
    $= (\alpha_{RD}(T,i+1) \setminus \{x\}) \cup FV(a)$
  - Lemma: $\alpha_{RD}(T,i)$
    $= (\alpha_{RD}(T,i+1) \setminus \{x\}) \cup FV(a)$
  - So $\alpha_{RD}(T,i) = d_{out}$
  - Thus $\alpha_{RD}(T,i) \sqsubseteq d_{out}$

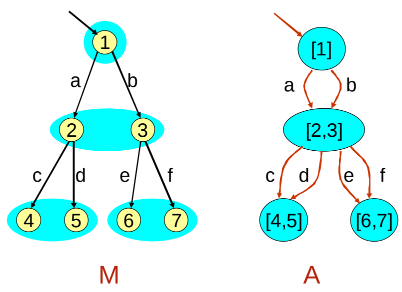
Conservative Abstraction



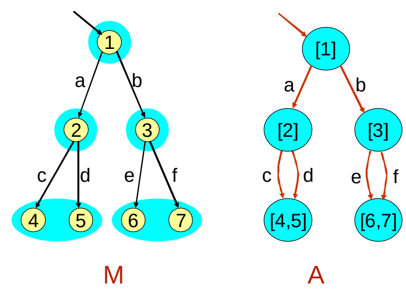
Conservative Abstraction

- Every trace of M is a trace of A
  - A over-approximates what M can do
    (Preserves safety properties!): $A \models \phi \Rightarrow M \models \phi$
- Some traces of A may not be traces of M
  - May yield spurious counterexamples - $\langle$ a, e $\rangle$
- Eliminated via abstraction refinement
  - Splitting some clusters in smaller ones
  - Refinement can be automated

Original Abstraction

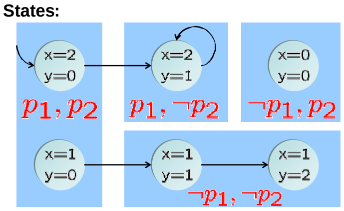


Refined Abstraction



Predicate Abstraction

Concrete States:



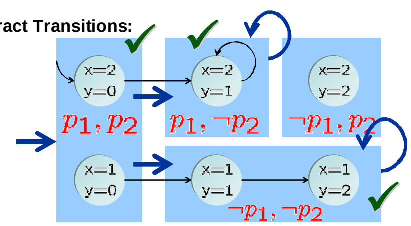
Predicates:

$p_1(s) = (s.x > s.y)$
$p_2(s) = (s.y = 0)$

Abstract transitions?

Predicate Abstraction

Abstract Transitions:



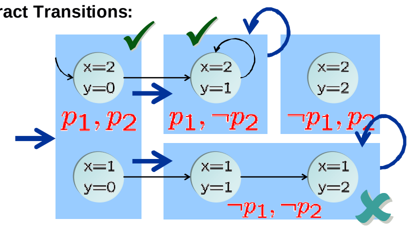
Property:

$p_1 \lor \neg p_2 \iff (s.x > s.y) \lor (s.y \neq 0)$

Property holds. Ok.

Predicate Abstraction

Abstract Transitions:



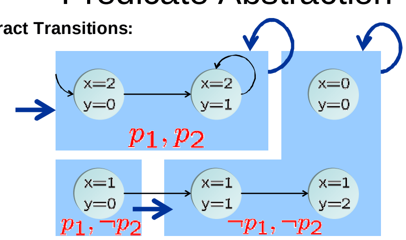
Property:

$p_1 \iff (s.x > s.y)$

This trace is spurious!

Predicate Abstraction

Abstract Transitions:



Property:

$p_1 \iff (s.x > s.y)$ ✓

New Predicates:

$p_1(s) = (s.x > s.y)$
$p_2(s) = (s.x = 2)$